# 무선 환경의 RMI 성능 개선

김현호 강대욱
전남대학교 전산학과
{freesite@cs, dwkang@chonnam}.chonnam.ac.kr

# Improvement of RMI Performance in Wireless Environments

Hyeon-Ho Kim Dae-Wook Kang
Dept. of Computer Science, Chonnam natl. univ.

**요 약**

무선 환경의 핸드오프로 인한 단시간 단절 또는 비트오류는 잦은 패킷 손실을 야기한다. TCP는 이를 패킷 폭주로 오인, 혼잡 제어 정책을 수행함으로써 링크가 다시 호전되더라도 곧바로 데이터 전송을 재개하지 못한다. 이 논문에서는 링크 상태를 파악하여 패킷 손실이 명백한 링크 단절 상황에서는 전송을 유보하고 링크가 복구된 후 즉시 전송을 시작하여 혼잡제어 정책을 피함으로써 전송 효율을 높이는 방안을 제시한다. 이를 위해 미들웨어 계층인 RMI의 동작을 확장하여 구현하고 실험을 통해 효율성을 확인한다.

## 1. 서 론

네트워크의 확장과 더불어 분산 시스템의 사용과 대화형 응용 프로그램 개발에 있어서 객체 기술의 사용이 점차 증가하고 있다. Java와 RMI(Remote Method Invocation)는 서로 다른 하드웨어와 소프트웨어 플랫폼에서 발생하는 문제를 투명하게 처리하고 자바 가상머신 위에서 작동중인 클라이언트 응용프로그램이 원격 자바 가상머신에서 구현된 서버 객체를 로컬 객체와 같은 방법으로 호출해서 사용할 수 있도록 지원해 주기 때문에 분산 서비스 응용프로그램 개발에 흔히 이용된다[1].

무선망의 보급으로 이용이 증가하고 있는 이동 호스트에서는 다음과 같은 단절 현상이 발생한다[2][3]. 첫 번째로 전파의 방해물이나 신호 감쇄에 의해 무선 신호가 사라지는 경우이다. 두 번째로는 중첩이 없는 무선 셀을 이동하거나 허용 주파수 변화로 인한 경우이다. 세 번째는 에너지 절약을 위해 일시적인 시간 동안 자동 혹은 수동적인 방법에 의해 단절 모드로 변환하는 것이다. 네 번째는 이동 호스트가 무선 셀 영역 밖에 있어 stand alone 상태로 변환된 경우이다.

첫 번째와 두 번째 같은 단시간 혹은 일시적인 단절은 무선상에서 흔하게 발생한다. 이러한 단절로 패킷이 손실된 경우에도 TCP와 같은 신뢰성있는 프로토콜에서는 혼잡이 발생한 것으로 인식하여 패킷의 재전송, 전송 윈도우 크기 감소, 재전송 타이머 설정 등의 혼잡 제어 정책을 수행하게 된다[2]. 이것은 실제 전송 가능한 상태가 되더라도 곧바로 전송을 할 수 없는 원인이 된다.

링크가 단절되었을 경우 패킷 전송은 그대로 패킷 손실로 이어지기 때문에 이러한 상황에서는 패킷을 전송하지 않고 링크가 연결이 되었을 때 바로 전송을 시작하는 것이 바람직하다. 이 논문에서는 현재 유선 상에 널리 쓰이고 적합하게 적응하고 있는 TCP와 IP를 수정하는 대신 응용프로그램과 전송 계층 사이의 미들웨어인 RMI를 수정한다.

RMI는 총 4개의 계층으로 구성[6]되어 있지만 실제적인 메소드 호출을 다루는 Remote Reference 계층을 수정하여 링크 모니터링의 결과[7]에 따라 데이터 전송 여부를 결정하는 기능이 추가 되도록 한다.

그리고 기존 RMI와 수정된 RMI를 이용하여 여러 가지 상황에 대해서 각각 메소드 호출하면서 이 때 전송되는 TCP 세그먼트의 Sequence 번호를 서로 비교하여 수정된 RMI가 기존의 RMI보다 더 많은 데이터를 전송할 수 있음을 보인다.

