실시간 CORBA상에서의 신뢰성 지원을 위한 결함허용 서비스의 설계 및 구현

김 영 재[†] · 박 기 용[††] · 홍 성 준[†††] · 한 선 영[††††]

요 약

현재 CORBA는 분산환경에서 소프트웨어의 복잡성을 해결하기위한 방안으로 제시되어왔으나 데이터 전송에 대한 신뢰성 보장성이 부족하다. 특히 네트워크이 불안하거나 정체 되었을 때 이를 처리해주는 메카니즘이 없는 실정이다. 기존 CORBA에서는 클라이언트가 서버객체가 가지고 있는 객체를 서비스 받을 때, 네트워크상의 정체가 있는 경우 클라이언트는 무조건 기다려야 하는 단점이 있다. 이에 본 논문은 이러한 정체 시 요구된 서비스 객체를 다른 서버 객체가 지능적으로 우회 시켜주는 결함허용 서비스(Fault-Tolerant CORBA Service)를 제안한다. 이를 위해 본 논문은 신뢰성 지원을 위한 결함허용 서비스를 CORBA ORB에 확장하여 설계 및 구현하였다. 그러므로 제안된 결함허용 서비스를 통해 신뢰성 있는 연결을 보장할 수 있다.

키워드 : 실시간 CORBA, 결함허용 서비스, 네이밍서비스, QoS

Design and implementation of Fault-Tolerant CORBA Service for reliability on Real-Time CORBA

Young-Jae Kim[†] · Ki-Young Park[††] · Sung-June hong[†††] · Sun-Young Han[††††]

ABSTRACT

Current CORBA has been suggested a solution for complexity of software in distributed environment. But it can't provide reliability about data transfer. For instance, CORBA can't provide object to client when the network are unstable or congested. In an existing CORBA, client has to wait when there is congestion between client and object implementation while client get a service from object implementation. So In this paper, we propose Fault-Tolerant CORBA Service (FTS) which has an intelligent redirection about an object that client requests. Moreover, we designed and implemented Fault-Tolerant Service to provide reliability by extending existing CORBA ORB. Therefore it provides reliable connection throughout the proposed Fault-Tolerant Service.

Key word : Real-Time CORBA, Fault-Tolerant CORBA, Naming Service, QoS

1. 서 론

기존의 시스템이 점점 대규모화 대고 확장이 됨에 따라 클라이언트 서버 모델을 기반으로 한 분산 네트워크 환경에서 프로그램 구현 시 소프트웨어의 복잡성의 문제가 야기되고 있다. 이런 소프트웨어의 복잡성 문제를 해결하기 위해서 최근에 CORBA (Common Object Request Broker Architecture) [1]나 DCOM (Distributed Component Object Model) [2], EJB(Enterprise Java beans) [2]등의 기술이 사용되고 있다. 여기서 CORBA는 분산환경에서 적용될 수 있는 컴퓨팅 표준이며, 분산환경에서 시스템을 통합하는 미들웨어를 말한다.

한편 사용자는 분산 멀티미디어 데이터 전송 시 보다 양질의 서비스를 요구하고 있으며 특히 QoS지원의 요구 와 신뢰성의 지원 요구가 대두되고 있다. 이러한 문제를 해결하기 위해서 OMG 그룹에서는 실시간 CORBA(Real-Time CORBA)[3]와 결함허용 CORBA (Fault-Tolerant CORBA) [16]에 등의 표준을 정의하는 노력을 진행되고 있다. 실시간 CORBA SIG(Special Interest Group)은 실시간 CORBA에 대한 정의를 하고 있는 그룹으로 시간 제약성(Timing Constraints), QoS (Quality of Service), 스케쥴링, 자원예약(Resource Reservation)등의 속성을 가져야 한다고 정의하고 있다. 결함 허용 CORBA는 높은 신뢰성을 요구하는 응용 프로그램에 대해 인터페이스, 정책, 그리고 서비스의 표준을 정의하고 있다. 결함허용 CORBA에서 결함발생을 인식하고 복구하는데 사용되는 결함허용 메커니즘은 객체 중복성, 결함인식, 그리고 복구등에 달려있다.

※ 본 논문은 1998년도 한국과학재단 연구비지원(과제번호 981-0917-088-2)에 의한 결과임.
† 준 회 원 : 건국대학교 대학원 컴퓨터정보통신공학과
†† 준 회 원 : 건국대학교 컴퓨터정보통신공학과
††† 정 회 원 : 여주대학 정보통신과 교수
†††† 정 회 원 : 건국대학교 정보통신원 원장
논문접수 : 2000년 8월 16일, 심사완료 : 2001년 1월 5일

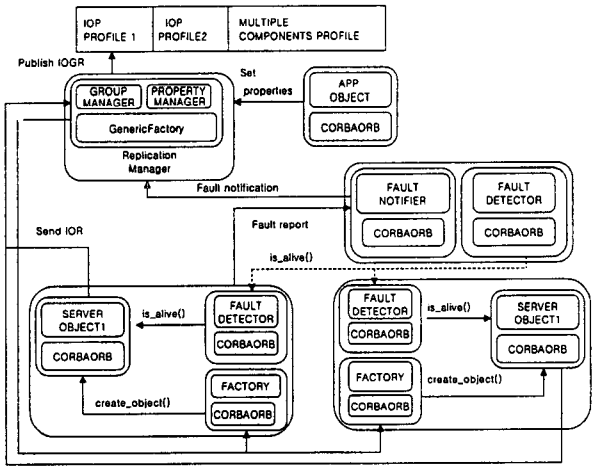
이런 실시간 CORBA와 결함허용 CORBA표준의 노력과 함께 워싱턴 대학의 TAO(The ACE ORB)[3]라고 명명된 실시간 CORBA와 DOORS(Distributed Object-Oriented Reliable Services)[16]라고 명명된 결함허용 CORBA등의 개발이 진행 중에 있다. 그러나 기존의 결함허용의 CORBA 표준은 아직 다른 벤더간의 상호 운용성을 지원하지 못하고 있으며 중복객체의 강한 일관성 유지를 처리하는 결정 행위(Deterministic Behavior) 등의 아직 보완해야 많은 요소를 가지고 있다.

