신뢰성 있는 Java RMI 객체 설계 및 구현

윤 태 진, 박 양 우, 이 채 수
경운대학교 소프트웨어공학과
e-mail:tjyun@kyungwoon.ac.kr

Fault-tolerant Java RMI Object Design and Implementation

Tae-Jin Yun, Yang-Woo Park, Chae-Su Lee
Dept. of Software Engineering, Kyungwoon University

요 약

CORBA, DCOM, Java RMI 등과 같은 분산 객체 기술이 분산 응용의 신뢰성을 직접적으로 향상시키지는 못한다. 이러한 분산 객체 기술에 고장 감내성을 추가하여 신뢰성 있는 시스템을 구축하기 위해서 객체 단위의 복제 그룹 관리와 고장 탐지 및 회복 메커니즘이 필요하다. 본 논문에서는 신뢰성 있는 고장 감내성 Java RMI 객체를 개발하기 위하여 고장 탐지와 그룹 관리를 위한 그룹 관리자와 원격 인터페이스를 설계하고, 고장 감내성 클래스를 정의한다. 또한 고장 감내성 객체의 투명한 그룹 참여를 위하여 Naming 클래스와 RMIRegistry를 확장한다. 응용 개발자는 고장 감내성 클래스를 상속함으로써 외부의 도움 없이 간단히 고장 감내성 응용 객체를 개발하여 신뢰성을 높일 수 있다. 100M Ethernet으로 연결된 Linux kernel2.2.1의 PentiumIII머신 3대와 Solaris2,6의 Sun Sparc1머신을 클라이언트한 시스템을 사용하고 JDK 1.2.2를 이용하여 설계한 객체를 구현하여 다양한 객체고장에 대해 복구되는 것을 확인하였다. 성능 평가는 그룹크기에 따른 함수의 응답속도와 메시지 크기에 따른 응답속도를 비교하였다.

1. 서 론

최근 컴퓨터와 통신 기술의 발전에 따라 분산 응용이 보편화되고 이를 개발하기 위한 많은 분산 객체 기술이 연구되었다. 대표적인 표준 기술로는 CORBA[1], DCOM[2], Java RMI[3] 등이 있지만 이러한 기술들이 응용의 신뢰성을 직접적으로 향상시키지는 못한다. 결과적으로 응용개발자들은 응용의 신뢰성과 가용성을 향상시키기 위하여 자체 메카니즘을 구현하여야만 한다. 최근 고장 감내 응용의 개발을 편리하게 하기 위한 재사용 가능한 도구들이 연구되어 왔다. 그러나, 이들 중 많은 시스템이 통신기법에 대해 프로그래머가 많은 것을 다루어야 하고, 시스템이 플랫폼에 종속적이며 또한 재사용성이 좋지 않아 매번 비슷한 고장 감내 코드를 만들어야 하는 등의 단점을 가진다.

본 논문에서는 Java RMI시스템에서 신뢰성과 높은 가용성을 보장하기 위해 고장 감내 RMI객체와 원격 인터페이스를 설계하였다. 복제의 기본단위로 객체를 지원하고, 복제본은 같은 원격 인터페이스를 구현하며 같은 소스코드를 가진다. 객체 복제본들의 집합은 객체 그룹을 형성하며 그룹통신을 통하여 객체간의 일관된 상태유지와 고장탐지를 수행한다. 객체 복제본 간의 그룹 통신을 지원하기 위하여 그룹관리자 객체를 생성하여 고장 감내 객체에 포함함으로써 외부 서비스의 도움 없이 객체 그룹을 형성할 수 있다. 본 시스템에서 응용 개발자는 상태 전달 인터페이스만을 구현함으로써 간단히 고장 감내 응용을 개발 할 수 있다.

2. 분산 객체 Java RMI와 고장 감내성

자바 기반의 분산 객체 시스템인 Java RMI는 같은 호스트나 다른 호스트의 JVM(Java Virtual Machine)상에서 동작하는 원격 객체에 의해 구현된 원격 인터페이스의 함수를 로컬 객체의 함수 호출과 같은 방식으로 호출할 수 있게 한다. 이와 유사한 기술로는 RPC(Remote Procedure Call)가 있다.

분산 환경하에서 고장 감내성을 향상시키기 위한 다양한 연구가 진행되어 왔다. 특히 분산 시스템의 표준이라 할 수 있는 CORBA(Common Object Request Broker Architecture)를 기반으로 한 최근 연구는 중복객체의 그룹관리 방식 및 메시지 전달방법 등에 따라 통합 방식, 인터셉트 방식, 서비스 방식으로 분류할 수 있다. 통합 방식은 CORBA의 ORB(Object Request Broker)아래에 신뢰성 있는 멀티캐스트 통신 서브시스템을 추가하여 새로운 ORB를 만드는 방식이다[4].