## 2. 관련연구

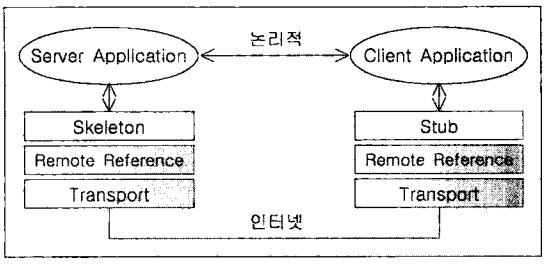
무선 상의 패킷 손실에 대해서 성능 저하를 극복하기 위하여 다양한 문제 해결 시도가 있었다. 링크 계층에서는 미디어 자체에서 패킷 전송 정보를 유지하면서 패킷이 손실되었을 경우 자체적으로 재전송을 실시하는 방법이 제시되었다[4]. 하지만 이것은 전송 계층과 중복된 일처리를 하게 되는 단점이 있다. 그리고 IP를수정하여 핸드오프 진행 사항을 TCP에전달할 수 있도록 하는 시도가 있었다[5]. 이것은 핸드오프가 완료되었을 경우 TCP에 중복된 응답을 보냄으로써 빠른 재전송을 실시하도록 하는 방법이다. 하지만 기존의 IP와 TCP를 수정해야 하는 문제점이 있으며 핸드오프 이외의 다른 단절 상황에는 적응 할 수가 없다.

위의 두 가지 접근 방법은 이미 손실된 패킷에 대해서 처리하는 방법을 제시한 것이다. 링크가 단절된 상황일 경우에는 패킷 전송 자체를 하지 않아야 하며 이로써 단절된 상황에서의 잘못된 TCP의 혼잡 제어 정책의 수행을 막을 수 있다.

## 3. Java RMI

자바에서 RMI는 원격 객체와 통신할 수 있는 분산된 응용 컴포넌트 구축을 가능하게 한다[1]. 특히 한 노드에서 실행중인 클라이언트는 객체의 메소드 호출에 의해 원격 서비스에 접근할 수 있다. 그래서 분산된 객체 기술을 이용하는 RMI 구조는 소켓과 같은 낮은 수준의 메시지 전달 방식보다 용이하게 응용 프로그램의 구현이 가능하다[8].

RMI의 계층 구조는 그림 1과 같다[6].




(그림 1) RMI 계층 모델

원격 호출을 위한 모든 객체는 반드시 Remote 인터페이스를 구현해야 한다. rmic과 같은 도구는 주어진 원격 객체에 대한 스켈레톤과 스텁을 서버의 디스크에 생성시키기 위해 사용된다. Naming 서비스를 통해 클라이언트에 옮겨진다. 스텁과 스켈레톤은 구현하고자 하는 원격 객체와 같은 메소드의 집합을 가지고 있어 원격 객체의 대리인 역할을 한다.

원격 객체에 대해서 호출을 하는 클라이언트는 실제로 Stub 객체를 호출한다. 스텁에서는 Remote Reference 계층에게 추상화된 마샬 스트림(marshal stream)을 통해 데이터를 보내는 것을 담당한다. 이 계층은 네트워크나 데이터 전송과는 상관이 없다. Remote Reference 계층은 원격 메소드 호출의 방법과 전송 방법 등을 결정한다. 전송 계층은 연결을 성립시키고 연결 관리 및 원격 객체 전달을 한다. 이 계층은 서로 다른 공간 사이에서 마샬링 된 스트림을 흘려 보내주는 하위 수준의 계층이다[9].

클라이언트에서 메소드 호출을 하면 스텁 계층은 다음과 같은 순서로 수행된다[8].

첫째, RemoteCall 객체가 Remote Reference 계층으로부터 획득된다. 이 객체는 원격 객체 끝단에서 요청을 처리하기 위한 식별자가 포함되어 있다.

둘째, 원격 메소드의 인자가 RemoteCall 객체 안으로 마샬링 된다.