그러므로 본 논문에서는 기존 결함허용 CORBA표준에서 지원하지 못하는 요소들을 분석하고 그 중에서 특히 구현 객체가 서비스를 제대로 제공하지 못할 때 다른 구현 객체를 통해 서비스하는 지능적 우회(Intelligent Redirection) 기능을 지원하는 결함허용 CORBA 서비스를 설계하였고 CORBA ORB내부에 구현에 대하여 언급하였다.

본 논문의 구성은 다음과 같다. 2장은 결함허용 CORBA에 대한 설명을 하였고, 3장은 지능적 우회를 지원하는 결함허용 CORBA서비스의 설계에 관하여 설명하였고, 4장은 지능적 우회 능력을 지원하는 결함허용 CORBA서비스의 구현에 관하여 언급하였다. 그리고 5장에서는 개발 결과를 언급하였고, 6장에서 개발에 대한 실험, 그리고 7장을 본 논문의 결론을 언급 하였다.

2. 결함 허용 CORBA

CORBA객체에 대한 결함허용은 중복(replication), 결함탐지(fault detection)과 복구(recovery)로 이루어진다. CORBA의 중복객체는 서버측 객체의 생성과 관리로 이루어진다. 이런 정책은 중복객체의 구성관리에 큰 융통성을 가진다.



(그림 1) 결함 허용 CORBA의 구조

(그림 1)은 결함허용 CORBA의 전체적인 구조를 보이고 있다. (그림 1)에서 보이는 모든 요소는 표준 CORBA객체로 구현되었다. 즉 CORBA객체는 CORBA IDL인터페이스를 사용해서 정의되고 java나 C++과 같은 표준 프로그래밍 언어로 쓰여진 서버 객체를 이용해서 구현된다. 각 요소의 기능은 IOGR(Interoperable Object Group Reference), 중복관리자, 탐지자(Detector)와 통보자(Notifier), 그리고 로깅과 복구 등이 있다.

IOGR를 살펴보면, 결함허용CORBA는 개별적인 중복 객체에 대해서 사용된 IOR(Interoperable Object Reference)의 형식의 표준화를 따른다. IOR은 CORBA객체를 유일하게 식별하게 하는 융통성 있는 주소 메커니즘이다. 또한 IOGR (Interoperable Object Group Reference)은 복합객체에 대한 IOR을 정의한다. IOGR는 서버 객체 그룹에 도착하기 위해서 사용된 다중 TAG_INTERNET_P 프로파일을 포함한다. 결함허용 CORBA 서버는 IOGR를 클라이언트들에게 알릴 수 있다. 클라이언트는 IOGR을 사용해서 서버의 오퍼레이션을 동작시킨다. 클라이언트 ORB는 요청을 처리하는 적당한 서버 객체에게 요청을 보낸다. 이때 클라이언트는 서버 중복객체의 존재를 알지 못한다. 서버 객체가 실패하면, 클라이언트 ORB는 IOR에 포함된 객체 참조자를 통해서 연결되어 동작한다. 만일 모든 서버 객체가 실패하면 IOGR에 참조자들은 유효하지 않은 것이고 그런 경우에 예외처리 메시지가 클라이언트에게 전송된다.

중복관리자(Replication Manager)를 살펴보면, 속성관리자 (Property Manager), GenericFactory 그리고 ObjectGroup 등의 3가지 요소를 포함한다.

속성관리자는 객체 그룹의 속성이 선택되게 한다. 공통적인 요소는 중복 스타일(Replication style), 멤버스타일(membership style), 일관성 스타일(consistency style), 중복객체의 최소 숫자 등을 포함한다. 중복 스타일의 예는 COLD_PASSIVE, WORM_ PASSIVE, ACTIVE등이 있다. COLD_PASSIVE는 중복스타일에서 중복 그룹은 클라이언트 메시지에 응답하는 Primary 중복 객체를 포함한다. Primary 객체가 실패하면, backup 중복 객체가 새 primary 중복 객체로 생성된다. WORM_PASSIVE는 중복 그룹이 클라이언트 메시지에 응답하는 Primary 중복 객체를 포함한다. 추가적으로 하나 이상의 back-up 중복 객체가 실패를 없애기 위해서 생성된다. ACTIVE는 모든 중복객체가 Primary이다. 각각은 서로 독립적으로 클라이언트 요청을 처리한다. 하나의 응답이 클라이언트에 보내지고 중복 객체가운데 일관성 상태를 유지하기 위해서 특별한 그룹 통신 프로토콜이 필요하다.

하부구조 조절 멤버스타일(Infrastructure-controlled membership style)에 대해서 중복관리자(ReplicationManager)는 GenericFatory를 사용해서 객체그룹을 만든다. 응용 프로그램 조절 멤버쉽 스타일에 대해서 응용 프로그램은 객체그룹(ObjectGroup)관리자를 사용해서 사용자 그룹의 멤버를 생성하고 추가하고 삭제한다.

탐지자(Detector) 와 통보자(Notifier)를 살펴보면, 결함 탐지자는 pull기반 메커니즘을 통해서 결함을 인식하는 책임이 있는 CORBA 객체이다. Pull 기반 모니터링 메커니즘은 객체가 살아있는지 알기 위해서 주기적으로 응용 프로그램을 호출한다. 결함허용 CORBA는 'is_alive' 동작을 하는 pull-Monitable인터페이스를 구현하기 위한 응용 객체를 필요로 한다.

