제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

모바일 호스트 상에서 수행되는 RMI

김 다 정*, 강 대 욱
전남대학교 전산학과
e-mail : myairbob@hotmail.com

The Remote Method Invocation on the Mobile Host

Da-Jeong Kim*, Dae-Wook Kang
Dept. of Computer Science, Chonnam National University

요 약

무선 링크는 유선에 비해 대역폭이 낮고 에러율이 높은 고유한 특성에 기인한 단절이 자주 발생하게 된다. 분산 응용 구축 수단인 RMI는 메소드 호출 중에 발생하는 단절로 인해 심각한 성능 저하가 유발된다. 하지만 기존의 RMI는 단절에 대한 어떠한 대책도 마련하지 않고 있다. 본 논문은 RMI의 하부 구조를 수정하여 RMI가 무선 링크의 단절에 적응하도록 한다.

1. 서론

RMI(Remote Method Invocation)는 CORBA, DCOM처럼 클라이언트/서버 기반의 분산 어플리케이션을 개발하기 위한 중요한 수단이다. RMI는 RPC(Remote Procedure Call)의 개념을 Java로 확대한 것이다. RMI는 서로 다른 주소 공간 심지어 서로 다른 머신 상에 위치한 메소드 호출을 지원한다는 점에서 RPC와 유사점을 가지지만 Java의 객체 지향적인 성격을 그대로 상속받게 된다는 점에서 차이점을 갖는다.

최근 들어Personal Device Assistants(PDA), 무선 LAN 카드을 장착한 노트북과 같은 네트워크 상의 모바일 호스트가 증가하게 되었다. 기존의 RMI는 네트워크 상의 모든 호스트가 정적이고, 하드웨어 상에 문제가 발생하는 경우를 제외하고 항상 클라이언트의 요청이나 서버의 응답이 실패없이 전달될 수 있다고 가정하여 무선 링크를 통해 발생하는 호출에 대해서는 고려하지 않고 있다[1]. 따라서 무선 환경에서 동작하는 RMI에서 위와 같은 가정은 유효하지 못하다. 무선 호스트를 연결하는 무선 링크는 유선 링크에 비해서 낮은 대역폭과 높은 에러율로 인해 단절이 자주 발생하기 때문이다.[2].

일시적인 무선 링크의 단절은 무선파가 전달되는 경로 상에 존재하는 방해물 등으로 인해 무선 신호의 감쇄가 일어나는 경우나 현재 셀로부터 새로운 셀로 모바일 호스트가 이동하는 핸드오프 상황에서 발생한다.

무선 링크의 단절은 RMI에서 메소드 호출 실패로 나타나 성능을 저하 시킨다[3]. 현재의 RMI는 이러한 상황을 통제할 어떤 기능도 가지고 있지 않다. 무선 링크 상에서 RMI의 성능을 향상시키기 위해서는 호스트의 물리적인 위치나 이동성과는 무관하게 단절 상황에서도 모바일 호스트가 계속 연산을 수행할 수 있도록 해주는 단절 상황 처리(Disconnected Operation)가 필요하다[4].

단절 상황에서의 연산은 장애 등으로 인하여 네트워크를 사용할 수 없는 경우나 네트워크를 사용이 고비용인 경우에 적용할 수 있는 방법이다. 단절 상황 처리는 단절 자체를 개선하는 것이 아니라 단절 상황에 RMI의 동작을 적응시키는 방법(Adaptation)이다. 메소드 호출의 시맨틱을 구체적으로 정의하는 RMI의 원격 레퍼런스층(Remote Reference Layer)을 수정하면 단절 상황 처리를 지원이 가능하다. 이 방법은 RMI의 하부 구조만을 수정하기 때문에 분산 어플리케이션 작성하는 개발자가 직접 프로그램에 단절에 관련된 코드를 삽입할 필요가 없고 이전에 작성된 RMI 프로그램의 재사용이 가능한 방법이다.

