모바일 호스트에서 수행되는 RMI의 설계 및 구현

*김다정, 김용수, *강대욱
*전남대학교 전산학과, 전남과학대학 정보통신과

A design and Implementation of the Remote Method Invocation on the Mobile Host

*Da-Jeong Kim, Yong-Su Kim, *Dae-Wook Kang,
*Chonnam national university, Department of Computer Science
Chonnam science university, Department of Information & Communication Engineering

Abstract

Due to low bandwidth and high error rate, the wireless Link is more frequent disconnected than the wired link. Link failure in the method invocation often appears as a failed method invocation. Framework of RMI is adjusted with queuing of method call. RMI's performance can be enhanced without breaking compatibility with Java RMI specification.

1. 서론

분산 컴퓨팅은 서버에서 객체를 생성하고 그것에 대한 메소드 호출은 클라이언트에서 수행되는 것처럼 서로 다른 주소 공간에서 연산을 수행됨을 의미한다. 자바 RMI는 순수 자바로 작성된 분산 응용 구축 수단으로 RPC (Remote Procedure Call)의 개념을 자바로 확대한 형태이다. RMI는 객체 지향적 성격을 상속받아 개발자가 프로토콜이나 통신에 관련된 작업을 직접 처리할 필요가 없게 되어 소켓 프로그래밍 방식에 비해 코딩 과정의 에러를 최소화시킨다. 또한 RMI는 다른 분산 어플리케이션 기술인 CORBA, DCOM처럼 IDL (Interface Definition Language)이 필요없고 자동적인 가비지 콜렉션을 지원하며 예외상황 처리가 용이하다.

무선 네트워크의 사용 증가와 함께 모바일 네트워크 상에서 RMI의 사용 사례 또한 증가하고 있다. 하지만 기존의 RMI는 네트워크 상의 모든 호스트가 정적이고 하드웨어 상에 문제가 발생하는 경우를 제외하고 항상 클라이언트의 요청이나 서버의 응답이 아무 장애없이 서로에게 전달될 수 있다고 가정하였다[1]. 그러나 유선 네트워크에 비해 낮은 대역폭과 높은 에러율을 가지고 있는 모바일 네트워크[2]에서는 위와 같은 가정은 유효하지 못하다. 무선 링크는 주위 장애물로 인한 장애물로 인한 신호 감쇄, 핸드오프, 한정된 베터리량, 등으로 인해 빈번히 단절이 발생할 수 있다.

무선 네트워크에서 발생하는 잦은 단절은 패킷이 재전송되는 시간동안 RMI의 메소드 호출 처리 시간이 증가한다[3]. 따라서 추가적인 오버헤드가 발생하더라도 호스트의 물리적 위치나 이동성과는 무관하게 단절 상황에서 모바일 호스트가 계속적인 연산을 수행하기 위한 처리가 필요하게 된다[4].

이에 본 논문은 단절시 클라이언트에서 발생하는 메소드를 클라이언트 머신 상에 캐싱해두었다가 링크가 복구되었을 경우 다시 캐싱된 메소드를 저장된 순서대로 메소드 호출을 재개하는 방법을 선택하고 이를 위해 메소드 호출의 시맨틱을 구체적으로 정의하는 RMI의 하부구조를 수정하도록 한다.

2. 관련연구

무선 환경에서 충분한 성능을 발휘하지 못하는 TCP에

서 동작하는 RMI의 성능 향상을 도모하는 방법의 하나로 WirelessRMI[5]는 2개의 중간 매개자를 사용한다. RMI가 전송하는 데이터 중 분산 가비지 콜렉션 프로토콜로 인한 데이터의 비율이 실제 메소드 호출에 필요한 데이터에 비해 현저히 높다. 이에 RMI 메소드 호출은 각 어플리케이션의 이동 호스트에 위치한 에이전트와 서버에 위치한 프럭시는 중재자로서 공동 작업으로 원격 메소드 호출을 수행한다. 유선 링크에 비해 상대적으로 전송 능력이 떨어지는 무선 링크 상에서 교환되는 데이터의 양을 최소화시키는 방법을 통해 느린 전송 계층을 보완하고 있지만 단절 상황에 대한 대처방안은 모색하고 있지 않다.

3. RMI 개요

이번 장은 자바 RMI의 계층 구조와 메소드 호출 요청이 서버로 전달되어 처리되고 결과값이 클라이언트로 반환되는 과정에 대해 살펴본다.

3.1. RMI 계층 구조

Java RMI 시스템은 그림 1과 같이 프록시 계층인 스텁(Stub)/스켈레톤(Skeleton)계층, 원격 참조(Remote Reference) 계층, 전송(Transport) 계층으로 구성되어 있다.