결함 탐지자(Fault Detector)는 결함 통보자(Fault Notifier)에게 결함을 보고한다. 이번에 결함 통보자는 이런 보고를 중복 관리자에게 알린다. 중복관리자는 복구 행동을 시작한다. 주기적으로 결함 활동을 관찰하는데 관심이 있는 시스템의 다른 응용 프로그램은 결함 통보자에게 등록해서 이벤트를 얻는다. 복잡한 응용 프로그램은 결함보고를 확장하고 분석하기 위해서 결함 분석자(Analizer)를 제공한다.

로깅과 복구를 살펴보면, 응용프로그램 조절 일관성 스타일에 대해서 응용 프로그램은 자신의 결함 복구에 대해서 책임이 있다. 하부 구조 조절 일관성 스타일에 대해서 결함허용 CORBA는 로깅과 복구 메커니즘을 정의한다. 이 메커니즘은 클라이언트로부터 서버까지 CORBA GIOP 메시지를 기초해서 로깅하는 책임이 있다.

3. 지능적 우회(Intelligent Redirection) 기능 지원 결함허용 CORBA서비스의 설계

결함허용 CORBA 표준의 요구사항을 살펴보면 다음과 같다.

첫째, 하부구조 조절 일관성 스타일에 대해서 중복 객체의 행동이 비 중복 객체와 같아야 한다.

둘째, CORBA객체 참조모델의 향상이 있어야 한다. 즉 결함허용 CORBA표준은 3가지 새로운 태그된(tagged)요소가 객체 표준 모델에 추가되었다. 태그된 요소는 중복 객체 참조자에 다중 요소를 언급하는데 사용된다.

셋째, 결함허용 CORBA는 분산 객체 내에서 'single point of failure'가 없어야 한다.

넷째, 결함허용 CORBA표준은 중복 매니저(Replication Manager)에게 시간제한의 탐지(detection)와 통보(notification)를 요구한다.

다섯째, 실패 복구와 중복 객체 그룹의 크기 관리는 클라이언트가 투명하게 객체 그룹에 요청한다. 클라이언트 어플리케이션은 중복객체인지 아닌지에 대해서 구별한 필요가 없다.

여섯째, 클라이언트 ORB는 중복객체에게 요청을 보내는데 IOGR을 정의한다. 결함이 발생하면, 클라이언트 ORB는 요청을 다른 ORB에게 우회한다. 클라이언트 ORB는 요청이 성공할 때 까지 요청을 반복한다. 이런 우회와 요청의 발생에 대해서 클라이언트는 몰라도 된다.

이러한 결함허용 CORBA의 결함허용 표준의 제약성을 살펴보면 다음과 같다.

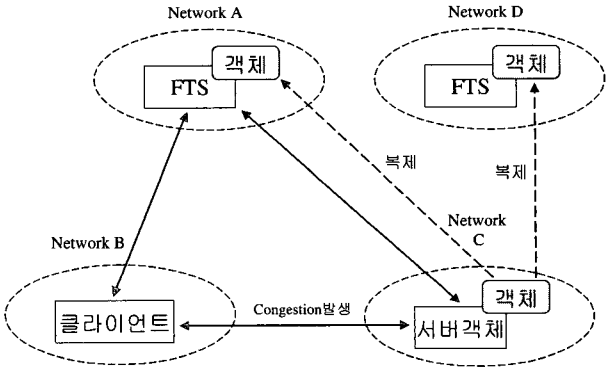
첫째, 결함허용 지원 ORB를 구현하는 벤더간의 상호 연동이 안 되는 점이다.

둘째, 중복 객체간의 데이터 일치성이 요구되는 객체에 대하여 결정 행위(Deterministic Behavior)가 요구된다.

셋째, 부분적인 망 결함에 대한 처리 메커니즘이 없다.

이에 본 논문은 위의 결함허용 CORBA 표준 중 세 번째 분산 객체 내에서의 'single point of failure'과 같은 결함허용을 지원하는 지능적 우회 서비스를 제안했다. 예를 들어, 클라이언트가 서버 객체가 가지고 있는 객체(Object)를 서비스 받을 때 정체(Congestion)가 있는 경우 클라이언트는 무조건 기다려야 하는 단점을 가지고 있다. 이러한 문제를 해결하기 위해 신뢰성을 지원하는 결함허용 서비스를 다음과 같이 네트워크상의 정체(Congestion)에 대한 지능적 우회 서비스를 설계했다.

(그림 2) 는 실시간 CORBA상에서 결함 허용 서비스 구조를 보여주고 있다. 클라이언트와 서버간에서 각각의 네트워크에는 결함 허용 서버가 있다. (그림 3)과 같이 객체 서버는 네이밍 서비스(Naming Service)에 객체를 등록해야 한다. 이때 이미 각 결함허용 서버는 서버객체가 가지고 있는 객체를 복제해서 가지고 있다. 그러므로, 클라이언트는 서버와 연결 시 네트워크상의 정체(congestion)를 확인하면 서버 객체가 가지고 있는 객체에 대해 서비스를 받기 위해 네이밍 서비스를 통해 등록되어 있는 결함허용 CORBA서버 중 가장 빠른 서버를 찾는다. 이렇게 알게 된 가장 빠른 결함 허용 서버로부터 자신이 필요한 서버이름과 객체이름에 대한 정보를 넘겨 받게 된다.