이 방식에서 하위 레벨의 그룹 통신 계층을 이용하여 클라이언트의 요청을 복제본에 신뢰성 있게 멀티캐스트 함으로써 고장 감내 객체시스템을 구현하였다. 인터셉트 방식은 TCP/IP와 같은 하위 레벨의 통신 시스템에서 클라이언트 객체에 의해 만들어지는 시스템 콜을 가로채서 신뢰성 있는 멀티캐스트 서브시스템으로 전달하는 것이다[5].

장점으로는 ORB의 수정 없이 응용에 투명하게 고장 감내성을 얻을 수 있다는 것이다. 서비스 방식은 그룹 통신을 ORB자체에 통합하지 않고 ORB상위에 CORBA 서비스로서 제공하는 것이다. 서버 측에서는 이 서비스를 이용하여 명시적으로 그룹을 만들고, 참여하여야 하며 클라이언트 측면에서는 서비스에 요청을 보내야 한다[6]. 서비스는 그룹의 관리와 클라이언트의 요청이 모든 그룹 복제본에 신뢰성 있게 전달되는 것을 보장하고 응답을 되돌려 준다.

복제 시스템은 고장 감내 시스템에서 매우 중요한 요소이다. 복제 방법은 복제본의 상태를 어떻게 동기화 하는 가와 클라이언트 객체와 상호작용 방법에 따라 크게 "Cold Standby","Worm Standby","Hot Standby"로 나눌 수 있다[7]. Cold Standby방식은 실행 시 프라이머리 객체와 백업 객체 사이에 어떠한 통신도 발생하지 않는다.

이 방식은 통신비용이 매우 적은 반면에 백업 객체가 호출되었을 때 현재 프라이머리 객체의 상태를 가져와야 하므로 회복시간이 많이 소요된다. Warm Standby방식은 프라이머리 객체각 호

줄된 요청과 내부상태를 주기적으로 백업 객체에 전달한다. 실행시간 동안, 프라이머리 객체는 자신의 상태가 변할 때마다 백업 객체의 상태를 갱신한다. 이 방식의 단점은 객체 관리자에서 단일점 실패가 나타날 수 있고 고장 복구와 새로운 객체 생성 프로토콜이 복잡하다는 것이다. 통신 비용은 Cold Standby에 비해 많으나 고장 복구 시간은 적다. Hot Standby방식은 다수개의 복제본이 모든 클라이언트의 요청에 대해 동시에 응답한다. 클라이언트는 모든 복제된 서버객체에 요청을 전달하고, 모든 서버 객체는 이 요청을 실행하고, 결과를 클라이언트에 되돌려 준다. 클라이언트는 첫 번째 리턴 값을 선택하거나, 모든 리턴 값을 받아서 그 중 하나를 선택할 수 있다. 만약 복제본 중 하나에 고장이 발생하더라도 다른 객체가 요청을 처리할 수 있으므로 복구과정이 필요 없다. 이전의 방식에 비해 통신 비용이 가장 많이 소요되지만 복구시간이 가장 짧다.

3. 고장 감내 RMI객체의 설계

본 논문에서는 고장 감내 RMI 객체를 만들기 위하여 복제 방법으로 Warm Standby방식을 채택하였다. 이 방식에서는 모든 클라이언트의 요청은 프라이머리 객체로 전달되고, 프라이머리 객체는 이 요청을 처리한 후 변경된 상태를 백업 객체에 멀티캐스트 한다. 또한 그룹관리와 상태 전달, 고장 탐지를 위하여 원격 인터페이스를 정의하고 원격 객체에 이 인터페이스를 구현한 그룹관리자를 내장함으로써 외부의 도움 없이 원격 객체간에 그룹통신을 가능하게 하였다.

3.1 고장 감내 RMI 시스템의 구조

분산 객체 시스템에 고장 감내성을 부여하기 위하여 그룹관리, 고장 탐지, 고장 복구, 객체의 동일한 상태 유지 등의 기능이 필요하다. 본 시스템에서 프라이머리 객체는 다수개의 백업 객체와 함께 그룹을 형성하며 그룹의 중재자 역할을 담당한다. 그룹의 모든 복제본은 그룹 관리와 고장 탐지를 담당하는 그룹 관리자를 포함하며, 이들은 가상 링을 형성하여 객체 고장을 탐지한다.

프라이머리 객체의 고장 시, 백업 객체 중 하나가 프라이머리 객체의 역할을 담당한다. 클라이언트는 원격 객체 호출을 위하여 RMIRegistry로부터 프라이머리 객체의 참조를 받아오기 때문에 모든 호출은 프라이머리 객체로 전달된다. 프라이머리 객체는 중요한 내부 상태가 변경되었을 때 그룹 관리자를 통하여 백업 객체로 상태를 멀티캐스트 함으로써 복제본의 상태를 동일하게 유지한다.