셋째, RemoteCall이 호출된다. 원격 객체 끝단에 있는 적절한 메소드가 수행되고 그 결과가 RemoteCall 객체 안으로 마샬링 된다.

넷째, 원격 메소드 호출의 결과가 RemoteCall 객체로부터 언마샬링 된다.

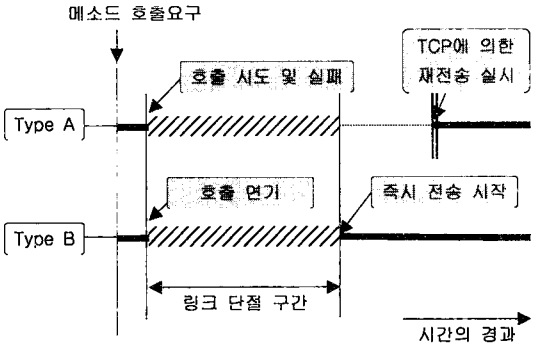
만일 서버 측에서 메소드 호출이 이루어지는 동안 예외 상황이 발생하면, 결과 대신 예외 객체가 마샬링 되어 클라이언트에게 되돌려진다.

서버의 가상 메모리에서 동작하고 있는 분산 가비지 수집기는 원격 객체에 대한 클라이언트 참조를 추적하면서 사용이 모두 종료되면 메모리 상에서 삭제시킨다.

## 4. 링크 상태 정보를 통한 데이터 전송 연기

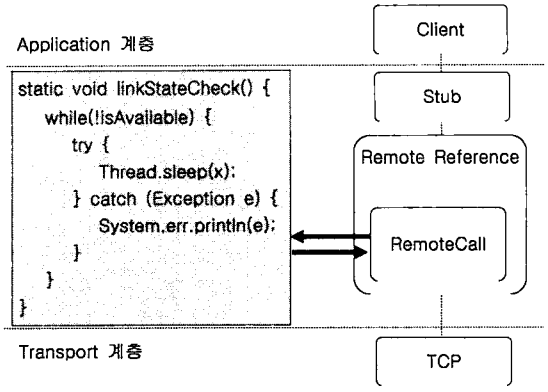
RMI의 Remote Reference 계층은 원격 메소드 호출에 대한 방법을 정의하기 때문에 대안적인 방법을 구현하기에 적합하다. 이 논문에서는 Remote Reference 계층을 수정하여 링크가 단절된 상황에서 패킷 전송을 연기시키는 기능을 RMI에 추가한다. 수정될 RMI가 전송 여부를 결정하는 근거로써 사용할 링크 상태 정보는 [7]에서와 같은 검사기를 활용하여 얻을 수 있을 것이다.

그림은 링크 단절시 일반적인 RMI(Type A)와 링크 상태 측정 결과를 통해 데이터 전송 지연을 시키는 수정된 RMI(Type B)가 동작하는 과정을 나타낸다. Type A가 실제 전송 가능한 상태가 되었음에도 TCP 특성상 재전송을 하지 못하는데 비해 Type B의 경우 전송 가능한 시점에서 바로 전송을 시작할 수가 있다.




(그림 2) 링크 단절시 일반적인 RMI와 수정된 RMI의 전송 시작

다음 그림은 클라이언트가 메소드 호출을 할 경우 Remote Reference 계층의 RemoteCall에서 링크 상태 정보를 확인하는 과정을 나타내 주고 있다. 스텁은 첫 번째로 Remote Reference 계층에 있는 RemoteCall을 얻게 되고 메소드 호출의 인자를 마샬링 하여 RemoteCall에 넣어준다. 수정된 RMI에서는 Remote Reference 계층에서 전송 계층으로 데이터를 전달하기 전에 링크 상태 측정 결과를 이용하여 데이터 전송 지연 여부를 결정할 수 있게 한다.