본 논문은 단시간 단절 상황에서 모바일 호스트가 RMI를 사용할 때의 성능이 유선에서 RMI를 사용하는 때의 성능 간의 격차를 줄이기 위해 단절 상황 처리를 적용한다. 설정된 상황은 그림1과 같이 유선 네트워크에 연결된 고정 호스트인 서버와 모바일 호스트인 클라이언트로 구성된다. 예를 들어, 병원의 경우에 의사가 모바일 호스트인 PDA를 통해 환자의 상태를 무선으로 전달하고 환자의 약물투여 시간이나 빈도 등의 데이터를 고정된 호스트의 데이터베이스에서 얻

어온다고 가정하자. 의사는 병동을 이동하며 여러 AP(Access Point)을 거쳐가게 되고 AP가 관할하는 영역 간의 이동의 있을 때마다 핸드오프가 발생한다. 또한 병원 내에 있는 여러 장비들에 의해서 방출되는 전자기파등에 의한 노이즈에 의해 무선 시그널의 변화가 일어날 수 있다. 이런 현상들은 단 시간의 네트워크 실패를 일으키게 되어 진행 중인 연산의 오작동이 가능하다.

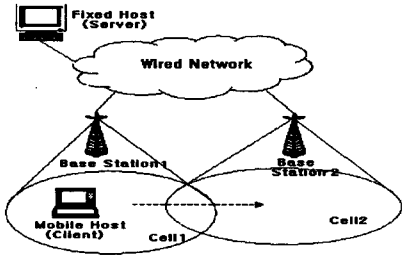

그림 1. 어플리케이션 모델

2. 관련 연구

최근 무선 네트워크에서의 분산 기술의 신뢰성과 성능을 향상시키기 위한 많은 연구들이 있어왔다.

호스트의 이동성을 지원하기 위해 자바 MobileRMI [5]는 RMI 시스템이 자동적으로 원격 레퍼런스를 갱신시키는 방법을 사용하였다. MobileRMI는 클라이언트에서 서버로의 메소드 호출을 수행하기 위해 TCP기반의 스트림 프로토콜을 사용하는 원격 객체인 UnicastRemoteObject대신 이를 모바일 환경을 위해 확장한 MobileUnicastRemoteObject(MURO)를 사용한다. MURO는 이동할 때 자신을 대신할 더미 객체를 남겨둔다. MURO로의 원격 레퍼런스는 그 객체가 이동하는 경로에 체인으로 연결된 더미객체를 따라 갱신된다. move()메소드를 이용하여 클라이언트가 이동을 하는 경우 본 논문처럼 클라이언트 측의 UnicastRef클래스를 수정하여 원격 레퍼런스를 업데이트 시키도록 하였다. 하지만 [5]는 본 논문과는 달리 클라이언트나 서버가 명시적으로 이동을 상대에게 통지하거나 더미 객체와 같은 트랙을 쫓아 원격 레퍼런스를 수정한다. 이는 주변 노이즈에 의해 예고도 없이 발생하는 단절 상황에 대해서는 고려하지 않았다.

상대적으로 속도가 느린 무선 링크 상에서의 RMI가 전송하는 데이터 중 분산 가비지 콜렉션 프로토콜로 인한 데이터의 비율이 실제 메소드 호출에 필요한 데이터에 비해 현저히 높다. 이를 개선하고 TCP/IP의 성능을 향상 시키기 위해 Wireless RMI[6]는 데이터 호출 부분을 제외한 나머지는 중재자(Mediator)를 통해 이루어지도록 한다. 각 어플리케이션의 이동 호스트에 위치한 에이전트와 서버에 위치한 프럭시는 중재자로서 공동 작업으로 원격 메소드 호출을 수행된다. Wireless RMI는 무선 링크에서 동작하는 RMI의 적응 방안의 하나로서 교환되는 데이터의 양을 최소화시키는 방법을 통해 느린 전송 계층을 보완하고 있지만 단절 상황에서의 계속적인 연산은 여전히 불가능하다.