백업 객체는 자신의 그룹 관리자 객체를 통하여 프라이머리 객체에 등록함으로써 그룹에 참여한다. 프라이머리 객체는 백업 객체에 그룹 뷰와 객체상태를 콜백 호출을 통하여 전달한다.

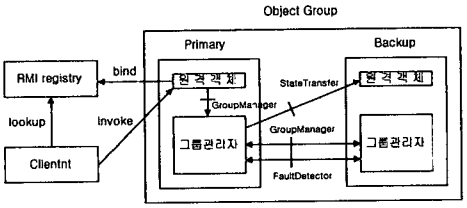

그림 1 고장 감내 RMI 시스템의 구조

Fig. 1 Fault tolerant RMI system architecture

고장 감내 RMI 시스템과 전체 구조는 그림 1과 같다. 각 복제본은 개발자가 구현한 원격 객체와 그룹 관리자로 구성되어 있으며, 수직선을 가진 화살표는 목표 객체가 화살표에 표시된 원격 인터페이스를 지원한다는 것을 보여주기 위해 사용되었다. 원격 인터페이스 타입의 객체 참조를 가진 클라이언트는 인터페이스에 의해 정의된 함수를 호출할 수 있다.

본 시스템에서 객체의 투명한 그룹 생성과 참여를 위하여 RMIRegistry를 수정하고, Naming 클래스를 확장한 FTNaming 클래스를 정의한다. Sun의 rmiregistry는 원격객체의 등록을 로컬내의 객체로 제한하므로 이를 확장하여 원격상에서 등록이 가능하도록 하였다. FTNaming 클래스는 유틸리티 클래스로서 고장 감내 원격객체를 rmiregistry에 등록하고 그룹을 생성하거나, 그룹에 참여할 수 있도록 도와준다.

3.2 고장 감내 RMI객체 시스템의 구성요소

본 시스템은 객체에 고장 감내성을 제공하는 고장 감내 클래스와 그룹관리를 위한 그룹 관리자 클래스로 구성되어 있으며 이러한 객체에 접근하기 위한 원격 인터페이스를 제공한다.

(1) FTUnicastRemoteObject 클래스

FTUnicastRemoteObject 클래스는 표준 Java RMI의 UnicastRemoteObject를 상속한 추상클래스로서 객체 그룹 관리와 고장탐지를 위하여 GroupManager 인터페이스와 FaultDetector 인터페이스를 구현한 그룹관리자 객체를 포함한다. 고장 감내 RMI 객체는 FTUnicastRemoteObject를 상속받아 개발자가 정의한 원격 인터페이스를 구현하고, 객체의 상태 전달을 위해 StateTransfer 인터페이스를 구현한다. 이렇게 구현된 객체는 FTNaming 클래스를 통하여 그룹을 형성하며 레지스티리에 등록된다.

클래스 상속도인 그림 2에서 그룹관리자 객체를 멤버변수로 가지는 FTUnicastRemoteObject는 서버 응용 객체 생성시 그룹관리자 객체의 인스턴스를 생성한다. 서버 응용 객체는 UnicastRemoteObject 클래스와 StateTransfer 인터페이스를 상속하며, StateTransfer 인터페이스는 FTUnicastRemoteObject 클래스에서 구현되지 않고 개발자에 의해 구현된다.

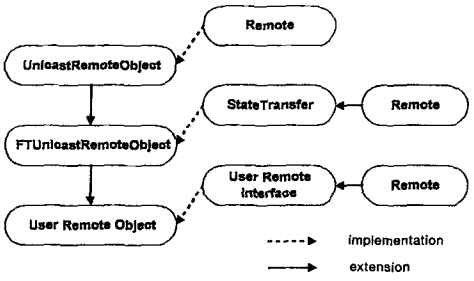

그림 2 고장 감내 클래스의 상속도

Fig.2 Fault tolerant class inheritance diagram

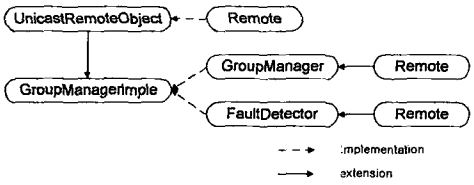

그림 3 그룹 관리자 클래스의 상속도

Fig. 3 GroupManager Class inheritance diagram

(2) 그룹 관리자

그룹 관리자는 원격 인터페이스인 GroupManager와 FaultDetector를 그림 3과 같이 구현함으로써 그룹 멤버간의 상호 호출을 통하여 그룹을 관리할 수 있다. 또한 FaultDetector 인터페이스를 통하여 객체 고장 탐지를 수행한다.