(그림 3) 수정된 RMI에서 링크 상태 정보 측정

## 5. 성능 비교 확인

기존 RMI와 수정된 RMI를 이용하여 여러 가지 데이터 크기와 단절 시간을 다양하게 바꾸어 가며 각각 메소드 호출하면서 이 때 전송되는TCP 세그먼트의 Sequence 번호를 서로 비교하여 수정된 RMI가 기존의 RMI보다 더 많은 데이터를 전송할 수 있음을 확인한다.

실험에 사용된 시스템은 다음과 같다. 클라이언트는 펜티엄 III-866 시스템에 리눅스 6.2를 설치했고 수정된 소스를 컴파일한 JDK 1.3.0 버전을 사용했다. 서버는 울트라 스팍 60 시스템에 솔라리스 2.7을 설치했고 마찬가지로 수정된 소스를 컴파일한 JDK 1.3.0을 사용하였다.

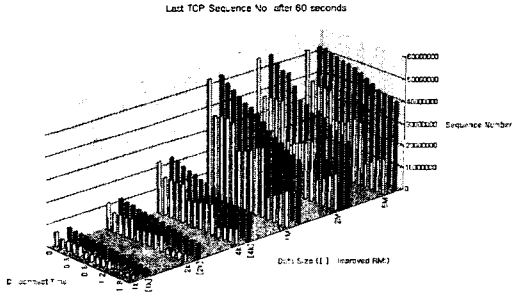
테스트에 사용된 메소드는 클라이언트에서 일정한 크기의

자료를 서버에 전송하고 서버로부터 처리 결과를 반환 받는다. 표 1은 이러한 메소드를 60초간 지속적으로 호출할 때 클라이언트에서 서버로 향한 패킷을 분석한 결과이다. 분석에는 lookup 과정에 소요된 패킷은 제외하고 순수메소드 호출에 사용된 패킷만을 사용하였다. 여기서 2초 동안 데이터를 전송한 뒤 링크를 단절시키며 단절 시간은 다음 표와 같다. 이러한 연결-단절 상황은 테스트 종료 시까지 반복된다.

| 번호 | 단절시간 | RMI | 패킷 수 | 호출 수 | TCP Sequence 증가 |
|---|---|---|---|---|---|
| 1 | 없음 | 기존 | 90942 | 5677 | 23,564,190 |
| 2 | 0.1초 | 기존 | 63773 | 3985 | 16,542,393 |
| 3 | | 수정 | 85383 | 5321 | 22,086,817 |
| 4 | 0.5초 | 기존 | 41863 | 2615 | 10,856,895 |
| 5 | | 수정 | 71892 | 4479 | 18,592,512 |
| 6 | 1.4초 | 기존 | 15655 | 975 | 4,050,889 |
| 7 | | 수정 | 53004 | 3300 | 13,699,658 |

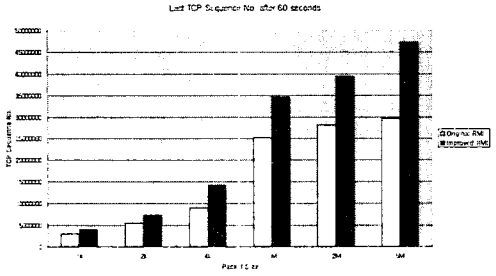
[표 1] 기존과 수정된 RMI를 이용한 메소드 호출 결과

그림 4는 기존의 RMI(흰색)와 수정된 RMI(검은색)를 이용하여 1KB, 2KB, 4KB, 1MB, 2MB, 5MB 크기의 자료를 서버로 전송하는 메소드를 실행시킨다. 이때 링크가 2초간 연결된 후 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0초간 단절 상황을 주고 이러한 것을 반복해서 실행했을 때 60초 후 최종 TCP Sequence Number를 나타낸 것이다. 단절 시간이 길어질수록 성능 향상 폭은 커졌다.



(그림 4) 데이터크기와 단절시간에 따른 Sequence Number 변화

그림 5는 임의의 단절시간에 대한 반응을 살펴 본 것이다. 위에서와 동일한 크기의 자료를 서버로 전송하는 메소드를 실행시키면서 링크가 2초간 연결된 후 0~2000ms 사이의 임의의 단절이 발생하고 이러한 상황을 반복시켰을 때 60초 후 최종 TCP Sequence Number를 나타낸 것이다. 수정된 RMI(검은색)가 일반 RMI(흰색)보다 평균 43.55%의 성능 향상을 보였다.



