석사학위논문

이중 비실시간 연결구조에서의 실시간 RPC 성능 향상 기법

제주대학교 대학원
전산통계학과

강 미 경

2000년 12월

이중 비실시간 연결구조에서의 실시간 RPC 성능 향상 기법

지도교수 이 정 훈

강 미 경

이 논문을 이학석사 학위논문으로 제출함

2000년 12월

강 미 경의 이학 석사학위 논문을 인준함

심사위원장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

제주대학교 대학원

2000년 12월

A performance enhancement scheme for real-time RPC on the dual non-real-time connections

Mi-Kyung Kang

(Supervised by Professor Jung Hoon Lee)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND STATISTICS
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

DECEMBER 2000

목 차

List of Tables

List of Figures

Abstract

This thesis designs and analyzes a message scheduling scheme for the real-time RPC over dual non-real-time networks. The targeted dual network models are as follows: First, isolated two distinct networks such as Ethernet where each node is physically connected via both of them. Second, ATM LAN networks where dual group connections can be established with differently allocated bandwidth via Q.2971 signaling protocol between the participating nodes. Third, Internet environment where there may be multiple paths or connections between the participants and the load of each connection may be maintained differently based on the controlled load service. The proposed scheme makes the load of two networks different and gives more chance to the urgent message by transmitting it via lower load network. In addition, the proposed scheme distributes each message to one of the two network according message type, namely request or reply, aiming at meeting more time constraints of real-time RPC's by giving higher priority to reply messages, hence, minimizing the useless computation time. The chop value which ranges from 0 to 0.5 determines the network loads of two networks after quantifying the slackness. Though the choice of chop value and partition factor is critical to the real-time performance, it is hard to decide the optimal value because it depends on many other network parameters. We use the statistical analysis model for the chop value selection. The simulation results performed via SMPL on the parameters such as interarrival time, server execution time, slackness, and chop value show that the proposed scheme enhances the deadline meet ratio of real-time RPC transactions. The simulation results are compared with another measurement for chop values selected statistically and that the difference is within 0.3 %. That is, effective chop value can be estimated. As a result, the proposed scheme is able to enhance the performance of real-time RPC primitives, accordingly, the RPC-based application such as remote education can be built upon it.

Ⅰ. 서 론

인터넷 사용자와 서비스 제공자의 급증에 따라 다양한 서비스들이 출현하게 되었으며 사용자와 서비스들간의 상호작용(interoperability) 형태 또한 다양화하고 있다. 이러한 경향에 있어서 두드러진 특징 중의 하나는 멀티미디어(multimedia)나 트랜잭션 시스템(transaction system) 등과 같은 실시간 응용(real-time application)의 출현이라 할 수 있으며 실시간 서비스의 필요성이 증가함에 따라 네트워크 상에 분산된 응용들간에 실시간 상호작용 혹은 메시지 교환을 효율적으로 지원하는 방식의 필요성이 대두되고 있다(Arvind 등 1991). 실시간 응용이 교환하는 메시지는 주어진 종료시한(deadline) 이내에 전송이 완료되어야 한다는 시간 제약조건(time constraint)을 갖는데 이를 만족시키기 위해서는 응용의 시간제약 요구나 서비스의 질 요구(QoS: Quality of Service)에 따라 응용의 수행에 필요한 자원을 할당하여야 한다. 자원의 사전 할당은 일정 수준의 종료시한 만족도(deadline meet ratio)를 보장할 수 있도록 하며 보장 과정은 네트워크나 운영체제 등 하부 플랫폼의 특성에 따라 최선 노력(best-effort) 수준에서부터 결정적인(deterministic) 수준까지 제공될 수 있다. 현재 인터넷의 하부구조는 결정적인 수준보다는 최선 노력 수준으로 실시간 응용을 지원할 수 있으며 부하제어 서비스(controlled load service)는 자원할당 방식의 한 예로서 실시간 연결에 대해 용량제어(capacity control) 기능을 제공하여 경로상의 요소들의 부하를 적정하게 유지할 뿐 아니라 자원을 사전에 할당함으로써 일부 요소가 과부하(overload) 상태가 될지라도 연결 설정 과정에서 승인된 서비스를 제공할 수 있도록 한다(Wroclawski, 1997).

분산 실시간 시스템(distributed real-time system)에서 프로세스들은 서로 다른 노드에 위치하여 시스템이 제공하는 특정한 상호작용에 의해 서로 협력하여 주어진 작업을 수행하게 되는데 이러한 상호작용은 메시지의 교환을 수반한다. 각 프로세스는 메시지 교환에 있어서 원격 프로시저 호출(RPC: Remote Procedure Call)이나 분산 객체 연산(distributed object operation) 등 다양한 통신 프리미티브를 이용할 수 있는데 RPC는 분산 시스템에서 서비스 요청과 수행에 있어서 가장 기본적인 통신 프리미티브로서 많은 분산 시스템에서 사용되고 있고 대부분의 RPC 구현은 지역 환경에서의 프로시저 호출과 동일한 시맨틱을 제공하도록 설계되었다(Birrel 등 1984). RPC와 같은 통신 프리미티브들의 시간 제약조건을 만족시키기 위해서는 우선적으로 시간 제약조건을 표현

할 수 있어야 하며 시스템은 이에 의해 프리미티브들이 생성하는 메시지들을 응용의 요구에 맞추어 스케쥴하여야 한다. RPC를 사용한 호출과 결과의 전달은 클라이언트와 서버 사이의 메시지에 의해 교환되며 다른 실시간 서비스와 같이 실시간 RPC의 정확성은 결과를 산출해내는 시간 뿐만 아니라 결과의 정확성에 의존한다(Sinha, 1992). 바꾸어 말하면 결과가 호출자에게 RPC의 종료시한 이내에 전송되어 돌아왔을 때에만 의미있게 된다. 경성(hard) 분산 실시간 시스템의 경우 메시지의 손실이 전혀 있어서는 안되는 반면 연성(soft) 분산 실시간 시스템의 경우에는 일부 메시지의 손실이나 종료시한 초과를 허용한다. 결과적으로 연성 분산 실시간 시스템에 대해서 메시지의 시간 제약조건을 고려한 효율적인 네트워크 스케쥴링 기능을 제공하는 것이 필수적이다.

실시간 시스템에서 메시지를 스케쥴링하는 것은 하부 네트워크 구조에 매우 밀접한 관련이 있다. 실시간 하드웨어는 시스템 종료시한을 만족시키기는 쉽지만 매우 고가이며 손쉽게 이용할 수 없다. 반면 이더넷(Ethernet)이나 토큰 링(token ring)과 같은 표준 네트워크와 드라이버들은 실시간 통신을 지원하지 못하고 메시지를 단순하게 FCFS(First Come First Service) 순으로 전송하는 경향이 있으므로 종료시한이 촉박한 메시지를 적시에 제대로 스케쥴하지 못한다. 그러나 여분의 네트워크가 사용가능하다면 실시간 네트워크의 지원없이도 비실시간 네트워크에서 실행중인 모든 연성 실시간 응용의 실시간 성능을 향상시키는 것이 가능하다.