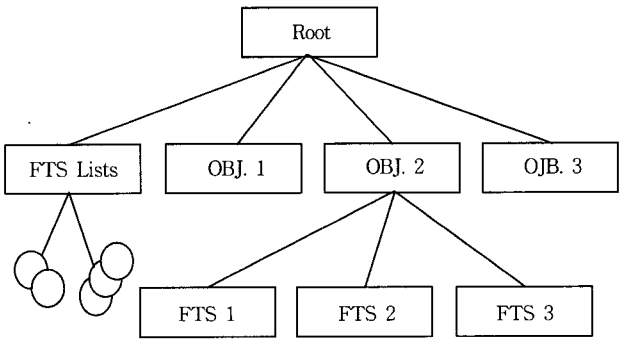
(그림 2) 결함 허용 서비스의 구조

그러므로 클라이언트는 서비스 객체를 가지고 있는 서버를 네이밍 서버를 통해 알 수 있고, 동시에 가장 빠른 결함 허용 서버를 알 수 있게 된다. 이때 클라이언트는 가장 빠른 결함 허용 서버를 선택하고, 결함 허용 서버로부터 서비스 객체를 얻게 된다. 이러한 동작을 통해서 클라이언트는 객체에 대한 서비스를 보장 받을 수 있다. 여기서 네트워크 C의 서버객체가

실제의 객체를 가지고 있고, 이 객체에 대한 복제된 객체를 네트워크 A와 네트워크 D의 결함 허용 서버가 가지고 있음을 보여 준다. 그러므로, 네트워크 B에 있는 클라이언트는 네트워크 C의 서버로 직접 연결할 필요 없이 네트워크 A의 결함 허용 서버로 연결을 해서 원하는 객체를 얻을 수 있다.

3.1 네이밍 서비스를 이용한 결함 허용 서비스 정보

(그림 3)은 네이밍 서비스를 이용한 결함허용 서비스의 구조를 보여주고 있다. 이 그림은 그림 왼쪽의 결함 허용 서버 리스트와 그림 오른쪽의 객체 트리의 두 부분으로 나누어 진다. 각 객체는 클라이언트에 빠른 연결을 가지는 결함 허용 서버에 대한 정보를 가지고 있는 여러 결함 허용 서버의 리스트를 가지고 있다.



(그림 3) 네이밍 문맥의 구조

클라이언트가 원하는 객체를 찾을 때, 클라이언트는 빠른 결함 허용 서버들에 대한 정보를 네이밍 서비스로부터 알 수 있다. 이때 클라이언트는 빠른 결함 허용 서버들 중에서 가장 빠른 결함 허용 서버를 선택하게 되는 것이다.

3.2 확장 ORB를 통한 결함 허용 서비스의 모듈 설계

본 논문에서 제안한 결함 허용 서비스는 실시간 CORBA에 있는 ORB의 확장을 통하여 진보된 실시간 CORBA의 ORB 구조를 가진다. 이 확장된 ORB에는 3가지 모듈 즉, Object Holding and Passing, Finding the Fast ORB, Making a New Connection모듈들이 추가 되었다. 결함 허용 서버는 서버로부터 해당 객체를 가져오게 된다. 또한 클라이언트는 그 중 가장 빠른 결함 허용 서버를 선택 할 수 있다. 이를 위해 ORB는 확장 ORB간에 랜덤 데이터의 전송률을 체크하는 기능이 있다. 이 기능으로 ORB는 빠른 결함허용 서버쪽으로 새로운 연결을 한다.

4. 지능적 우회기능 지원 결함 허용 서비스의 구현

본 논문에서는 결함 허용 서비스의 구현은 NT환경을 사용하였고, CORBA ORB로 워싱턴 대학에서 개발한 실시간 CORBA인 TAO ORB를 가지고 구현하였다. 본 논문의 결함허용 서비스의 구현 모듈은 다음의 3가지 모듈로 구성된다.

첫째, Object Holding and Passing 모듈이고,
둘째, Building a New Connection모듈,
셋째, Finding the Fast ORB모듈이다.

4.1 결함허용 서비스 인터페이스 정의

이러한 결함허용 서비스를 생성하기 위해 CORBA IDL로 (그림 4)에서처럼 FTS_Service() 인터페이스를 정의했다. 이 IDL은 간단한 동작을 하는 오직 하나의 인터페이스를 가지고 있다.

```
Interface FTS_Service {
        String GetVersion()
}
```

(그림 4) FTS_Service IDL

위 그림에서 FTS_Serivce 인터페이스에는 버전정보를 제공하는 GetVersion() 메소드를 가진다. 이 IDL 스크립트를 작성하여 나온 소스코드를 가지고 인스턴스를 생성하면 결함허용 CORBA 서버가 되는것이다.

4.2 Object Holding and Passing 모듈

결함 허용 서버를 구현하기 위해, 확장 시킨 실시간 CORBA ORB인 TAO를 이용한다. 모든 CORBA의 객체들은 CORBA_ Object를 모체로 하고 있기 때문에 객체에 대한 정보를 가지고 있기에 가장 좋은 클래스이다. 그러므로 결함 허용 서비스를 위해서 (그림 5)에 나타난 것처럼 ORB 객체를 확장했다.

그 중 public멤버 함수인 VerifyServer()메소드의 기능은 이 ORB가 결함 허용 서버로서 동작할 때, 클라이언트로부터 요구 되어지는 객체에 대한 정보와 서버에 대한 정보를 가지고 객체에 접근, 객체를 자신의 ORB에 가지고 있게 하기 위한 기능을 수행한다. (그림 6)는 이 VerifyServer() 메소드의 모든 동작 과정을 보여준다.

```
class TAO_Export CORBA_Object:public TAO_Iunknown
{
 ...
 CORBA::Object_ptr VerifyServer(CORBA::String ObjName,
      CORBA::String SvrName, int argc, char** argv);
 CORBA::String GetArbitraryData();
 CORBA::String FindFastORB(char* lists[50], int numof);
 Static CORBA::Object_ptr connect(CORBA::String ObjName,
      CORBA::String SvcName, int argc, char** argv);
 CORBA::Float GetElaspedTime();
 Int isConnected;
 Float elapsedTime;
 CORBA::ORB_var resolved_orb;
 CORBA::Object_ptr resolved_obj;
}
```

(그림 5) Object.h내의 CORBA_Object class의 확장 부분

```
ORB를 초기화 한다.

TAO_TRY {                    // 연결 시도에 대한 에러 catch

  ORB_init()                 // ORB초기화
  Resolve_initial_reference() // 네임 서버 초기화
  Bind()                     // 네임 문맥 정보를 획득
  Resolve()                  // 해당 객체를 획득
}
획득한 객체 레퍼런스를 리턴.
```

(그림 6) VerifyServer메소드의 구현

위 그림에서 보여주는것처럼 먼저 ORB_init()을 통해 ORB 초기화를 한다. 그리고 네임서버를 초기화 하기위해 Resolve_ initial_reference() 를 호출한다. 또한 네임문맥 정보를 얻기위해 Bind() 메소드를 그리고 해당 객체를 얻기위해 Resolve() 메소드를 호출한다.