3. RMI 동작 원리

이번 장은 자바 RMI 시스템의 구조와 실행 절차에 대하여 설명한다.

3.1. RMI 계층

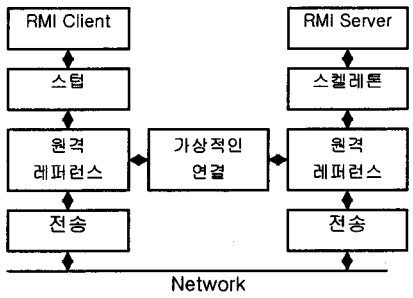

그림 2. RMI 계층

Java RMI 시스템은 그림 2와 같이 프록시 계층인 스텁(Stub)/스켈레톤(Skeleton)계층, 원격 참조(Remote Re-ference) 계층, 전송(Transport) 계층으로 구성되어 있다 [7, 8]. 각 계층은 독립적인 구조로 다른 계층에 영향을 받지 않고 수행되므로 특정 계층에 원하는 기능을 추가할 수가 있다. 스텁 계층은 원격 객체에서 정의한 동일한 메소드를 가지고 원격 객체의 클라이언트 쪽 프럭시로 원격 참조 계층을 호출함으로써 원격 객체로의 호출을 초기화하고 매개변수를 직렬화시켜 마샬 스트림에 기록하며 메소드 호출의 리턴값이 도달할 때까지 기다렸다가 서버로부터 전달되는 결과값을 마샬 스트림으로부터 역직렬화하여 클라이언트에게 전달한다. 스켈레톤 계층은 실제 원격 객체로의 호출을 해당 원격 메소드로 전달(Dispatch)시키는 서버 쪽 프럭시로서 마샬 스트림으로부터 매개변수를 언마샬링하여 실제 원격 객체 인스턴스로 원격 호출을 전달(Upcall)하고 원격 호출의 결과값을 마샬 스트림을 통해 다시 클라이언트로 전송하는 역할을 담당한다. 원격 참조 계층은 호출 시맨틱과 원격 객체에 대한 레퍼런스등을 정의하는 계층이다. 전송 계층은 원격의 주소 공간과의 연결 설정과 관리, 상태 모니터 등을 담당하는 계층이다.

3.2. RMI 메소드 호출 처리 과정

원격적으로 호출될 수 있는 객체는 java.rmi.Remote를 상속 받은 원격 인터페이스를 통해 생성된다. 서버 응용 프로그램은 이 인터페이스를 구체적으로 정의하고 있다. 클라이언트는 원격 인터페이스를 통해서만 원격 객체에 접근 가능하다. 자바 시스템은 rmic 명령어를 사용하여 원격 객체에 대한 스텁과 스켈레톤을 생성한다. 스켈레톤과 스텁은 원격 객체의 인터페이스에 정의된 것과 동일한 메소드들을 가지고 있다[9].

다음으로 원격 객체를 가상 머신 상에 Export 되어야 한다. 이는 원격 객체가 서버와 다른 가상 머신 상에서 작동하는 객체에 의해 원격적으로 호출될 수 있

게 하고 네임서버 프로세스에서도 객체를 사용할 수 있도록 하기 위한 것이다. 이때 서버의 자바 가상 머신(JVM, Java Virtual Machine)으로 서버와 클라이언트 측 전송 객체와 레퍼런스 객체, 스텁/스켈레톤 객체의 인스턴트들이 이동하게 된다. 서버가 Bind나 Rebind 메소드를 이용하여 네임서버상에 원격 객체를 등록시키면, 클라이언트 측 레퍼런스 객체와 전송 객체가 서버에 의해 네임서버로 직렬화되어 이동한다.

클라이언트가 호출하려는 객체의 스텁 객체의 직렬화된 복사본이 네임 서버에서 클라이언트로 다운로드 된다. 클라이언트가 네임서버를 통해 원격 객체의 레퍼런스를 획득하게 되면 원격 객체에 대한 호출은 이 레퍼런스를 통해서 이루어진다.