(3) 인터페이스

그룹관리와 고장 탐지, 상태 전달을 위하여 그림 4와 같이 인터페이스를 정의한다.

```
public interface GroupManager extends Remote {
    public int join_Group(ServiceRef sr)
                        throws RemoteException;
    public void leave_Group(int crash_seq)
                        throws RemoteException;
    public void reset_GroupView(ServiceRefs grns, int id)
                        throws RemoteException;
}
public interface FaultDetector extends Remote {
    public boolean is_Alive()
                        throws RemoteException;
}
```

그림 4 FaultDetector와 GroupManager 인터페이스

Fig. 4 FaultDetector and GroupManager interface

GroupManager 인터페이스는 고장 감내 객체의 그룹관리자간의 그룹 관리를 위한 인터페이스로서 백업 객체가 그룹에 참여할 때, 프라이머리 객체의 그룹관리자를 join_Group 함수로 호출한다. 입력 파라미터로서 자신의 객체참조를 전달하고, 그룹 내에서 유일하며 가상 링을 구성할 때 자신의 순서를 식별할 일련번호를 되돌려 받는다.

leave_Group 함수는 고장 감내 객체가 고장 탐지를 통하여 고장이 의심되는 객체를 탐지한 경우 그 객체의 일련번호를 파라미터로 하여 프라이머리 객체의 그룹관리자를 호출한다. 프라이머리 객체의 그룹관리자는 이 함수가 호출되면, 그룹 전체에 대해 고장을 탐지하고 새로운 그룹 뷰를 구성하여 백업객체에 멀티캐스트한다.

reset_GroupView 함수는 새로운 그룹 뷰가 구성되었을 때, 프라이머리 객체의 그룹관리자가 백업 객체의 그룹관리자를 호출하여 새로 구성된 그룹 뷰를 전달한다.

FaultDetector 인터페이스는 고장 탐지를 위하여 is_Alive 함수를 가진다. 고장 감내 RMI 객체의 복제본 그룹은 그룹 등록 순서에 따라 가상 링을 형성하며, 이 순서에 의해 자신의 상대편에게 is_Alive 함수를 호출하여 고장을 탐지한다.

```
public interface StateTransfer extends Remote {
    public Object get_State() throws RemoteException;
    public String set_State(Object obj)
                        throws RemoteException;
    public Remote get_GroupManager()
                        throws RemoteException;
}
```

그림 5 StateTransfer 인터페이스

Fig. 5 StateTransfer interface

상태전달을 위한 StateTransfer 인터페이스는 FTUnicastRemoteObject에 의해 상속되며 응용 개발자에 의해 set_State 와 get_State 함수가 그림 5와 같이 구현되어야 한다. 그룹 관리자는 이 함수를 콜백 호출하여 객체에 상태를 전달한다.

표준 RMIregistry는 원격 객체의 등록을 제한함으로 본 시스템은 이를 확장하여 원격 객체가 등록할 수 있도록 수정한다. FTNaming 클래스는 고장 감내 객체의 RMIregistry 등록과 투명한 그룹 형성을 도와 주는 유틸리티 클래스이다.

3.3 객체 등록

본 시스템에서는 RMIRegistry에 등록된 이름 문자열을 기준으로 그룹을 형성한다. 가장 먼저 생성된 고장 감내형 객체는 FTNaming클래스의 bind 함수를 통하여 RMIRegistry에 등록되고, 자신을 프라이머리 객체로 설정된다. 이후에 생성된 백업 객체는 bind 함수에 의해 같은 이름을 가진 객체가 레지스트리에 존재하므로, 백업 객체로 설정되고 프라이머리 객체의 그룹 관리자에 그룹 참여를 알린다. 이러한 과정은 그림 6과 같다.

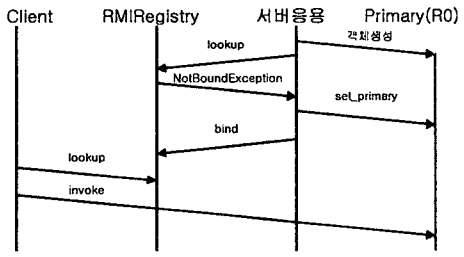

그림 6 프라이머리 객체 등록

Fig 6. Primary object registeration

서버 응용은 고장 감내 RMI 객체를 생성시키고, rmiregistry에 등록하기 위하여 주어진 문자열 이름으로 FTNaming 클래스의 정적 함수인 bind 함수를 호출한다. bind 함수는 rmiregistry의 lookup 함수를 호출하여 같은 이름으로 등록된 원격 객체가 있는지 확인한다. 만약 같은 이름으로 등록된 객체가 없다면 R0는 프라이머리 객체로 설정되고, 마지막으로 rmiregistry에 등록되어 클라이언트의 요청을 처리한다. 그러나, 다른 프라이머리 객체가 rmiregistry에 등록되어 있고 정상적으로 동작하고 있다면, 이후에 생성되는 고장 감내 객체는 백업 객체로 설정된다. 그림 7은 프라이머리 객체 R0와 백업 객체 R1이 동작하고 있을 때, 새로운 고장 감내 객체 R2가 백업 객체로 등록되는 과정이다.