4.3 Building a New Connection 모듈

클라이언트가 네이밍 서버로부터 결함 허용 서버 리스트를 얻은 후에, 클라이언트는 객체의 참조를 얻기 위해 이 리스트 중에서 가장 빠른 결함 허용 서버를 알아야 한다. 이때의 동작과정은 다음과 같다.

첫째, connect()메소드를 이용해 원하는 객체를 가지고 있는 네이밍 서버로 연결한다.

둘째, 가장 빠른 결함 허용 서버를 찾기 위해 FindFastORB() 를 호출한다.

셋째, VerifyServer()를 통해 클라이언트는 선택된 결함 허용 서버를 통해 원하는 객체를 검색한다.

넷째, 결함허용서버는 원하는 객체를 클라이언트로 되돌려 준다. 이러한 멤버 메소드는 정적(static) 메소드로 정의 되어 있기 때문에 CORBA환경에서 언제나 쓰일 수 있다.

4.4 Finding the Fast ORB 모듈

가장 빠른 결함허용서버 ORB를 찾기 위해, 각 클라이언트의 ORB는 각 결함허용 서버들로부터의 전송률을 체크 해야 한다. 이때, 클라이언트 ORB는 원하는 결함허용 서버로부터 몇 번의 임의의 데이터를 받는다. 그리고 이를 가지고 전송 시간의 평균을 계산한다.

(그림 7)은 구현한 FindFastORB() 메소드를 보여준다. 그리고 임시 데이터를 제공하고 그것의 참조를 되돌리기 위해 GetArbitary Data()라는 메소드를 호출한다.

위 (그림 7)에서는 등록되어있는 서버 리스트중에서 가장 빠른 결함허용 서버를 찾는 모듈이다. 서버 리스트에 있는 서버중에 한 서버에 접속하여 서버와의 데이터 전송률을 확인한다. 그래서 현재의 서버가 가장 빠르다면 그 서버 이름과 전송률을 저장한다.

```
For( # of server list ) {
                // 서버 리스트의 수 만큼 for문으로 돌린다.
  Orb_init()    // 초기화
  Connect to target server
                // 해당 서버이름으로 접속
  Check transfer rate of target server
                // 서버의 일정량의 데이터 전송률을 본다.
  Compare transfer rate with old one
                // 전송률을 이미 저장되어진 speed값과 비교.
  Store faster server name
                // 가장 빠른 서버 이름 저장
  Store transfer rate value
                // 데이터 전송속도 저장
}
```

(그림 7) FindFastORB()메소드

(그림 8)는 몇 번의 임의의 데이터를 받아 전송 시간의 평균을 계산하기위해 data라는 배열에 1000개의 문자를 저장하고 그 배열의 포인터를 되돌려주는 GetArbitaryData() 메소드를 사용하였다.

```
Char data[1000]

Data 배열에 1부터 1000을 삽입
Data의 포인터 리턴
```

(그림 8) GetArbitaryData() 메소드

5. 개발 결과

본 논문에서는 결함 허용 서버와 서버객체, 클라이언트를 각각 다른 네트워크에 배치시켰다. 즉, 서로 다른 3개의 네트워크에 각각 결함 허용 서버,서버객체, 클라이언트 프로그램을 위치 시킨다. 네이밍 서버 데몬은 어떤 네트워크에 위치해도 상관없다. 그리고 나서 클라이언트를 제외한 각각의 프로세스들을 실행 시킨다.

5.1 결함 허용 서버

결함 허용 서버는 서버나 또는 클라이언트로서의 역할을 하는 것이 아니라 ORB에 내장되기 때문에 특정 서비스 객체를 네이밍 서버에 등록하거나 객체 서비스를 받기 위한 다른 서버로의 접속을 필요로 하지 않는다.

```
C:\workshop\TAO\FTS>FTS_Service myFTS
Warning. That name already exists.
Trying to register this daemon as "myFTS1"
Well Registered FTS Daemon as "myFTS1"
```

(그림 9) 결함 허용 서버의 실행

먼저, 결함 허용 서버를 실행하기 위해 (그림 9)에서 보인 것과 같이 콘솔상에서 "FTS_Service myFTS"라고 결함 허용 서버의 고유 이름을 지정한다. 이것은 네이밍 서버로 등

록 시 가져야 할 고유 이름을 의미한다. 만약 이미 네이밍 서버에 등록된 이름이라면 임의로 추가된 이름으로 등록시키고 결과 값을 출력한다.

이때 결함 허용 서버가 등록하려고 할 때, 이미 네이밍 서버에 등록된 같은 이름이 있다면 결함 허용 서버는 'myFTS1'과 같이 다른 이름을 선택하여 결함허용 서버이름을 등록한다.