본 논문에서는 이중의 연성 실시간 시스템을 구축함에 있어서 실시간 RPC에 대한 메시지 스케쥴링 기법을 제안하고 객체들간의 실시간 상호작용을 지원하기 위해 실시간 RPC의 종료시한 만족도를 개선하는 기법을 제안하고 성능을 평가한다. 인터넷과 같은 최선 노력 수준으로 실시간 응용을 지원하는 하부구조에서 RPC 메시지들의 종료시한 만족도를 개선하기 위해서는 메시지의 종료시한을 고려한 스케쥴링이 필요하여 종료시한이 촉박한 RPC 메시지는 종료시한에 여유가 있는 메시지보다 신속히 전송이 완료되어야 한다(Dao 등 1991). 그러나 하부 네트워크가 단순히 FCFS 순으로 전송한다면 이와 같은 우선순위(priority)를 부여할 수 없다. 그런데 네트워크의 부하가 낮을수록 메시지의 전송 시간은 단축되므로 이중화된 네트워크 연결을 통해 두 연결의 부하나 대역폭을 다르게 유지하고 낮은 부하의 연결을 통해 여유시간이 촉박한 시간 제약조건을 갖는 RPC 메시지를 전송한다면 종료시한에 따른 우선순위를 부여할 수 있게 되어 종료시한 만족도를 개선할 수 있다.

본 논문의 구성은 다음과 같다. Ⅰ장에서 문제에 대한 도입을 한 후 Ⅱ장에서 전통적

인 RPC에 대해 분석한다. Ⅲ장에서는 실시간 RPC와 실시간 CORBA 등의 관련연구를 소개하고 Ⅳ장에서 실시간 RPC 성능 향상 기법을 제안한다. Ⅴ장에서 각 제안된 기법에 따른 성능 평가를 한 후 마지막으로 Ⅵ장에서 논문을 요약하고 결론을 도출한 후 추후 연구과제에 대해 기술한다.

Ⅱ. RPC(Remote Procedure Call)

1. 전통적 RPC

분산 프로그램은 네트워크로 연결된 여러 컴퓨터에서 실행되는 소프트웨어 요소 집합으로 볼 수 있으며 클라이언트-서버 통신 방식에 기반하여 클라이언트는 서버에게로 요청 메시지를 전송함으로써 서버측의 오퍼레이션을 호출하고 서버는 요청된 오퍼레이션을 수행한 후 클라이언트에게 다시 응답 메시지를 전송하는 호출 특성을 갖는다 (Coulouris 등 1994).

본 절에서는 고수준 언어(high-level language)를 기반으로 전통적인 프로시저 호출 방식과 유사한 RPC에 대한 전반적인 소개와 설계·구현방법에 대한 설명과 함께 RPC 시스템의 실례를 들어 RPC의 디자인과 사용으로 얻을 수 있는 이점들을 살펴본다.

1) RPC의 정의

RPC는 한 프로그램이 네트워크 상의 다른 컴퓨터에 위치하고 있는 프로그램에 서비스를 요청하는데 사용되는 프로토콜로서 서비스를 요청하는 프로그램은 네트워크에 대한 상세 내용을 알 필요가 없도록 투명성(transparency)을 지원한다. 서브루틴과 서브루틴 호출자가 다른 호스트에 존재하지만 호출자는 자신의 서브루틴을 호출하는 것처럼 RPC를 호출한다. RPC는 클라이언트-서버 모델을 사용하는데 서비스를 요청하는 프로그램이 클라이언트이고 서비스를 제공하는 프로그램이 서버이다. 지역 프로시저 호출(local procedure call)과 마찬가지로 RPC도 요청하는 프로그램이 원격 프로시저 처리 결과가 반환될 때까지 대기하여야 하는 동기(synchronous) 운영 방식이다. 그러나 같은 주소공간을 공유하는 스레드의 경우는 여러 개의 RPC들을 동시에 수행될 수 있도록 허용한다.

RPC는 네트워크를 통한 프로시저 호출 기능으로 분산시스템 개발에서 필수적인 요소인데 RPC의 효과는 부하 균배(load balancing)와 서버의 고유 기능 실행에 있다. RPC의 기본 목적은 네트워크 프로그래밍을 전통적인 함수 호출과 같이 사용할 수 있

도록 하는 것이다.

함수 호출은 같은 시스템에서 호출되어 실행되고 값의 전달(pass-by-value), 참조의 전달(pass-by-reference), 이름의 전달(pass-by-name) 등으로 인자(argument)를 전달하며 데이터의 표현 방식은 호출하는 쪽과 호출되는 쪽 모두 동일하고 함수의 주소는 코드가 있는 메모리의 위치를 나타낸다. 반면에 RPC는 네트워크 상에서 호출되어 실행되고 값 전달 방식으로 인자가 전달되며 데이터의 표현은 기계의 종류에 따라 다를 수 있다. 그리고 기계주소 및 함수이름이 함수의 주소가 되며 불순한 의도를 가진 접근을 제어할 수 있는 보안능력을 가진다.

<Figure 1>에서 보는 바와 같이 지역 프로시저 호출은 클라이언트와 서버가 동일한 시스템에서 호출되므로 직접 인자와 결과값이 전달되는 반면 RPC는 클라이언트 프로시저의 인자가 클라이언트 스터브(stub)로 전달되어 요청 메시지가 네트워크를 통해 서버의 스터브로 전송되고 서버의 결과 수행 이후에도 응답 메시지가 네트워크를 통해 클라이언트 스터브로 전송된다.