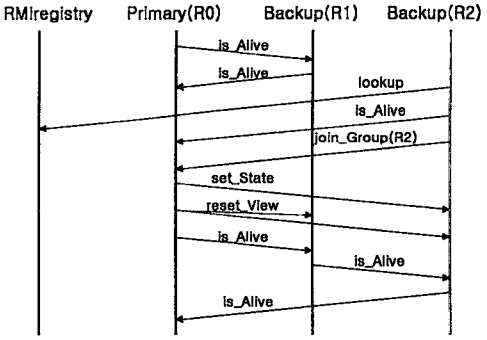

그림 7 백업 객체의 그룹 등록

Fig. 7 Backup object group registeration

새로 생성된 백업 객체 R2는 주어진 이름 문자열을 가지고 FTNaming 클래스의 bind 함수를 호출한다. 이 함수는 주어진 이름 문자열과 연관된 객체 참조가 있는 지를 확인하여, 만약 없다면 프라이머리 객체로 R2를 설정하고 RMIRegistry에 등록한다. 그러나, 같은 이름을 가진 객체가 존재하면 프라이머리 객체의 GroupManager인터페이스 객체를 받아와서 그룹에 참여한다. 프라이머리 객체는 GroupManager 인터페이스의 join_Group 함수가 호출되면, 백업 객체의

GroupManager 인터페이스의 set_State 함수를 호출하여 객체의 상태를 전달하고, 그룹의 모든 멤버에 새로 형성된 그룹 뷰를 전달하기 위해 reset_View 함수를 호출한다.

3.4 고장 탐지 및 고장 복구

객체 그룹의 멤버들은 그룹뷰의 순서에 의해 가상 링을 형성하며, 이 순서에 의해 열정시간 간격마다 자신의 상대편에게 FaultDetector 인터페이스의 is_Alive 함수를 호출함으로써 객체의 고장을 탐지한다. 만약 복제된 객체그룹의 멤버 중에서 고장이 탐지되면, 그룹의 중재자 역할을 하는 프라이머리 객체에 GroupManager 인터페이스의 leave_Group 함수 호출을 통하여 보고된다.

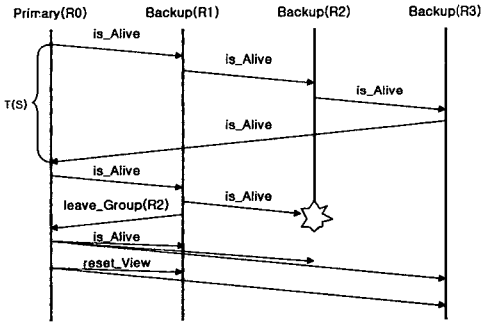

그림 8 백업 객체의 고장 복구

Fig. 8 Backup object fault recovery

프라이머리 객체는 새로운 그룹뷰를 형성하기 위하여 그룹의 각 멤버의 FaultDetector 인터페이스의 is_Alive 함수를 호출한다. 그림 8은 백업객체의 고장 탐지시 새로운 복구 과정을 보여준다.

프라이머리 객체는 객체 그룹의 중재자로써 고장 탐지시 고장 복구와 클라이언트의 모든 요청을 처리하는 중요한 역할을 담당한다. 그러므로 프라이머리 객체의 고장이 탐지되었을 때 백업 객체중 하나가 새로운 프라이머리 객체로 선출되어야 한다. 새로운 프라이머리 객체 후보는 가상 링을 구성하는 순서에 따라 결정된다. 새로 결정된 프라이머리 객체는 고장이 탐지된 이전의 객체 참조를 대신해 자신의 객체 참조를 RMIRegistry에 갱신한다.