결함허용서버 구동 시 이름을 가지고 Name Server에 등록시키는 과정은 다음과 같다. 결함허용 서비스 데몬은 FTS_Service 라는 간단한 IDL Interface 정의를 가지는 객체를 가지고 구동된다. 이미 ORB 부분에서 객체를 우회시킬 수 있는 기능을 포함하고 있기 때문에 본 FTS Service 데몬이 구동 되면서 초기화 되어져야 할 내용은 자신을 네임서버에 등록하는 일이 전부이다. 그리고 구동 시에 입력 받은 스트링을 네임서버에 등록될 이름으로 선택하게 되는데 아래의 init_ naming_ service 함수에 그 등록 내용이 있다.

```
Int FT_Server::init_naming_service (CORBA::Environment& env)
{
  int result; isBounded, selectedFtsName;
  result = this->my_name_server_.init (this->orb_,
                    this->child_poa_); // 네임서버 초기화
  if (result < 0)
     return result;

  TAO_CHECK_ENV_RETURN (env,-1);
  CORBA::Environment _env;
  Fts_service= this->Fts_service_impl_._this (env); // FTS 서비스 객체 (IDL에 정의)

  CosNaming::Name context_name (1);
  context_name.length (1);
  context_name[0].id = CORBA::string_dup ("FTS_List");

 // 상위 FTS_List Context를 bind 해 온다

this->context_ = my_name_server_->bind_new_context(context_name, env);
  TAO_CHECK_ENV_RETURN (env,-1);

 //bind된 FTS_List에 새로운 FTS를 등록한다.
   CosNaming::Name fts_name (1);
  fts_name.length (1);
  while(isBound)
  {
       if(selectedFtsName)
            selectedFtsName에 새로운 FTS 이름 선택
       else
         selectedFtsName에 시작시 등록한 FTS 이름 선택
       factory_name[0].id = CORBA::string_dup (selected
                          FtsName);
       isBound = this->context_->bind (fts _name,
                  Fts_service.in (), env);
 }
 bind된 FTS 이름 출력;
 TAO_CHECK_ENV_RETURN (env,-1);
 return 0;
}
```

(그림 10) 네임서버에 등록

FTS Service 데몬을 네임서버에 등록할 때에는 클라이언트에서 공통적으로 접근이 가능한 'FTS_List' 라는 컨텍스트의 Sub 컨텍스트로 등록을 합니다. 이로서 네임서버로의 FTS 서비스 데몬의 등록은 마치게 된다. 그런 다음 클라이언트의 접속에 의해서 FTS_List 중 한 개의 결함허용 서버가 선택되어지게 된다.

5.2 네이밍 서버

네이밍 서버는 다른 서버들로부터 객체 등록을 받기 위해서 실행된다. (그림 11)은 대기 상태에 들어가 있는 네이밍 서버의 실행 모습을 보여 주고 있다.

```
[kkucc: Naming_Service 55]Naming_Service
TheIORis:
<IOR:000000000000001549444c3a4d656469615657276696365 3a312e3000
0000000000000100000000000003c000100000000000056f78656e0000271f000
0002850333632646462626330303037323531375 26f6f74504f412f6368696c64
5f706f612f6d65646961>
The multicast server setup is done.
```

(그림 11) 네이밍 서버의 실행

이때 CORBA객체를 유일하게 식별하게 하는 융통성 있는 주소 메커니즘인 IOR이 위에서처럼 화면에 출력된다. 그리고 네이밍 서버가 준비가 되었다는 것이 화면에 출력된다.

```
Bound:<MediaService_impl,nil>
(15906|1) Received multicast
```

(그림 12) 네이밍 서버가 객체 서버로부터 연결을 받았을 경우

네이밍 서버는 클라이언트나 결함 허용 서버 혹은 서버객체로부터 연결 요구를 받고 연결에 대한 정보를 표시하게 된다. (그림 12)는 객체 서버로부터 객체 등록을 요구 받은 후의 네이밍 서버의 상태를 보여주는데 위 메시지는 MediaService_impl객체가 바운드 되었다는 메시지이다.

5.3 객체 서버

객체 서버는 자신이 서비스 해야 할 객체를 초기화 하고 이를 네이밍 서버에 연결하여 등록해야 한다. (그림 13)은 TAO로 짜여진 서버 프로그램을 실행 시킬 때 자신이 서비스 해야 할 객체 정보인 IOR을 출력하면서 시작하게 되는 과정이다.

(그림 13)에서처럼 콘솔에서 "multisvr"이라고 객체서버를 실행시키면 객체서버가 시작한다는 메시지를 출력하고 다이나믹 서비스인 Resource_Factory, Client_Strategy _Factory, Server_ Strategy_ Factory 등이 실행된다. 마지막으로 객체 서비스를 위해 객체를 인식할 수 있는 유일한 주소 체계인 IOR을 화면에 출력하고 객체서비스를 위해 클라이언트의 요구를 기다린다.

```
D:\workshop\TAO\multisvr>multisvr
Starting up daemon <unknown>
Opening dynamic service Resource_Factory
Did dynamic on Resource_Factory, error = 0
Opening dynamic service Client_Strategy_Factory
Did dynamic on Client_Strategy_Factory, error = 0
Opening dynamic service Server_Strategy_Factory
Did dynamic on Server_Strategy_Factory, error = 0
Listening as object
<IOR:000000000000002049444c3a436f734e616d696e672f4e616d696e6743
6f6e746578743a312e300000000001000000000000004200010000000000056
f78656e0000271f0000002e503336326464626263303030373235313752 6f6f7
4504f412f6368696c645f706f612f4e616d65536572766963 65>
```

(그림 13) 객체 서버를 실행시킨 모습

5.4 클라이언트

클라이언트를 실행시킬 때는, 확장된 CORBA프로그램이 플러그인 된 넷스케이프를 실행시키면 된다. URL란에는 http://cclab/paper.sgm와 같이 간단하게 자신이 원하는 서버 이름과 자신이 원하는 객체 이름을 입력하면 된다.

객체 이름을 입력하면 넷스케이프는 플러그인 라이브러리를 실행시킴으로 인해 CORBA 클라이언트를 실행시키게 된다. 그때 CORBA 클라이언트는 서버 객체 또는 결함 허용 서버로부터 객체를 얻을 수 있게 된다.