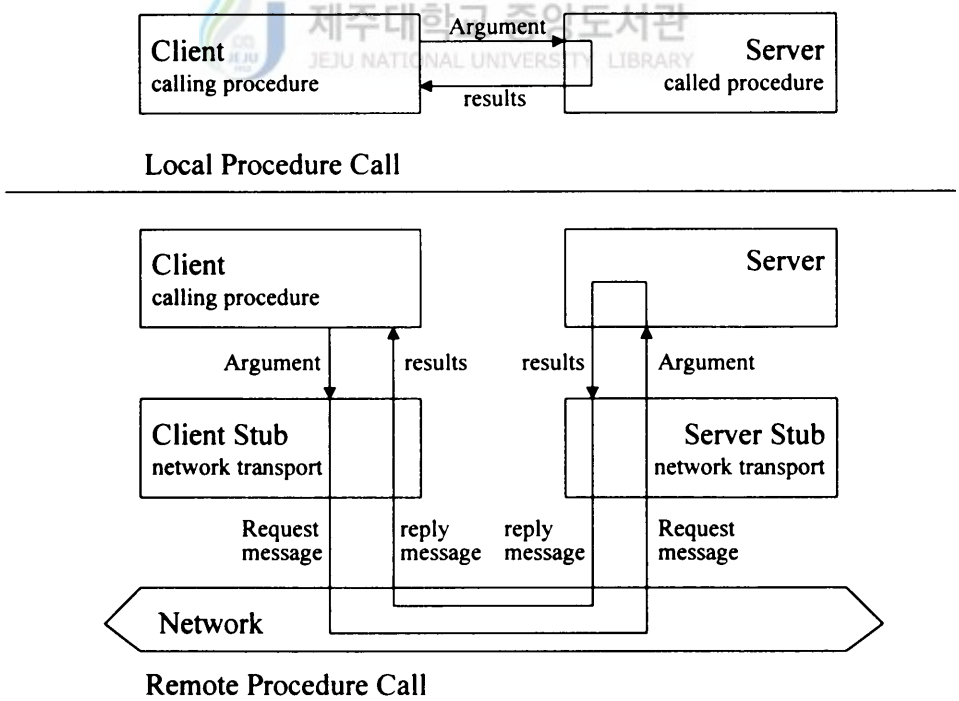

Figure 1. Local procedure call vs. remote procedure call

지역 호스트에서 실행되는 프로시저가 원격 호스트에서 실행되는 프로시저를 호출할 수 있도록 해주는 RPC 기법에서는 프로시저의 인자와 결과값은 투명하게 네트워크를 통해 전송된다. 분산 동작의 실제 과정이 지역 호출과는 실제로 많이 다르기 때문에 완전한 투명성을 지원하는 것은 불가능하다. 예를 들면 피호출 프로시저가 호출 프로시저와 다른 호스트에서 수행되는 경우에는 두 프로시저가 동일한 호스트에서 수행될 때에는 발생하지 않는 여러 가지 문제가 발생할 수 있다. 두 소프트웨어 요소 각각이 독립적으로 고장일 수도 있고 두 요소가 실행될 때 그들간의 연결성이 잠시 상실될 수 있으며 지역 환경에서는 발생하지 않는 관리와 보안에 관련된 문제가 발생할 수도 있다 (Coulouris 등 1994). 그렇지만 대부분의 RPC 매커니즘은 가능한 한 많이 기존의 프로그래밍 기법을 유지하고 분산 환경 특유의 복잡성을 처리하는 장점을 갖고 있다.

RPC 기법은 일반적으로 메시지 전달 매커니즘의 위에 구축되며 인수정리(marshaling) 프로세스를 이용하여 호출 프로시저와 피호출 프로시저간에 메시지가 교환되는데 있어서 네트워크 메시지의 전달 및 포맷팅을 수행한다. 클라이언트-서버 환경에 RPC 기법을 사용하는 주된 이유는 분산 기능에 포함된 많은 작업을 자동으로 처리할 수 있고 응용 개발자가 클라이언트-서버 응용을 설계, 구현, 유지하는 데 용이하기 때문이다.

2) 설계 논점

RPC를 설계하는 데 있어서 중점적으로 다루어야 할 문제를 몇 가지로 요약하면 다음과 같다.

첫 번째, 두 개의 RPC 시스템 클래스 중 하나는 RPC 서버를 구축하고 다른 하나는 RPC 클라이언트를 구축한다. 이 클래스들의 주요 용도는 응용개발자들이 ONC(Open Network Computing)나 UNIX 시스템 V에서 RPC API(Application Program Interface)들의 세부 사항을 알 필요가 없도록 상위 수준의 RPC 인터페이스들을 제공하는 것이다.

두 번째, RPC 기법은 그 인터페이스를 기술할 수 있는 특수한 프로그래밍 언어로 조합되어야 한다. 그리고 클라이언트와 서버 사이의 인터페이스를 기술하기 위해 특수 목적의 인터페이스 정의 언어(IDL: Interface Definition Language)를 사용해야 한다.

RPC 인터페이스를 정의할 때는 프로시저의 이름과 인자 타입을 기술해야 하고 어떤 인자가 입출력으로 사용되는지 기술해야만 한다.

세 번째, RPC는 지정한 인자가 부적절하거나 기타 네트워크 장애 등 예외상황(exception)이 발생했을 경우 이를 처리할 수 있는 기법을 가져야 한다. 예외처리 기법은 예외상황이 발생했음을 알리는 것(notification)과 발생한 예외상황을 처리할 수 있는 두 개의 부분으로 이루어진다. 최근의 프로그래밍 언어들은 예외상황을 처리해 줄 수 있는 기능을 제공하고 있다.

네 번째, 전송을 보장하기 위한 방법으로는 요청 메시지를 재전송하는 방법(retry request message)과 복제 필터링 방법(duplicate filtering), 그리고 서버의 오퍼레이션을 재실행하지 않고 손실된 응답 메시지를 재전송 할 수 있도록 응답 메시지의 히스토리를 갖는 선택(retransmission of replies)이 있을 수 있다. 이러한 전송 보장 방법의 조합으로 <Table 1>에서와 같이 호출자에 의한 원격 프로시저의 신뢰성을 제공하는 다양한 구문을 가능하게 한다. 즉 maybe call semantic은 호출 후 응답 수신 없이 계속 진행하여 요청 메시지도 재전송하지 않고 중복된 요청인지 확인한다거나 재전송할 필요가 없다. at-least once semantic은 서버는 불리는 대로 수행하되 중복된 요청에 대해서도 확인 없이 재수행하고 클라이언트는 응답이 안 오면 재요청 하는 방식으로 idempotent 연산에 적용된다. 그리고 at-most once semantic은 재시도에 의한 중복연산을 피하는 방식으로 가장 많이 쓰이며 이 경우 서버 고장 시에만 수행이 불가능하다 (Coulouris 등 1994).

Table 1. RPC call semantics

Delivery guarantees			RPC call semantics
Retry request message	Duplicate filtering	Re-execute procedure or retransmit reply	
No	Not applicable	Not applicable	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

다섯 번째, 투명성이다. 데이터 매개변수를 네트워크에 투명한 형식으로 인자를 정리함으로써 양쪽 기계에서 정보를 정확하게 변환할 수 있도록 한다. 그래서 지역 호출과

원격 호출의 차이를 호출자가 알지 못하도록 하여 호출자는 RPC 수행시의 재시도 여부 혹은 오류 발생 여부를 알지 못하도록 한다.

3) 구현

RPC의 동작 과정은 크게 다음과 같이 볼 수 있다.

RPC를 사용하는 프로그램 문장들이 실행 프로그램으로 컴파일될 때 컴파일된 코드 내에 RPC의 대리인(agent)처럼 동작하는 스터브가 포함된다. 스터브는 원격지에 위치해 있는 큰 프로그램을 대리하기 위한 작은 프로그램 루틴이다. 그 프로그램이 실행되어 프로시저 호출이 이루어질 때 스터브는 그 요구를 받아서 그것을 동일한 컴퓨터 내에 위치한 클라이언트 런타임(run-time) 프로그램에게 전달한다. 클라이언트 런타임 프로그램은 원격 컴퓨터와 서버 프로그램과 어떻게 접촉해야 하는지에 대한 지식을 가지고 있으므로 네트워크를 통해서 원격절차 요구 메시지를 보낸다. 이와 유사하게 서버는 런타임 프로그램과 원격절차 그 자신과 인터페이스 작동을 하는 스터브를 포함한다. 처리 결과들은 같은 방식으로 되돌려진다.