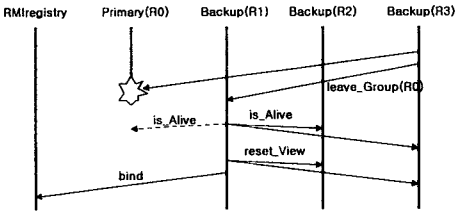

그림 9 프라이머리 객체의 고장 복구

Fig. 9 Primary object fault recovery

그림 9은 프라이머리 객체의 고장시 새로운 프라이머리 객체를 어떻게 결정하는지 보여준다. 백업 객체 R3는 프라이머리 객체 R0의 고장을 탐지하고, 백업 객체 R1의 GroupManager 인터페이스의 leave_Group (R0)를 호출한다. 백업 객체 R1은 프라이머리 객체 R0의 고장을 확인하기 위하여 is_Alive 함수를 호출한 후 응답이 없으면, 자신을 프라이머리 객체로 설정하고, RMIRegistry에 자신의 객체 참조를 등록한다. 새로운 프라이머리 객체 R1은 나머지 복제본에 is_Alive 함수를 호출하여 새로운 그룹 뷰를 형성하고, reset_view 함수를 호출하여 각 복제본에 전달한다.

4. 실험 및 성능 평가

고장 감내 RMI객체를 구현하고 성능을 평가하

기 위한 시스템의 구성은 100M Ethernet 으로 연결된 Linux kernel2.2.1의 Pentium III 500 머신 2대와 Pentium II 350머신 1대 등 3대를 사용하였고 클라이언트 머신으로는 Solaris 2.6의 Sun Sparc머신을 사용하였다. 구현은 JDK 1.2.2를 이용하였고 최적화 하여 컴파일 하였다. 시스템의 구성도는 그림 10과 같다.

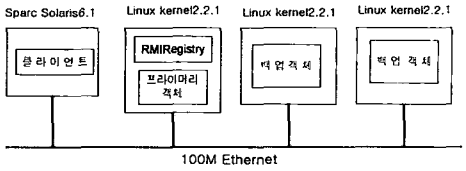

그림 10 실험에 사용된 시스템 구성도

Fig. 10 Fault-tolerant RMI system

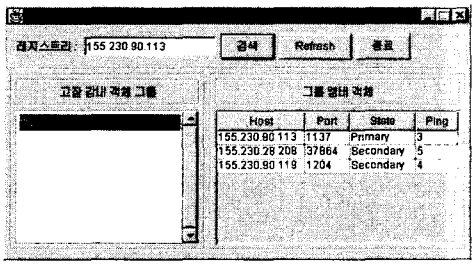

그림 11 고장 감내 객체 브라우저 실행 화면

Fig. 11 Fault-tolerant RMI object browser

프라이머리 객체와 rmiregistry를 같은 호스트에서 동작 시켰으며 백업 개체를 다른 두 호스트에서 실행하였다. 그림 11은 본 논문에서 제작한 프로그램을 이용하여 동작 중인 고장 감내 객체들의 상태를 보여준다.

실험에 사용된 고장 감내 객체는 일정 크기의 문자열을 저장하고 받아오는 함수를 정의한 원격 인터페이스를 구현하였다. 인터페이스의 정의는 그림 12와 같다.

```
interface FTfoo extends Remote
{
   public void set_Str(String data)
            throws RemoteException;
   public String get_Str()
            throws RemoteException;
}
```

그림 12 실험에 사용된 원격 인터페이스

Fig. 12 Remote interface

실험에 사용될 FTfoo인터페이스는 set_Str과 get_Str 두개의 함수로 구성된다. set_Str 함수는 주어진 일정 크기의 문자열을 서버객체에 저장하는 함수이고, get_Str 함수는 서버객체로부터 멤버변수에 저장된 문자열을 읽어오는 함수이다. set_Str 함수는 내부의 상태를 변화시키므로 프라이머리 객체에 의해 백업 객체로 상태가 멀티캐스트된다. 이러한 멀티캐스트 통신은 응답시간을 지연시킨다. 반면에 get_Str 함수는 단순히 객체 내부의 상태를 읽어오는 함수로서 내부의 상태를 변화시키지 않으므로 백업 객체로의 상태전달은 없다.

FTfoo인터페이스를 이용하여 다음의 2가지 실험을 하였다. 첫 번째 실험은 고장 감내 RMI객체에서 고장 발생에 대한 객체의 복구를 프라이머리객체 R0와 백업객체 R2의 고장과 네트웍의 고장에서 발생할 수 있는 백업 객체 R1과 R2의 동시 고장에 대해 실험하고 고장 복구 시간을 측정하였다. 두 번째 실험은 FTfoo 인터페이스를 구현한 표준 RMI객체와 고장 감내 RMI객체의 응답시간을 비교하였다.