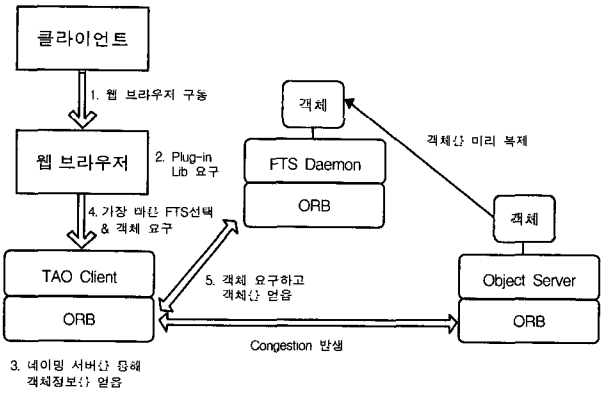
```
Bound:<FTS_Service_impl,nil>
(15906|1) Received multicast.
```

(그림 14) 결함 허용 서버가 클라이언트로부터 요구를 받았을 경우

(그림 14)은 객체 서버 또는 결함 허용 서버가 클라이언트로부터 요구를 받게 되면 제대로 메시지를 받았다고 화면에 출력한다. 즉 FTS_Service_impl 객체가 바운드되어 서비스를 할 수 있는 경우 나타나는 메시지 이다.

5.5 결함 허용 서비스의 동작

결함 허용 서비스의 모든 동작은 (그림 15)와 같이 세 개의 모듈로 나누어서 설명할 수 있다.



(그림 15) 결함 허용 서비스를 이용한 내부 동작

(그림 15)는 결함허용 CORBA서비스가 동작하는 전 과정을 보여주며 그 순서는 다음과 같다.

첫째, 클라이언트가 웹 브라우저의 URL란에 객체 이름과 서버이름을 입력한다.

둘째, 이때 웹 브라우저는 넷스케이프에 CORBA 플러그인 라이브러리를 통해 CORBA 클라이언트를 동작시킨다.

셋째, 클라이언트 ORB는 네이밍 서버로부터 객체에 대한 정보를 요구한다.

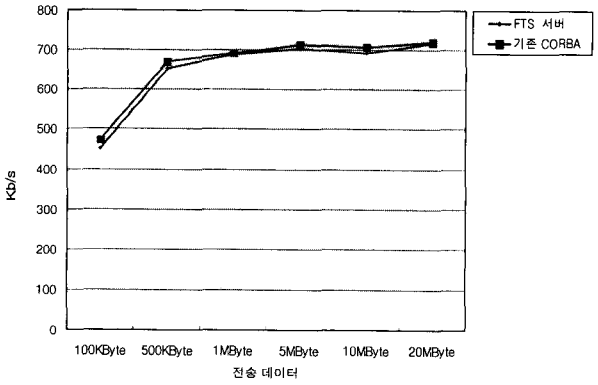
넷째, 클라이언트는 네이밍 서버로부터 받은 객체에 대한 정보를 통해 가장 빠른 결함 허용 서버를 선택하고 객체를 요구한다.

다섯째, 이때 결함 허용 서버는 클라이언트가 원하는 객체를 가지고 있는 객체 서버로 연결을 한다.

여섯째, 그러면 결함허용 서버는 클라이언트가 요구한 객체를 주게된다.

6. 실 험

다음은 본 논문에서 구현한 신뢰성 지원을 위한 결함허용 서비스의 두 가지의 경우에 대해 테스트했다. 첫번째는 (그림 16)에서처럼 정상 네트워크일 때의 결함허용 CORBA 서버와 기존 CORBA와의 네트워크 속도의 성능 비교이다.

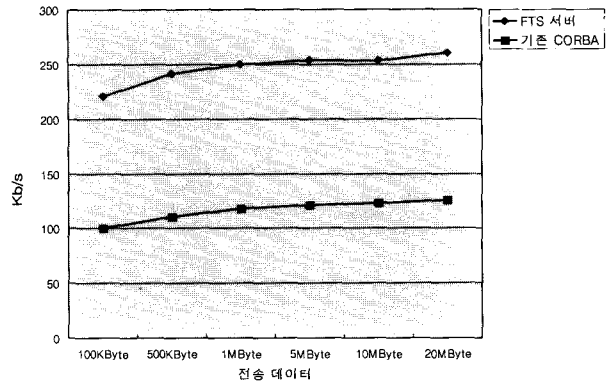


(그림 16) 정상 네트워크 때의 결함허용CORBA 서버와 기존 CORBA의 성능비교

여기서 볼 때 정상 네트워크일 때의 결함허용 서비스(Fault-Tolerant CORBA Service)서버와 기존 CORBA와의 성능 비교에는 큰 차이가 없다. 정상 네트워크일 때는 결함허용 서비스와 같은 우회서비스를 클라이언트가 받을 필요가 없기 때문이다.

그러나, 두 번째의 경우는 (그림 17)와 같이 정체(Congestion)가 발생했을 때, 결함허용 CORBA 서버와 기존 CORBA와의 네트워크 속도의 성능 비교이다. 이때 아래 그래프에서 보듯이 정체 시에 그대로 기다리는 것이 아니라 지능적으로 우회해주는 서비스 때문에 기존 CORBA보다 결함허용 서비스

를 지원하는 CORBA서버가 더욱 빠른 전송률을 보이고 있는 것을 알 수가 있다.



(그림 17) 결함 발생시의 결함허용CORBA 서버와 기존 CORBA의 성능비교

그러므로 위 실험 결과에서 보듯이, 본 논문에서 제안한 결함허용 CORBA 서비스가 정체가 발생했을 때 지능적으로 우회서비스를 제공함으로 네트워크의 속도의 저하를 막아준다는 것을 알 수 있다. 그럼에도 결함 발생시에는 정상 네트워크에 비해 속도가 느린 것은 가장 빠른 결함허용 CORBA 서버를 찾아주는 오버헤드와 객체를 우회해주는 오버헤드가 있기 때문이다.

7. 결 론

현재의 실시간 CORBA는 하부 망 구조를 ATM등과 같은 고속 망을 이용한다고 명시만 되어 있고 더 이상 언급을 하지 않음으로써 망 결함 발생에 대한 연결 보장을 못한다는 단점이 있다. 따라서 본 논문에서 제시한 결함 허용 서비스의 계발로 인해서, 기존의 실시간 CORBA 의 단점인 신뢰성 있는 전송을 위한 연결 보장성을 극복하게 되었다.