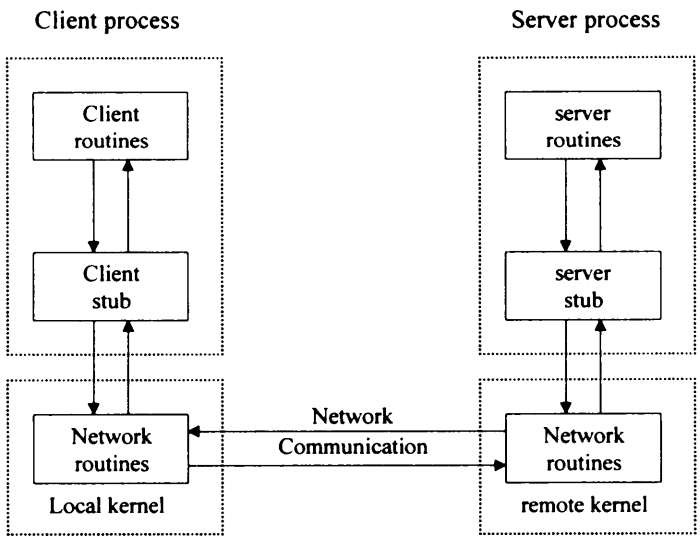

Figure 2. Client process vs. server process

<Figure 2>는 클라이언트 프로세스와 서버 프로세스가 RPC을 수행하기 위해서 클라이언트에서 지역적인 네트워크 루틴을 수행한 후 네트워크 통신을 수행하고 원격지 커널에서도 네트워크 루틴을 통하여 호출이 이루어지며 서버에서 클라이언트로의 전송 방법도 동일하다는 것을 보여준다.

클라이언트 스터브는 네트워크 상의 호출을 지역 호출로 보이게 하고 서버 스터브는 클라이언트로부터의 요청을 기다리며 스터브는 요구를 받아들여서 그것을 원격 절차에게 전달한다. 원격지에 있는 프로시저가 서비스 수행을 끝내면 결과가 스터브에게 반환되며 스터브는 그것을 서비스를 요구했던 프로그램에게 전달한다. 이러한 RPC를 구현하기 위해서는 다음 세 가지 작업이 이루어져야 하는데 그 첫 번째가 인터페이스 처리이다. 상용 프로그래밍 언어로 작성된 RPC 기법은 서버와 클라이언트에 있는 인자들을 정리할 수 있어야 하며 요청 메시지를 서버에 있는 프로시저에게 신속히 전달해 줄 수 있어야 한다.

클라이언트 측의 프로그램은 다음과 같이 수행된다. 원격 프로시저가 클라이언트에 의해 호출될 때 스터브 프로시저가 생성되며 지역 프로시저를 원격 프로시저 호출로 서버 측에 변환해 주는 것이 주목적으로 다음과 같은 일을 한다. 인자들을 정리해서 프로시저 식별자로 묶어 전송 메시지에 싣고 전송 메시지를 서버에 보내고 응답을 기다린다. 그리고 나서 결과로 받은 메시지를 자료구조로 복원한다. 즉 스터브는 인자 및 실행 결과를 메시지 형태로 정리하는 역할을 한다. <Figure 3>은 클라이언트 프로그램, 사용자 패키지, RPC 인터페이스 하위의 소프트웨어 사이의 관계를 보여준다.

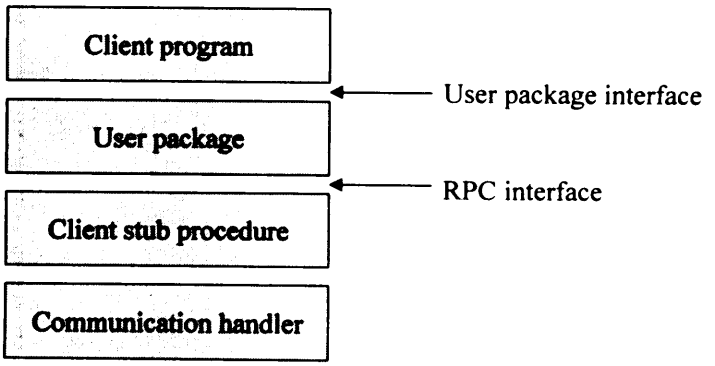

Figure 3. Levels in the client software

서버측의 프로그램을 설명하면 다음과 같다. 디스패처(dispatcher)는 요청 메시지에 있는 프로시저 식별자를 사용해서 서버 측의 스터브 프로시저를 선택해 인자를 싣는 역할을 한다(Coulouris 등 1994). 서버측의 스터브 프로시저는 클라이언트로부터 넘어온 인수를 분석하여 적당한 서비스 프로시저를 호출한 후 그 결과가 되돌아오면 그 결과값을 응답 메시지에 실어 보내는 역할을 한다. 오류가 발생할 경우는 오류코드 등을 보내 오류보고를 한다.

즉 <Figure 4>에서와 같이 클라이언트 스터브는 클라이언트 프로시저의 지역 호출을 전달받아 인자를 정리하고 요청 메시지를 서버로 전송하는데 서버측에서는 적절한 프로시저를 선택하는 디스패처 단계를 거치고 스터브가 인자를 언마샬링하면 서버가 실제 수행을 하게 된다. 계산된 결과는 서버 스터브에서 정리되어 네트워크를 통해 응답 메시지가 전송되고 클라이언트의 스터브에서 결과값을 언마샬링하여 지역 호출의 결과값을 반환하는 것처럼 호출자에게 전달한다.

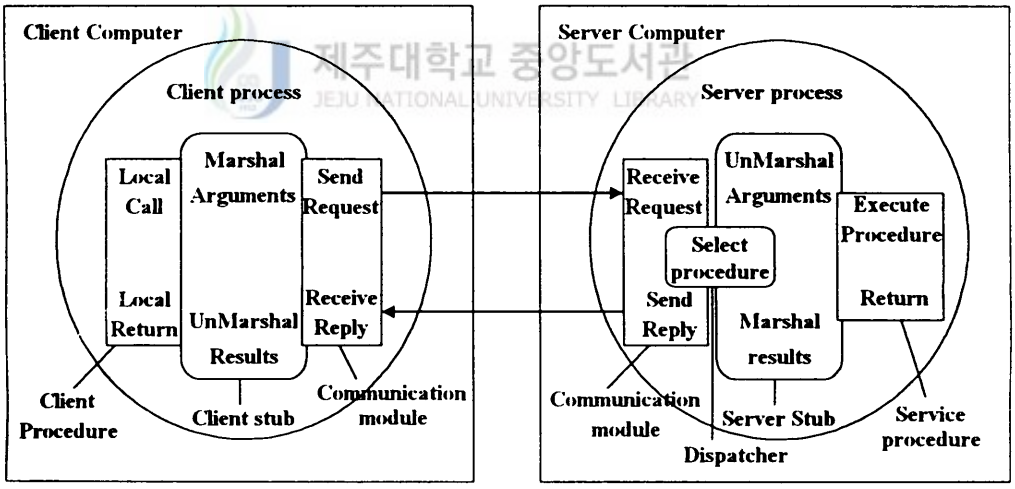

Figure 4. Stub procedures

인터페이스 컴파일러는 프로그래밍 언어로 기술된 인터페이스의 정의부분을 실행하며 호출의 종류가 보통의 프로시저 호출인지 RPC인가를 구별하여 실제 프로시저 호출로 변환하는 일을 한다. 즉 클라이언트의 스터브 프로시저를 생성하고 서버의 스터브 프로시저를 생성하고 인터페이스에 기술된 프로시저 이름과 인자를 이용하여 각각의

스터브 프로시저에 맞는 마샬링 및 언마샬링 연산을 생성해 낸다. 그리고 난 뒤 각 프로시저의 헤더부분을 생성해내고 프로그래머는 프로시저의 몸체 부분을 구성하면 된다.