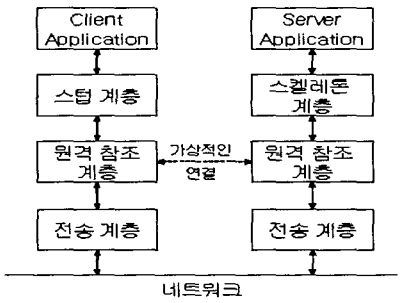

그림1. RMI 계층 구조

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 프럭시이고 스켈레톤 계층은 실제 원격 객체로의 호출을 해당 원격 메소드로 전달(Dispatch)시키는 서버 쪽 프럭시이다. 프럭시 계층은 매개변수와 메소드 호출의 결과값에 대한 마샬링과 언마샬링을 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

원격 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 객체의 인터페이스에 정의된 것과 동일한 메소드를 가진 원격 객체에 대한 스텁과 스켈레톤을 생성한다.

다음으로 원격 객체가 가상 머신 상에 Export 되어야 한다. 이는 원격 객체가 서버와 다른 가상 머신 상에서 작동하는 객체와 네임서버 프로세스에서도 객체를 사용할 수 있도록 하기 위한 것이다. 이때 서버의 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 lookup을 시행하면 호출하려는 객체의 스텁 객체의 직렬화된 복사본이 네임 서버에서 클라이언트로 다운로드된다. 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. 단절 상황에서의 RMI

이 장은 무선 상에서의 RMI가 유선 상에서 호출되는 RMI와 최소한의 차이만 나도록 하는 방안을 제안하도록 한다.

4.1. RemoteRef 객체 수정

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 객체의 프럭시인 스텁 객체의 로컬 레퍼런스를 통해 원격 객체에 접근한다. 원격 참조 계층은 원격 레퍼런스를 생성/관리하며 클라이언트가 알고 있는 스텁의 레퍼런스와 원격 객체의 레퍼런스 사이의 맵핑을 담당한다. 원격 참조 계층이 맵핑 과정을 추상화시킨 객체가 RemoteRef 객체이다.

클라이언트가 스텁 객체를 통해서 원격 메소드 호출하면 RemoteRef객체는 invoke() 메소드를 호출시킨다. invoke () 메소드는 RemoteCall 인터페이스 타입의 객체를 리턴한다. 그리고 원격 메소드를 실행시키기 위하여 매개 변수를 RemoteCall 객체로 마샬링하여 RemoteCall 객체의 executeCall() 메소드를 실행시킨다.

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.

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[6]. 이 기술은 사용자의 관점에서 보면 네트워크가 단절된 상태에서도 메소드 호출이 원활하게 이루어지고 있는 것처럼 보인다. 하지만 실제로는 단절 시간동안 호출에 관련된 객체를 캐싱함으로써 응용 프로그램이 네트워크와 투명하게 작동하게 된다.

기존의 RMI와 수정된 RMI의 메소드 호출 과정을 비교하여 도식화해보면 그림 2와 같다. 클라이언트와 서버의 프럭시 계층은 각각 C_s, S_s이고 레퍼런스와 전송 계층은 Ref_c, Ref_s이며 원격 객체는 S로 표시한다.

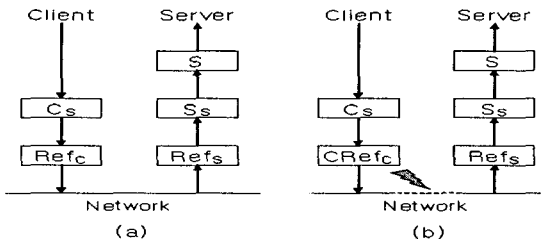

그림2. 기존의 RMI와 수정된 RMI

수정된 RMI는 그림 2에서 보이는 바와 같이 export, lookup, 바인드 과정은 모두 동일하게 발생한다. 이 객체들은 lookup 연산이 다 끝난 후 레퍼런스를 통해 서로 연결(hold across)되고 다음에 이어지는 동일한 메소드로의 호출은 이렇게 생성된 링크를 통해 이루어진다.