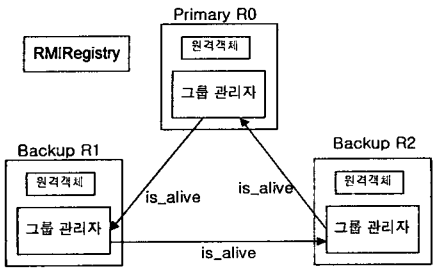

그림 13 고장 복구 실험의 시스템 구성도

Fig. 13 Fault recovery experimentation

고장 복구실험에서 프라이머리 객체의 고장의 복구는 다음과 같다. 그림 13에서와 같이 프라이머리 객체 R0와 백업객체 R1, R2가 그룹을 이루며 고장 탐지를 수행할 때, 프라이머리 객체 R0의 고장은 백업객체 R2에 의해 탐지되어 백업객체 R1에 보고된다. 프라이머리 객체의 고장을 보고 받은 R1은 그룹의 새로운 그룹 뷰를 형성하고 나머지 백업객체에 그룹 뷰를 멀티캐스트 한다. 마지막으로 R1은 자신을 프라이머리 객체로 설정하고 rmiregistry에 등록한다. 이 실험에서 고장의 탐지에서부터 백업객체 R1이 rmiregistry에 자신을 등록하는데 걸린 시간은 112ms 였다.

두 번째 고장의 경우로서 백업객체 R2의 고장은 R1에 의해 탐지되며 이는 프라이머리객체에 보고된다. 프라이머리객체는 새로운 그룹 뷰를 형성하고 이를 백업객체에 멀티캐스트 한다. 프라이머리 객체가 고장 탐지에서부터 새로운 그룹 뷰를 멀티캐스트 하는데 54ms 소요 되었다.

마지막으로 백업객체 R1과 R2가 동시에 고장이 발생하였을 때, 프라이머리 객체는 백업객체 R1의 고장만을 탐지하게 된다. 그러나 새로운 그룹 뷰를 형성하는 과정에서 백업객체 R2의 고장이 탐지되고 프라이머리 객체는 자신을 유일한 멤버로 가지는 그룹 뷰를 가지게 된다. 이 경우 고장 복구시간은 10ms 였다.

표 1 고장 복구 시간

Table. 1 Fault recovery time

고장의 종류	복구 시간(ms)
프라이머리객체 R0	112
백업 객체 R2	54
백업 객체 R1,R2	10

실험의 결과에서와 같이 프라이머리 객체의 고장은 rmiregisry에 대한 등록시간과 백업객체가 프라이머리 객체로의 전환에 필요한 시간 때문에 백업객체의 고장 복구 시간보다 크게 나타났다. 그리고 백업객체 R1과 R2의 동시고장은 프라이머리 객체가 그룹 뷰를 멀티캐스트 할 필요가 없으므로 고장 복구 시간이 짧게 나타났다.

실험을 통해서 본 시스템은 프라이머리객체와 백업객체의 고장에 대해서 잘 동작하였으며, 네트웍의 고장과 같이 다수의 복제본에 고장이 발생하는 경우에도 복구 될 수 있음을 보였다. 그러나, 본 시스템은 rmiregistry가 단일 점 실패를 가질 수 있다는 단점을 가진다. 이 문제는 Java RMI가 클라이언트와 서버에 대해 rmiregistry의 위치에 대한 사전 지식을 요구하므로 rmiregistry의 이동성이 제한되기 때문에 단순히 rmiregistry의 복제를 통해서는 해결될 수 없다. [9]에서는 모든 클라이언트와 서버가 위치하는 모든 호스트에 rmiregistry의 복제본을 생성하고 동기화 함으로서 해결하였다. 그림 14는 첫

번째 실험의 결과를 나타낸다.

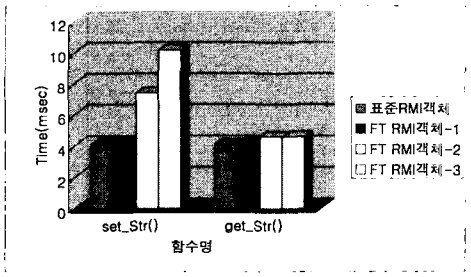

그림 14 그룹크기에 따른 함수의 응답속도 비교

Fig. 14 Function response time with group size

메시지 크기를 8문자로 하고, 복제본의 크기를 1, 2, 3으로 변화시키면서 표준 RMI객체 와 응답시간을 비교하였다. set_Str 함수는 그룹 크기가 1일 때, 표준 RMI객체와 비교하여 거의 비슷한 응답 속도를 나타내거나 2-3% 늦은 응답속도를 나타내는데 이것은 고장탐지를 위한 쓰레드의 영향 때문이다. 그리고, 그룹크기가 1에서 3까지 증가시켰을 때, 고장 감내 객체의 응답시간은 비례적으로 증가하는데, 이것은 프라이머리 객체의 상태가 백업 객체로 멀티캐스트하기 위하여 소비되는 시간이다. 그러나 get_Str 는 상태전달을 위한 그룹통신이 발생하지 않으므로 그룹크기에 상관없이 일정크기의 응답시간을 가진다.