<Figure 5>는 Sun RPC의 경우 서버와 클라이언트에서 작성된 프로시저가 스터브와 런타임 라이브러리와 함께 컴파일되어 실제 수행가능한 프로그램을 생성해내는 과정이다(Stevens, 1990).

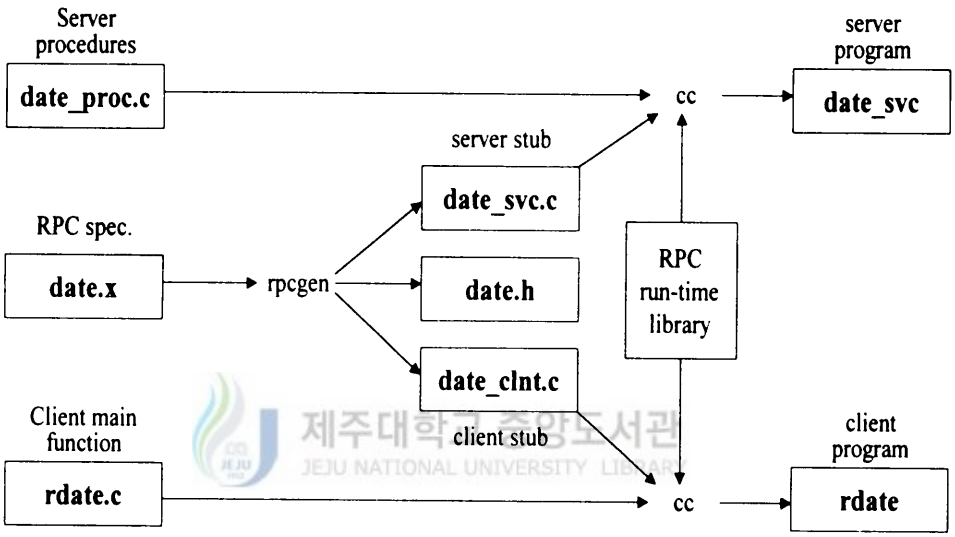

Figure 5. Sun RPC

클라이언트-서버 통신 모델은 일반적으로 UDP(User Datagram Protocol) 계층상에서 동작하여 프로토콜 시맨틱과 오류를 처리한다. 송신 노드, 수신 노드 및 게이트웨이에서 기각되거나 네트워크 분할에 의해 전송 불가 상태가 되는 경우와 프로세스가 다운되는 경우 오류가 발생하므로 재시도하거나 계수종료(timeout) 등의 기법을 필요로 하게 된다.

<Table 2>에는 Spector에 의해 구분된 세가지 RPC 프로토콜에서 클라이언트와 서버 사이에 전송되는 메시지가 요약되었다. R(request) 프로토콜, RR(request-reply) 프로토콜, RRA(request-reply-acknowledge reply) 프로토콜은 여러 가지 타입의 RPC를 구현하는데 사용된다(Coulouris 등 1994).

Table 2. RPC protocols

Name	Messages sent by		
	Client	Server	Client
R	Request		
RR	Request	Reply	
RRA	Request	Reply	Acknowledge reply

R 프로토콜은 프로시저로부터 반환되는 값이 없고 클라이언트가 프로시저 실행에 대한 확인을 필요로 하지 않을 때 사용된다. 클라이언트는 요청 메시지를 전송한 후 응답 메시지를 기다릴 필요 없이 계속 수행된다.

RR 프로토콜은 요청-응답 프로토콜에 기반을 두기 때문에 대부분의 모든 클라이언트-서버 통신에 유용하게 사용된다. 서버의 응답 메시지가 클라이언트의 요청 메시지에 대한 응답 확인으로 간주되기 때문에 특별한 응답 확인 메시지가 필요 없게 된다. 마찬가지로 클라이언트로부터의 다음 요청은 서버의 응답 메시지에 대한 확인으로 볼 수 있다. 시간 제약조건을 갖는 실시간 RPC에서는 주로 RR 시맨틱을 따른다.

RRA 프로토콜은 요청-응답-응답확인(request-reply-acknowledge reply)의 세 개의 메시지 교환에 기반을 둔다. 부가 메시지를 포함한다고 해도 응답 메시지가 클라이언트에게 도달한 후에 응답 확인을 전송하므로 블록킹될 필요가 없으며 프로세싱 및 네트워크 자원을 사용한다.

요청 메시지가 재전송되는 경우에는 서버가 한번 이상 메시지를 수신하게 된다. 예를 들어 서버가 요청 메시지를 수신하지만 명령어를 실행하고 응답 메시지를 반환하기 위한 시간이 클라이언트의 주어진 시간을 초과하게 되는 경우가 있다. 이러한 경우 서버는 같은 요청에 대해 한번 이상 연산을 수행해야 하는 결과를 초래하게 된다. 이것을 피하기 위해 같은 클라이언트로부터의 메시지를 메시지 식별자(ID)를 가지고 판단할 수 있고 중복된 요청을 추출해 낼 수 있도록 프로토콜이 설계되어야 한다. 하지만 서버가 아직 응답 메시지를 전송하지 않은 경우라면 특별히 취해야 할 일은 없으며 단지 수행중인 연산을 끝마쳤을 때에 응답메시지를 전송하기만 하면 된다.

중복된 요청을 수신했을 때 서버가 이미 응답 메시지를 전송시킨 경우라면 결과를 얻기 위해 연산을 수행할 필요가 있다. 이것은 응답 메시지의 손실이 요청 메시지의 재전송을 초래한 경우이다. 한번 이상 연산을 수행했을 때 idempotent 연산은 여러번 수

행하여도 똑같은 결과를 반복적으로 수행해 내는 연산이다. 예를 들어 덧셈 연산을 수행하는 경우 수행될 때마다 똑같은 결과값이 나오기 때문에 idempotent 연산에 속한다. 반면에 연속적으로 항목값을 증가시키는 연산은 수행될 때마다 결과값을 변화시키기 때문에 nonidempotent 연산이다.