클라이언트가 원격 메소드를 호출할 경우, 스텁 객체는 클라이언트의 원격 참조 계층 객체인 RemoteRef 객체의 invoke()메소드를 수행시켜 메소드 호출을 실행한 후 결과값을 다시 클라이언트로 반환한다.

4. 단시간 단절에 적응하는 RMI

이번 장은 RMI의 원격 레퍼런스 계층을 수정하여 단절에 적응하도록 하는 방법을 살펴본다.

4.1. 원격 레퍼런스 계층 수정 이유

클라이언트가 로컬 객체의 메소드를 호출하는 것과 동일한 방식으로 원격 객체의 메소드를 호출하기 위해서는 원격 객체의 레퍼런스를 얻어야 한다. 하지만 원격 객체는 로컬 객체와는 달리 원격 가상 머신 내에 있고 원격 객체에 대한 레퍼런스(Reference)는 원격 가상 머신의 메모리에 있는 원격 객체의 위치를 나타내기 때문에 다른 머신 상에서는 이 값이 무의미하다. 이에 클라이언트는 원격 객체에 대한 레퍼런스가 아닌 원격 객체의 프럭시인 스텁 객체의 로컬 레퍼런스를 통해 원격 객체에 접근한다. 클라이언트는 스텁으로 원격 객체에 대한 호출을 전달하면 스텁은 이를 다시 원격 참조 계층을 통해서 원격 객체로 전달한다.

원격 참조 계층은 원격 레퍼런스를 생성/관리하며 클라이언트가 알고 있는 스텁의 레퍼런스와 원격 객체의 레퍼런스 사이의 맵핑(Mapping)을 담당한다. 원격 참조 계층이 맵핑 과정을 추상화시킨 객체가 RemoteRef 객체이다.

클라이언트가 스텁 객체를 통해서 원격 메소드 호출하면 3장의 RMI의 호출 과정에서 살펴본 것처럼 RemoteRef객체는 invoke() 메소드를 호출시킨다. invoke() 메소드는 우선 원격 객체의 이름과 객체 내 연산을 매개 변수로 하여 새로운 연결(Connection)을 초기화하고 RemoteCall 인터페이스 타입의 객체를 리턴한다. 그리고 이렇게 생성된 연결을 통해 원격 메소드를 실행시키기 위하여 매개 변수를 RemoteCall 객체로 마샬링하여 RemoteCall 객체의 executeCall() 메소드를 실행시킨다[10].

RemteRef.invoke() 메소드는 원격 메소드 호출이 스트림으로 마샬링되어 데이터가 링크를 통해 전달되기 직전에 클라이언트에서 수행되는 메소드이기 때문에 단절 상황 처리 invoke()메소드에 추가시키는 것이 바람직하다.

4.2. 원격 레퍼런스 계층 수정 방법

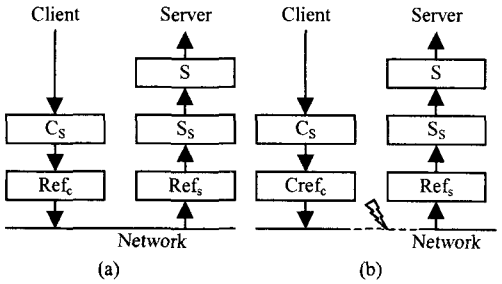

그림 3. 기존의 RMI 와 수정된 RMI

링크가 단절된 상태에서 원격 메소드가 호출되면 수행이 실패하게 된다. 단절 상태에서의 연산 수행 방법으로 이 논문은 각 메소드 호출시 마다 생성되는 RemoteCall객체를 캐싱하는 방법을 제안한다. 이 기술은 사용자의 관점에서 보면 네트워크가 단절된 상태에서도 메소드 호출이 원활하게 이루어지고 있는 것처럼 보인다. 하지만 실제로는 호출을 지연시키기 위해 리스트를 사용함으로써 응용 프로그램이 네트워크와 투명하게 작동하게 된다.