그룹크기가 1인 set_Str 함수와 get_Str 함수가 비슷한 응답시간을 보이는데 이것은 set_Str 함수는 클라이언트 측에서 서버 측으로 매개변수로 문자열을 직렬화시켜 전달하고, get_Str 함수는 서버 측에서 클라이언트 측으로 리턴 값으로 문자열을 직렬화시켜서 전달하므로 비슷한 부하를 가지기 때문이다.

그림 15는 그룹의 크기를 1, 2, 3으로 하고, 메시지 크기를 8, 512, 1024 문자로 하여 응답시간을 측정하였을 때의 결과이다. 또한 표준 RMI객체에 대해서도 동일한 크기의 메시지를 전송하는 대한 시간을 측정하였다. 결과는 메시지 크기의 증가에 따라 응답시간이 비례적으로 증가하였다.

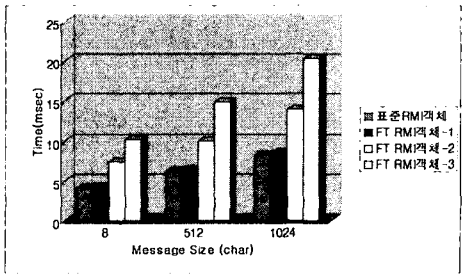

그림 15 메시지 크기에 따른 응답 속도 비교

Fig. 15 Function response time with message size

본 시스템에서 객체간의 상태전달을 위한 메시지를 구성하기 위해 객체 직렬화를 사용하였다. 그러나 [9]에서 제시된 것처럼 데이터의 복사와 비효율적이 버퍼링 때문에 객체의 직렬화와 연관된 고도의 오버헤드가 RMI에 존재한다. 멀티캐스트 방법을 선택하는데 있어서 원격 인터페이스를 통한 방법은 프로토콜의 단순화를 위하여 효율성을 희생하였다는 것을 고려할 때 적당한 성능을 보였다.

5. 결 론

본 논문에서는 Java RMI 객체에 신뢰성과 가용성을 향상시키기 위하여 고장 감내 메커니즘을 구현한 그룹 관리자와 원격 인터페이스를 설계하고 구현하였다. 그룹 관리자는 각 객체와 함께

생성되며 그룹의 고장 탐지 및 복구와 상태 전달을 담당한다. 복제의 기본 단위는 객체이며 이러한 복제본들은 그룹을 형성한다.

프라이머리 객체는 그룹의 고장 복구시 중재자 역할을 담당하며, 프라이머리 객체 고장 시 그룹 뷰에 의해 형성되는 가상 링의 순서에 의해 백업 객체가 프라이머리 객체 역할을 대신한다. 프라이머리 객체는 RMIRegistry에 등록되며, 클라이언트와 백업 객체는 레지스트리로부터 프라이머리 객체의 객체 참조를 가져와서 원격 호출과 그룹 등록을 한다. 백업 객체는 FTNaming 클래스를 이용하여 투명하게 그룹 참여를 할 수 있다. 분산 응용 개발자는 고장 감내형 RMI 클래스를 상속하여 상태 전달 인터페이스만을 구현함으로써 간단히 고장 감내 RMI 객체를 개발할 수 있다.

참 고 문 헌

[1] Object Management Group. "The Common Request Broker : Architecture and Specification," http://www.omg.org, 1999

[2] N. Brown, C. Kindel, Distrbuted Component Object Model Protocol - *DCOM/1.0. Internet Draft*, 1996

[3] Sun Microsystems, *Remote Method Invocation Specification*, 1999

[4] IONA and Isis, *An Introduction to Orbix+Isis*, IONA Technologies Ltd. and Isis Distributed Systems, 1994

[5] Narasimhan, P. Moser, L.E., Melliar-Smith, P.M., "The interception approach to reliable distributed CORBA object", *In Proceedings of USENIX Third Conference of Object-Orientde Technologies and System (COOTS'97)*, 1997

[6] P. Felber, B. Garbinato, R. Guerraoui, "owards Reliable CORBA: Intergration vs. Service Approch," *The Proceedings of 11th European Conference on Object-Oriented Programming*, 1996

[7] Guang-Way Sheu, Yue-Shan Chang, Deron Liang, Shyan-Ming Yuan, and Winston Lo, "A fault-tolerant object service on CORBA," *In Proceedings of the 17th International Conference on Distributed Computing*, 1997

[8] N. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "Interception in the Aroma System," *Proceedings of the ACM 2000 Java Grande Conference*, 2000

[9] Arash Baratloo, P. Emerald Chung, Yennun Huang, Sampath Rangarajan, Shalini Yajnik, "Filterfresh: Hot Replication of Java RMI Server Objects.," *In Proceedings of the 4th Conference on Object-Oriented Technologies and Systems (COOTS98). Santa Fe, NM, USA*, pp 65-78, 1998.