서버가 연산을 재수행 하지 않고도 응답 메시지를 재전송할 수 있도록 히스토리가 사용될 수 있다. 히스토리는 전송되는 응답 메시지의 복사본을 보유하는 구조로 메시지 식별자를 이용하여 클라이언트가 재요청했을 때 응답 메시지를 바로 전송할 수 있도록 하는 것이다. 메모리 비용을 초래하고 속도가 늦어지며 메모리 크기 문제나 언제까지 저장해야 하는지 애매하다는 것이 문제점이다. 클라이언트가 한번에 하나의 요청만 전송하기 때문에 서버는 각 요청을 이전 응답에 대한 확인으로 해석할 수 있다. 그러므로 히스토리는 각 클라이언트에게 보내진 최후의 응답 메시지만을 보유한다. 그러나 많은 클라이언트들이 존재할 때 서버의 히스토리에 존재하게 되는 많은 응답 메시지가 문제이다. 일반적으로 클라이언트가 수신한 최후의 응답 메시지에 대한 확인을 보내지 않으므로 히스토리의 메시지는 보통 제한된 시간 이후 버려지게 된다. 그 대안으로는 RRA 프로토콜을 사용하여 클라이언트가 각 응답 메시지에 대한 수신 확인을 하게 함으로써 서버가 히스토리의 개체를 삭제할 수 있도록 하는 것이다.

지역 네트워크에서 사용되는 메시지 전달 프로토콜은 요청이나 결과의 크기가 다양한 RPC 시스템에서 사용하기에는 부적절하다. 따라서 요청이나 결과의 크기가 한 패킷을 넘는 경우 다중 패킷 요청-응답 메시지를 사용하며 메시지를 재조합하는 기능 및 선택적 재전송하는 기능을 필요로 한다. 다중 패킷 메시지는 응답확인이나 재전송하지 않고 연속적으로 요청 패킷들을 전송함으로써 구현되며 서버는 모든 패킷이 수신되었을 때야 비로소 전체 요청 메시지에 대한 응답을 확인한다. 전송이 잘못된 경우에는 클라이언트가 시간 초과후 모든 패킷을 재전송하거나 서버로 하여금 손실된 패킷을 선택하도록 하는 방법이 있다. 대안으로는 서버가 각 요청 패킷에 대한 응답확인을 하게 하여 클라이언트가 시간 초과후 그 패킷을 재전송하게 할 수도 있다. 최후의 패킷은 요청에 대한 응답으로 확인이 되는데 이러한 방법은 송신자가 수신자가 받을 수 있을 정도로 전송하는 흐름 제어 기법을 제공하는 이점을 가지며 수신측에서의 버퍼 공간 부족으로 손실되는 경우가 발생하지 않는다.

2. 연성 실시간 통신

실시간 통신 메시지들은 시간 제약 조건을 갖는다. 주어진 종료시간 이내에 전송이 완료되어야 하므로 종료시한을 초과하는 결과는 무의미하게 되며 실시간 통신에서는 어떤 MAC(Medium Access Control) 프로토콜을 사용하는지에 아주 밀접한 관계를 갖는다. 실시간 통신의 하부 네트워크는 FDDI(Fiber Distributed-Data Interface), 우선순위 셀 교환 ATM(Asynchronous Transfer Mode) 교환기 등의 고비용 실시간 네트워크와 일반적으로 많이 사용되는 이더넷이나 순수 토큰링과 같은 비실시간 네트워크로 나뉜다.

경성 실시간 통신(hard real-time application)은 실시간 통신을 통해 메시지의 손실이 전혀 있어서는 안 된다. 손실이 생기는 경우 시스템 운영에 치명적인 결과를 초래하기 때문이다. 경성 실시간 통신은 오프라인 분석과 대역폭 할당을 사전에 할당하는 스케쥴링 기법이 필요하여 FDDI, ATM 등에서 많은 연구가 행해지고 있다. 하나의 메시지라도 손실되면 파급효과가 큰 공장제어 시스템이나 자동 항공 시스템이 여기에 속한다(Arvind 등 1991).

반면 연성 실시간 통신(soft real-time application)은 메시지의 일부가 손실되거나 종료시한을 초과하는 경우를 허용한다. 가능한 한 많은 메시지의 종료시한을 만족시키기 위한 통신으로 트랜잭션 시스템이나 멀티미디어 표현장치가 이에 속하며 가능한 한 많은 메시지를 빠르게 응답하는 것을 목적으로 하는 데이터베이스 등이 포함된다.

Ⅲ. 관련연구

1. 실시간 RPC

전형적인 실시간 RPC 트랜잭션은 클라이언트로부터 서버로 호출의 전달, 서버에서의 수행, 서버로부터 클라이언트로의 결과 전달 등 순차적인 과정으로 구성된다. 클라이언트의 호출 과정은 시간 제약조건, 목적지 서버 주소 등과 아울러 수행을 원하는 프로시저에 대한 정보, 이에 관련된 입력 인자 등을 하부 네트워크를 통해 서버로 전송한다. 요청을 받으면 서버는 메시지에 명시된 입력 인자를 기반으로 요청된 프로시저를 수행하는데 동시에 여러 개의 요청이 도착한 경우 요청들은 서버 큐에 삽입되어 추후에 수행되도록 한다. 서버 큐의 대기 정책은 서버가 수행되는 운영체제에 의해 결정되며 이에 가능한 정책으로는 FCFS, EDF(Earliest Deadline First) 등이 고려될 수 있다(Ramamritham 등 1984 ; Deitel, 1994). 요청에 대한 수행이 완료되면 서버는 서비스를 요구한 클라이언트에게 계산 결과를 전송하는데 이러한 RPC 트랜잭션 과정이 모두 종료시한 이내에 완료되어야 RPC의 시간 제약조건을 만족시킬 수 있다.

1980년대에 여러 회사들에 의하여 다양한 RPC가 구현되었다. 그들 중에는 SUN 마이크로 시스템즈의 오픈 네트워크 컴퓨팅(ONC)과 Apollo 컴퓨터의 네트워크 컴퓨팅 구조(NCA)가 있었다. 오늘날 휴렛 팩커드의 HP-UX, IBM의 AIX, SUN 마이크로 시스템즈의 SUN OS 4.1.X, 그리고 산타크루즈 오퍼레이션의 SCO UNIX와 같은 대부분의 상업용 UNIX 시스템들은 모두 ONC 기법을 기반으로 RPC를 구현하였다. NFS의 일부로서 포함된 Sun사의 RPC 기능은 프로그램 구성요소들이 인터넷상에서 일반적인 프로시저 호출 기법을 이용하여 통신할 수 있도록 해준다. 사용자가 인지할 수 있는 NFS 기능중의 대다수는 Sun사의 RPC기능 위에서 수행된다

RPC는 고수준의 언어로 쓰여진 프로그램으로 네트워크를 통한 통신을 제공하는데 유용하게 사용되는 기법이다. Andrew D. Birrel과 Bruce Jay Nelson은 원격 프로시저 호출을 제공하는 패키지와 패키지 설계시의 선택사양, RPC 바인딩, RPC 기법의 전체적인 구조, 트랜스포트 계층의 통신 프로토콜 등을 연구하여 결과적으로 Cedar 프로젝트를 위한 RPC를 구현하게 되었다(Birrel 등 1984). RPC 설계시에는 호출시 정확한 시

맨틱, 공유된 주소 공간에서 주소를 포함하는 인자 시맨틱, 원격 호출과 기존 프로그래밍 시스템과의 통합, 바인딩, 호출자 사이의 적절한 데이터 전송 및 제어 프로토콜, 개방형 통신 네트워크에서 데이터의 무결성(integrity)과 보안을 제공하는 방법 등을 고려해야 한다. 통신 비용과 서버에 의해 유지되는 영역을 감소시키기 위해 트랜스포트 계층의 연결 관리 기법을 이용한 바인딩 시맨틱은 굉장히 강력할 뿐만 아니라 간단하고 구현하기에 쉽고 효율적이다.