기존의 RMI는 그림 3(a)에서 보는 것과 같다. 서버는 서버 객체 S를 생성한다. 클라이언트 스텁 객체(C_s), 서버 스켈레톤 객체(S_s), 클라이언트 측 레퍼런스와 전송 객체(Ref_c)와 서버측 레퍼런스와 전송 계층(Ref_s)이 S의 생성을 통해 서버의 JVM으로 export된다. 서버가 네임서버 상에 C_s를 바인드시키면 C_s와 Ref_c가 네임서버로 마샬링되고 클라이언트의 lookup 요청이 네임서버로 전달되면 클라이언트로 C_s와 Ref_c가 이동한다. 이 객체들은 lookup 연산이 다 끝난 후 레퍼런스를 통해 서로 연결(hold across)되고 다음에 이어지는 동일한 메소드로의 호출은 이렇게 생성된 링크를 통해 이루어진다.

이런 하부 구조에 클라이언트상의 캐싱을 추가하기 위해 그림 3(b)에서처럼 이 논문은 Ref_c 를 수정한 $CRef_c$ 객체를 사용하였다. $CRef_c$는 아래와 같은 방식으로 수정을 된다.

```
pulbic class UnicastRef implements RemoteRef {
    public Object invoke() throws Exception{
        LinkedList forcall
        ……
        while{link state in not available} [
            forcall.add(RemoteCall object);
        }
        if(forcall != null) {
            while(forcall<=forcall.size()) {
                RemoteCall remotecall;
                remotecall = forcall.getFirst();
                remotecall.executeCall();
            }
        }
```

```
        RemoteCall.executeCall();
        ......
    }
}
```

서버 객체의 생성, Export, 바인딩, Lookup은 여전히 기존의 RMI와 동일한 방식으로 이루어진다. $CRef_c$객체는 Ref_c객체와 마찬가지로 매 원격 메소드 호출마다 RemoteCall 객체를 새로 생성한다. 스텁은 기존의 Ref_c 객체가 바뀐 것을 알지 못한 채 원격 메소드의 매개변수를 이전과 마찬가지로 RemoteCall 객체로 마샬링한다. 커넥션의 링크가 단절되면 RemoteCall 객체는 리스트상에 저장되었다가 네트워크가 데이터를 보낼 수 있는 상태가 되면 리스트에 삽입된 순서에 따라서 차례대로 RemoteCall 객체의 executeCall()을 실행한다. 링크가 복구되고 리스트상에 모든 RemoteCall 객체의 실행을 마친 후에 RMI는 RemoteCall을 리스트에 저장하지 않고 기존의 방법과 마찬가지로 메소드 호출을 시행한다. 서버 측 Ref_s 객체가 메소드 호출의 결과값을 네트워크를 통해 클라이언트로 전달할 때 발생하는 단절에 대한 처리도 Ref_c과 동일한 방식으로 이루어지기 때문에 본 논문은 $CRef_c$객체에 대해서만 고려하여 문제를 단순화시킨다.

5. 테스트 환경

실험은 고정 호스트인 서버와 Orinoco Wireless network Interface Card(11Mbps)를 장착한 모바일 호스트인 클라이언트 사이에서 오고 가는 데이터를 패킷 분석 프로그램인 ethereal를 이용하여 수집하여 분석한다. 서버와 클라이언트는 팬티업 IV 시스템에 운영체제는 리눅스를 이용한다. 클라이언트상의 리눅스용 JDK 1.3.1 소스 배포판을 설치하여 수정한다.

실제 무선 환경에서의 실험은 단절 시간에 대한 제어가 어렵고 실험의 재현이 불가능하기 때문에 이 대신 유선 환경에서 무선 상의 단절을 연출하는 프로그램을 수행시켜서 결과를 얻는다.