본 논문에서는 신뢰성 보장을 지원하기 위한 결함 허용 서비스의 설계와 구현에 관해 언급하였다. 이러한 결함 허용 서비스를 통하여 CORBA 응용프로그램은 네트워크에서 서버로의 연결에 대해 안정성과 신뢰성을 보장한다. 그러므로 본 논문의 결함허용 CORBA는 확장된 ORB를 이용해 신뢰성 있는 연결과 이식성을 보장할 수 있게 된다.

참 고 문 헌

[1] Jon Siegel, "CORBA Fundamental and Programming," WILEY, pp.84-204, 1996.

[2] Robert Orfali & Dan Harkey, "Client/Server Programming with Java and CORBA," 2nd Edition, Wiley Computer Publishing, p.331, p.651, 1998.

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

[4] Paul Ferguson, et al, "Quality of Service," Wiley Computer Publishing, 1998.

[5] Sunyoung Han, et al, "Scaleable and Reliable Synchronous Collaboration Environment on CORBA using WWW," Proceedings of the Second IEEE High-Assurance Systems Engineering Workshop(HASE'97), 11-12 August 1997.

[6] 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.

[7] Steve Vinoski, "CORBA : Intergrating diverse applications within distributed heterogeneous environments," IEEE Communications Magazine, February 1997, pp.46-55.

[8] Zhigang Chen, See-Mong Tan, Roy H.Cahpbell, Youngcheng Li, "Real-Time Video and Audio in the World Wide Web," Proceedings of the 4th International World Wide Web conference, Dec. 1995, available at (http : //www.w3.org/pub/WWW/Journal/1/stan.211/papaer/211.html).

[9] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I .Zykh, and R. Johnston, "Real-Time CORBA," in Proceedings of the Third IEEE Real-Time Technology and Applications Symposium, (Montreal, Canada), June 1997.

[10] V. Fay-Wolfe, J. K. Black, B. Thuraisingham, and P. Krupp, "Real-Time Method Invocations in Distributed Environments," Tech.Rep.95-244, University of Rhode Island, Department of Computer Science and Statics, 1995.

[11] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," Theory and Practice of Object Systems, Vol.3, No.1, 1997.

[12] I. Wakeman, A.Ghosh, J. Crowcroft, V. Jacobson, and S. Floyd, "Implementing Real-Time packet forwarding policies using streams," Technical report, January 1995, available at ftp : //cs.ucl.ac.uk/darpa /usenix-cbq.ps.Z.

[13] E. Bakken, E. Schantz and A. Zinky, "QoS Issue for Wide-Area CORBA-Based Object Systems," Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '96), (California, USA), Feb. 1996, pp.110-112.

[14] Z. Chen, S.Tan, R. Cahpbell, and Y. Li, "Real-Time Video and Audio in the World Wide Web," Proceedings of 4th International World Wide Web Conference, Dec. 1995, pp.333-348.

[15] OMG, "RealTime CORBA 1.0," Real-Time CORBA 1.0 RFP, Fragingham Corporate Center, Jan. 1998, orbos /97-09-31, http : //www.omg.org/library/schedule/Realtime_CORBA_1.0_RFP.htm.

[16] Object Management Group, Fault Tolerant CORBA Specification, OMG Document orbos/99-12-08 edition, December 1999.

[17] Andy Gokhale, Bala Natarajan, Douglas C. Schmidt and Shalini Yajnik, Applying Patterns to Improve the Performance of Fault-Tolerant CORBA, Proceedings of the 7th International Conference on High Performance Computing (HiPC 2000), ACM/IEEE, Bangalore, India, December 2000, available at http : //www. cs.wustl.edu/~schmidt/new.html.

[18] Balachandran Natarajan, Aniruddha Gokhale Yajnik, Douglas C. Schmidt, DOORS : Towards High-performance Fault Tolerant CORBA, Proceedings of the 2th Distributed Application and Objects(DOA) conference, Antwerp, Belgium, Sept 21-23, 2000.

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
관심분야 : CORBA, Internet Caching, Mobile IP, 차세대 인터넷 프로토콜, 보안

홍 성 준

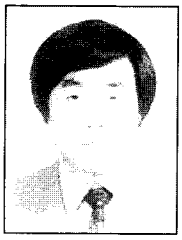
e-mail : sjhong@mail.yeojoo.ac.kr

1991년 경원대학교 전자계산학과 졸업 (공학사)
1993년 건국대학교 전자계산학과 졸업 (공학석사)
1998년 건국대학교 컴퓨터 정보통신공학과 졸업(공학박사)
1993년~1999년 한국통신 연구소 전임연구원
1999년~현재 여주대학 정보통신과 전임강사
관심분야 : 컴퓨터 네트워크, 멀티미디어 시스템, 차세대 인터넷 프로토콜

박 기 용

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

1997년 건국대학교 수학과 졸업(공학사)
1999년 건국대학교 컴퓨터.정보통신공학과 졸업(공학석사)
관심분야 : 분산객체, Internet Caching, Mobile IP



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 컴퓨터공학과 교수
1989년~1990년 미국 Maryland대 컴퓨터과학과 객원부교수
1990년~현재 개방형 컴퓨터 통신 연구회 이사
1990년~1997년 한국과학기술원 인공지능 연구센터 참여교수
1991년~1993년 한국 정보과학회 정보통신연구회 부위원장
1996년~1998년 개방형 컴퓨터 통신 연구회 총무이사
1998년~1999년 미국 Maryland 대학교 컴퓨터 과학과 객원교수
2000년~현재 건국대학교 정보통신원 원장
관심분야 : 실시간 CORBA, Internet Caching, 차세대 인터넷 프로토콜, Mobile IP, VoIP, Ipv6