그림 2(b)는 RMI자체가 네트워크 상태에 따라 투명하게 메소드 호출을 처리하도록 메소드 호출이 클라이언트 상의 하부 구조인 Ref_c 대신 캐싱을 추가한 $CRef_c$를 통해 이루어진다는 것이다. 서버 측 원격 레퍼런스 객체인 $CRef_s$도 동일한 방식으로 메소드 매개변수를 획득하여 결과값을 다시 반환하기 때문에 본 논문은 클라이언트 측 레퍼런스의 수정만으로 캐싱의 성능 면에서의 이점을 증명하고자 한다. Ref_c는 아래와 같은 방식으로 수정된다.

```
public class UnicastRef implements RemoteRef {
  public object invoke() throws Exception {
    ……
    while(link state is not available) {
      enqueue RemoteCall Object;
    }
    if(queue is not null) {
     cashed method's RemoteCall.executeCall();
     }
     RemoteCall.executeCall();
     ……
}}
```

4.2. 시스템 적용 결과

실험은 고정 호스트인 서버와 무선랜을 장착한 모바

일 호스트인 클라이언트 사이에서 오고 가는 데이터를 패킷 분석 프로그램인 ethereal를 이용하여 수집하여 분석한다. 서버와 클라이언트는 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 연출하여 결과를 얻는다.

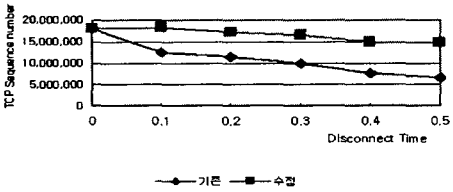

그림 3. 단절 시간에 따른 성능비교

기존의 RMI와 원격 레퍼런스 계층이 수정된 RMI의 단절 시간동안의 신뢰성을 비교하기 위해 일정한 크기(4K)의 데이터를 일정 시간(60초)동안 네트워크를 통

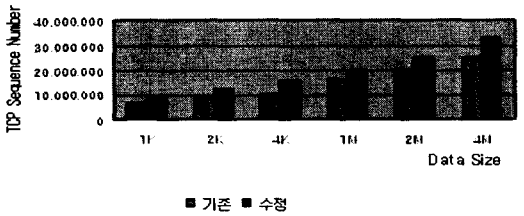

그림 4. 데이터 크기에 따른 성능 비교

해 전달했을 때 TCP 시퀀스 번호를 비교하도록 한다.

그림3에서처럼 단절 시간이 길어질수록 수정된 RMI가 더 많은 패킷을 전달하고 있음을 알 수 있다.

그림4는 0.0~0.5초 사이의 임의의 단절 시간을 발생했을 때 데이터 크기가 변화함에 따른 기존 RMI와 수정된 RMI의 비교 결과이다. 수정된 RMI가 기존 RMI에 비하여 41.6%의 성능 향상을 얻었다.

5. 결론 및 향후연구

무선환경에서의 단절은 불가피한 문제이다. 기존의 RMI는 모든 호스트가 정적이라고 가정하고 연산을 수행하기 때문에 무선 상황에서는 이런 가정이 유효하지 못하다. 본 논문은 링크가 단절된 경우에 캐싱 기능을 추가하여 RMI를 단절 상황에 적응시켰다. 이 방법은 역호환성(backward compatibility)을 유지하며 사용자가 어플리케이션 계층에서의 특별한 부가 처리없이 시스템은 단절을 투명하게 처리한다는 장점을 가진다.

단절 상황에 관련된 처리를 하기 위해서는 단절 발생 가능성에 대한 예측이 선행되어야 한다. 향후 연구에서는 단절이 발생하기 전에 링크를 통해 전달되는 신호의 질과 강도를 통해 링크 상태를 측정하고 단절 여부를 파악하는 정확한 리소스 상태 모니터를 구축하는 방안이 전제되어 있다.

Reference

1. B.R.Badrinath, P.Sudame. "*To Send or not to Send: Implementing Deferred Transmissions in a Mobile Host*". in the *Proceedings of the 16th International Conference on Distributed Computing Systems*. 1996. Hong Kong

2. Jie Xiaohua, Lee Man Kei. "*An Efficient RPC scheme in Mobile CORBA Environment*". in the *Proceedings of the 2000 International Workshop on Parallel Processing*. 2000. Toronto, Canada

3. G.Welling, M.Ott. "*Structuring Remote Object Systems for Mobile Hosts with Intermittent Connectivity*". in the *The 18th International Conference on Distributed Computing Systems*. 1998. Amsterdam, The Netherlands

4. P.Honeyman, L.B.Huston, "*Communications and Consistency in Mobile File System*". 1995, Center for Information Technology Integration University of Michigan.

5. S.Campadello, O.Koskimies, H.Helin, K.Raatikainen, and S.Ltd. "*Wireless Java RMI*". in the *Fourth International Enterprise Distributed Object Computing Conference*. 2000. Makuhri, Japan

6. A.D.Joseph, A.F.deLespinasse, J.A.Tauber, D.K.Gifford, and M.F.Kaashoek. "*Rover: A Toolkit for Mobile Information Access*". in the *Proceedings of the Fifteenth Symposium on Operating Systems Principles*. 1995