서버 상에서 RMI 네임서버를 데몬 상태로 실행시키고 서버 프로그램을 구동시킨다. 클라이언트에서는 단절 시간과 전송되는 데이터의 크기를 변화시키면서 메소드 호출을 시작한다.

단절 시간 및 상대방으로 보내지는 데이터의 크기에 따라서 성능을 차이를 알아보기 위해 기준 시간 동안 증가한 TCP 세그먼트의 일련 번호를 측정하여 수정된 RMI와 기존의 RMI의 성능을 비교한다.

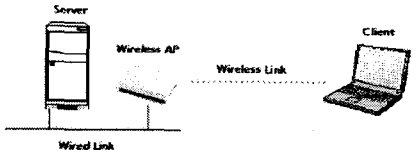

그림 4. 테스트 환경

6. 결론 및 향후 연구

무선을 이용한 이동 컴퓨팅은 여러 분야에 적용될 수 있지만 무선환경에서의 단절은 피할 수 없는 문제이다. 기존의 RMI는 모든 호스트가 정적이라고 가정하고 연산을 수행하기 때문에 무선 상황에서의 RMI 성능은 사용자를 만족시키지 못한다. 이에 본 논문에서는 RMI를 수정하였고 아래와 같은 장점을 가진다.

첫째로 기존에 RMI로 작성된 프로그램에 어떠한 수정을 가하지 않아도 될 뿐더러 새로운 RMI 프로그램을 작성할 때도 기존과 동일한 방식을 사용하면 된다.

둘째로 사용자는 네트워크에서 단절이 일어날 수 있는 상황에 대해 어플리케이션 단계에서의 어떠한 처리를 추가하지 않아도 시스템은 단절을 투명하게 처리할 수 있게 되었다.

하지만 링크 상태를 측정하기 위해 사용되는 추가적인 처리 절차는 기존 RMI에 비해 오버헤드가 발생하게 된다. 이로 인해 단시간 단절이 일어나지 않는 안정적인 곳에서도 링크 상태를 확인하기 위해 걸리는 시간과 오버헤드로 인해 수정된 RMI가 오히려 더 낮은 성능을 보일 수 있다.

참고 문헌

1. B.R.Badrinath and P.Sudame. "To Send or not to Send: Implementing Deferred Transmissions in a Mobile Host." Proceedings of the 16th International Conference on *Distributed Computing Systems*. 1996. *Hong Kong*: IEEE.
2. Xiaohua, Lee Man Kei, and Jia. "An Efficient RPC scheme in Mobile CORBA Environment." International Workshop on Parallel Processing. 2000. Toronto, Canada.
3. G.Welling and M.Ott. "Structuring Remote Object Systems for Mobile Hosts with Intermittent Connectivity." The 18th International Conference on Distributed Computing Systems. 1998. Amsterdam, The Netherlands: IEEE.
4. P.Honeyman and L.B.Huston, "Communications and Consistency in Mobile File System." 1995, Center for Information Technology Integration University of Michigan.
5. A.Vecchio, M.Avvenuti, and. "Embedding Remote Object Mobility in Java RMI." Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing System. 2001. Bologna,Italy.
6. S.Campadello, O.Koskimies, H.Helin, K.Raatikainen, and S.Ltd. "Wireless Java RMI." Fourth International Enterprise Distributed Object Computing Conference. 2000. Makuhri, Japan.
7. A.Wollrath, R.Riggs, and J.Waldo. "A Distributed Object Model for the Java System." 2nd Conference on Object-Oriented Technologies. 1996. Toronto, Ontario, Canada: USENIX.
8. "Java Remote Method Invocation Guide." http://java.sun.com/j2se/1.4/docs/guide/rmi/index.html
9. T.B.Downing, "Java RMI: Remote Method Invocation. 1998: IDG Books
10. A.Baratloo, P.E.Chung, and Y.Huang. "Filterfresh: Hot Replication of Java RMI Server Objects." Proceeding of the 4th USENIX conference on Object-Oriented Technologies and Systems. 1998. Santa Fe, New Maxico: USENIX.