현재 대부분의 네트워크 프로토콜은 파일 전송과 원격 로그인과 같은 전통적인 응용을 지원하는데 적합하다. 그러나 분산 응용은 프로세서 실패의 경우에도 효율적인 at-most-once 원격 프로시저 호출이 이루어져야 하는 등 또다른 요구사항을 충족시켜야 한다. 그래서 새로운 요구사항을 만족시키기 위해 상황에 따른 프로토콜을 사용하는 대신 점대점 통신과 멀티캐스트 통신을 제공하며 거의 네트워크 자원을 필요로 하지 않는 FLIP(Fast Local Internet Protocol)이라는 새로운 프로토콜을 설계하였다(Kaashoek 등 1993). 뿐만 아니라 FLIP을 사용함으로써 원격 프로시저 호출과 그룹 통신과 같은 상위 수준의 프로토콜을 간소화하고 프로세스 이동과 보안에 대한 지원을 강화시켰다. FLIP 구현은 Amoeba 분산 운영 시스템의 새로운 커널의 일부로 이루어져 일상적으로 사용되고 있다.

VMTP(Versatile Message Transaction Protocol) 프로토콜은 1986년에 RPC, 멀티캐스트, 실시간 통신을 지원하기 위해 설계되어 효율적인 페이지 계층의 네트워크 파일 접근을 지원한다(Cheriton, 1988). VMTP는 트랜스포트 계층에서 파일 전송을 구현하기 위해 그 이상의 계층을 위한 전송 지원을 한다. 연결관리는 응답 확인 전송을 하기 위해 요청-응답 프로토콜을 사용하며 선택적 Ack와 비트 매스킹으로 RPC 메시지 교환 과정에서의 오버헤드를 감소시킨다. 이와 아울러 중복되거나 손실된 패킷을 처리하기 위하여 패킷 그룹화를 기반으로 한 흐름제어 기능을 제공한다. 전송자는 한번에 하나의 패킷 그룹을 전송하고 수신자는 이 패킷 그룹을 하나의 유닛으로 수신하고 나서 응답 확인을 한다. VMTP는 주로 트랜스포트 계층에 집중되어 있는데 다소 낭비되더라도 성능의 최대 지연요소인 프로토콜 처리 시간을 최소화하기 위해 프로토콜 처리 요구량을 최소화하고 현재 네트워크의 낮은 오류율의 장점을 충분히 이용하기 위해 오류 회복에 과도한 처리를 하지 않는다. 보안, 실시간, 비동기 메시지 전송, 스트림화, 멀티캐스트와 idempotent 연산을 포함하는 VMTP의 전체 기능은 VMTP 사용자 계층에서 다양한 선택을 할 수 있게 한다.

M. Dao의 실시간 RPC의 성능 향상 기법은 서버가 요청을 수행하기 전에 클라이언트의 종료시한을 만족시킬 수 있는지 여부를 먼저 계산하게 한다(Dao 등 1991). 사전에 수행을 할 필요가 있는지 확인을 함으로써 RPC 종료시한 만족도를 증가시키지 않더라도 클라이언트가 타이밍 오류를 회복할 수 있는 시간을 늘리게 된다.

J. Lee가 제안한 실시간 RPC를 지원하는 통신 시스템 설계 기법은 실시간 RPC를 종료시한과 유형, 즉 메시지가 요청 메시지인지 응답 메시지인지에 따라 RPC 메시지를 동적으로 할당하고 서버 실행시간의 낭비를 막는다(Lee, 1996). RPC 요청 메시지 전송과 과정에서 확인 메시지 전달과 서버에서 수행시간을 예약하고 종료시한 만족이 가능한 경우만 RPC가 수행되기 때문에 네트워크나 서버의 낭비시간이 전혀 없다. 이 기법은 특수한 네트워크나 TDMA(Time Division Multiple Access)가 구현된 이더넷에 제한적이어서 일반적인 네트워크 환경에 적용이 불가능하다.

B. Gao는 이중의 비실시간 네트워크상의 메시지 여유시간에 따라 부하를 분배하는 스케쥴링 기법을 제안했다(Gao 등 1996). 이 기법은 연성 실시간 메시지의 종료시한 만족도를 높이는 데 초점을 두어 각 메시지들이 독립적인 상황에서 효율적으로 동작하기는 하지만 요청-응답 시맨틱을 고려하지 않았다. 또 비실시간 네트워크에서 적용이 불가능한 EDF 정책에 따라 각 노드들로 하여금 메시지를 스케쥴하도록 가정하였다(Ramamritham 등 1984). 이러한 메시지 스케쥴링 정책이 실시간 메시지에 다양하게 적용될 수 있다.

성능은 RPC 이용에 결정적이다. 효율적인 커널 수준의 서비스를 제공하고 클라이언트와 서버를 네트워크 통신시 상세한 부분으로부터 분리시키는 RPC 설계 대안은 3년간의 워싱턴 대학의 HCS(Heterogeneous Computer Systems) 프로젝트가 시초가 되었다(Chung 등 1989). HCS 프로젝트는 이기종 컴퓨터 시스템을 상호연결시키는데 드는 비용을 감소시키기 위한 기법을 설계하고 프로토타입(prototype)을 제작하였다. 이기종간의 전자메일, 원격 계산, 지명서비스 같은 서비스 요구는 HCS RPC에서의 유연성과 효율성 사이의 많은 트레이드오프에 초점을 두게 된다. 이 대안은 다양한 정도의 유연성을 갖는 표준 RPC 시맨틱을 수행하도록 하는 총제적인 접근법으로 컴파일된 스터브와 해석된 스터브, 패킷 버퍼를 이동하는 데이터에 대한 순차적인 코드와 직렬 코드, 블록 복사와 개별적인 데이터 항목 복사, 바이트 스와핑 존재여부에 따른 성능 관계를 살펴보았다. 결과적으로 컴파일된 스터브는 크고 불규칙한 인자들의 경우에만 해석된 스터브보다 성능이 좋지만 해석의 유연성은 시스템 구조의 이점을 가져온다. 직렬 데이

터 이동은 순차적인 데이터 이동에 비해 약간의 성능향상을 가져오지만 순차적인 호출은 기본적인 복사 연산에 비해 비용이 덜 들게 된다. 바이트 스와핑은 패킷에 데이터의 개별 항목을 복사하는 것과 별 차이가 없다. 패킷으로 데이터를 블록 전송하는 것은 개별 항목을 복사하는 것보다 훨씬 성능 향상을 시킬 수 있다.