(그림 5) 2초 전송 후 임의 시간동안 링크 단절이 있을 경우

## 6. 결론 및 향후 연구 방향

무선 환경에서 피할 수 없는 문제는 단시간 링크 단절이다. 이것은 잦은 핸드오프와 무선 자체의 물리적인 오류로 인하여 발생하며 이러한 상황을 TCP는 유선상의 패킷 폭주에 의한 손실로 보고 혼잡 제어 정책을 수행한다. 이로 인하여 링크 상태가 회복되더라도 곧바로 전송을 할 수 없게 된다. 링크 상태가 패킷을 전달할 수 없는 상황에서는 다시 가능해질 때까지 패킷 전송을 지연시킴으로써 패킷이 무의미하게 손실되는 것과 TCP의 혼잡제어 정책 수행으로 인한 성능 저하를 막을 수 있다. Java RMI는 소켓과 같은 낮은 수준의 메시지 전달보다 분산된 객체 기술을 이용하는 응용프로그램의 구현이 가능하며 지작이 용이하다. 이 논문에서는 RMI를 수정했고 이것은 다음과 같은 장점을 갖는다.

첫 번째로 기존 RMI 코드를 이용하여 일부분만을 확장하기 때문에 쉽게 적용할 수 있고, 상대방의 RMI 및 다른 계층을 수정하지 않기 때문에 투명하게 적응할 수 있다.

두 번째로 링크 단절 상황에 적절하게 대응하므로 무선 호스트 및 고정 호스트에서 불필요한 전송을 줄일 수 있고, TCP의 혼잡 제어 정책에 의존하지 않기 때문에 링크 단절 종료시 바로 전송을 재기할 수 있다.

핸드오프나 비트 오류가 없는 안정적인 곳에서는 링크 상태 확인에 걸리는 시간으로 인해 수정된 RMI가 오히려 낮은 성능을 보일 수 있다. 향후 연구에서는 링크 상태 확인에 허용될 수 있는 시간을 정량적으로 밝힐 필요가 있다.

## 7. 참고문헌

[1] Vijaykumar Krishnaswamy, Dan Walther, Sumeer Bhola, Ethendranath Bommaiah, George Riley, Brad Topol, and Mustaque Ahamad, "Efficient implementations of Java Remote Method Invocation (RMI)," In Proc. of the 4th USENIX Conference on ObjectOriented Technologies and Systems (COOTS'98), 1998.

[2] Ajay V. Bakre and B.R. Badrinath, "Reworking the RPC paradigm for mobile clients, " Mobile Networks and Applications vol. 1, pages 371-385, 1996.

[3] B. R. Badrinath and Pradeep Sudame "To send or not to send: Implementing deferred transmission in mobile hosts", ICDCS, Hong Kong, May 1996.

[4] S. Paul, E. Ayanoglu, T. F. LaPorta, K. H. Chen, K. K. Sabnani, and R. D. Gitlin, "An asymmetric link-layer protocol for digital cellular communications," In Proc. of InfoComm '95, 1995.

[5] A. Caceres, "The Effect of Mobility on Reliable Transport Protocols," In Proc. of the 14th Intl. Conf. on Distributed Computing Systems, pages 12-20, June 1994.

[6] http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html, "Java RMI technology Specification".

[8] Girish Welling and Maximilian Ott, "Structuring Remote Object Systems for Mobile Hosts with Intermittent Connectivity," NEC USA, Inc., In Proc. of the The 18th International Conference on Distributed Computing Systems.

[7] 양진기, 강대욱, "무선 링크에 적응하는 자바 응용프로그램을 위한 리소스 모니터 설계 및 구현", 전남대 석사 졸업 논문, 2002.

[9] Qusay H. Manmoud, "Distributed Programming with Java, " Manning, 2000.