대부분의 경우 RPC 통신은 만족스러운 수행을 하지만 때로는 문제가 발생할 수 있다. RPC가 본질적으로 주종관계를 이루고 점대점으로 연결되어 있다면 더욱 그렇다. Amoeba 4.0에서는 RPC가 긴 메시지를 전송할 때 연속된 패킷으로 전송하게 하는데 각 패킷은 정지대기 프로토콜을 이용하여 개별적으로 전송확인을 하게 된다 (Tanenbaum 등 1990). 이러한 기법이 간단하기는 하지만 시스템의 속도를 저하시키는 요인이 된다. Amoeba 5.0에서는 전체 메시지의 전송확인만을 하게 하여 고대역폭을 기반으로 수행할 수 있게 한다. RPC가 본질적으로 점대점으로 연결되어 있기 때문에 TSP(Traveling Salesman Problem)와 같은 병렬 응용에서는 문제를 일으키게 된다. 프로세스가 현재 발견된 경로보다 더 나은 경로를 발견했을 때 즉시 알리기 위하여 수많은 프로세스로 멀티캐스트 메시지를 전송해야 하는데 이것은 불가능한 일이며 다중 RPC로 모의실험 하는 등의 기법을 이용해야 한다. Amoeba 5.0은 멀티캐스트를 이용하여 완전한 그룹 통신을 지원한다. 그룹으로 전송된 메시지는 모든 멤버에게로 전송되거나 아니면 어디에도 전송되지 않는다. 신뢰할 수 없는 네트워크에 100 % 신뢰가능한 멀티캐스팅을 구현하기 위해 고수준의 프로토콜이 고안되었고 Amoeba 5.0은 결함허용을 지원하기 위해 복제기술을 이용한다. 모든 LAN이 브로드캐스팅과 멀티캐스팅을 지원하는 것은 아니지만 이더넷과 같이 이러한 능력을 가지고 있다면 많은 응용에 크나큰 성능 향상을 가져올 수 있다.

x-kernel은 인터넷을 통한 일관적인 자원 접근을 할 수 있도록 하는 개인 워크스테이션을 위한 실험적인 운영체제로 TCP/IP 인터넷과 유사한 네트워크의 상호연결이며 NREN(National Research and Education Network)로 불린다(Peterson 등 1990). 지역 자원보다는 전역적인 자원을 접근하는 측면에 초점을 두어 다른 실험적 운영체제와 달리 했다. 운영체제 설계 추세는 같은 지역 네트워크의 워크스테이션을 밀접하게 연결시켜 같은 커널과 같은 목적의 프로토콜로 실행되는 것이었다. 그러한 설계 유형에는 V 시스템과 그 시스템의 메시지 트랜잭션 프로토콜, Sprite 네트워크 운영체제와 그의 RPC 프로토콜, Amoeba 시스템과 그의 RPC 프로토콜 등이 포함된다. 그러나 네트워크로 연결된 워크스테이션 사용자들은 지역 네트워크에서 사용 가능한 자원보다 훨씬 더

많은 자원을 접근하려 했다. NERN은 전역적인 데이터베이스, 파일 시스템, 디렉토리 서비스, 슈퍼컴퓨터, 그외 특수 하드웨어에 연결시켜 네트워크 대역폭과 연결정도를 증가시키며 자원의 활용도를 높이는데 그렇게 하기 위해서는 사용자의 워크스테이션에 필수적인 통신 프로토콜을 구현해야 한다. 즉 Sun 네트워크 파일 시스템의 파일을 접근하기 위해서는 NFS 프로토콜을 구현해야 하며 Andrew 파일 시스템의 파일을 접근하기 위해서는 AFS 프로토콜을 구현해야 한다. 마찬가지로 Sun 운영체제에 의해 제공되는 서비스를 호출하려면 Sun RPC 프로토콜을 구현하고 Sprite 운영체제에 의해 제공되는 서비스를 호출하려면 Sprite RPC 프로토콜을 구현해야 한다. 일반적으로 워크스테이션에 프로토콜이 많으면 많을수록 접근할 수 있는 자원의 수도 많아진다. 단일 파일 접근 프로토콜인 RPC 프로토콜과 디렉토리 프로토콜은 문제를 단순화시키지만 분산 운영체제가 지역 네트워크에 적합한 싱글 프로토콜을 지정하는 방법으로는 전역 네트워크 자원에 접근할 수 없다. x-kernel은 네트워크 프로토콜 구현이 쉬운 구조를 제공함으로써 지역 네트워크 자원과 원격 네트워크 자원을 접근할 수 있게 하며 프로토콜의 라이브러리를 제공한다.

2. 실시간 CORBA

최근 네트워크가 발전하고 소프트웨어의 복잡도가 커지면서 이질의 분산 환경에서 동작하면서 이식성, 재사용성, 상호운용성을 가지는 객체 기반의 소프트웨어에 대한 중요성이 점점 더 커지고 있다. 이러한 요구에 부응하기 위하여 객체 지향 기술을 선도한다고 할 수 있는 OMG(Object Management Group)에서 분산 객체 관리의 표준으로 제안한 구조가 CORBA(Common Object Request Broker Architecture)이다(Wolfe 등 1997 ; OMG, 1998).

실시간 CORBA는 개발자로 하여금 ORB 자원 할당에 보다 직접적인 제어가 가능하도록 하고 CORBA 요소들의 실행 시간을 설정할 수 있으며 우선순위 부여가 가능하며 스케쥴링이 가능하고 시간에 민감한 응용이나 프로세스 제어 응용에 유용하다.

실시간 ORB는 종단간의 예측을 가능하게 하기 위한 제어를 기반으로 고정된 우선순위 스케쥴링과 유연한 통신으로 구성된다. 실시간 CORBA의 가장 큰 이점은 더욱 결

정적인 수준으로 OMG에 의해 제안된 신뢰할 수 있는 분산 객체 모델을 사용할 수 있다는 것이다.

CORBA의 분산 객체라는 개념은 객체의 성격을 가지면서도 네트워크로 연결된 어디에서든지 접근이 가능하다는 성질을 포함한다. 이 CORBA 객체는 컴파일된 목적 코드로 패키지화 되어 원격의 클라이언트가 함수를 호출함으로써 접근할 수 있다. 이러한 CORBA 객체는 특정 프로그래밍 언어와 운영체제 하에서 생성되지만 다른 프로그래밍 언어와 다른 운영체제 하의 클라이언트에서 접근되는 투명성을 가진다. 이러한 투명성은 기존의 원격 함수 호출 방식인 RPC나 소켓 방법에서는 가지지 못하는 성질로 매우 중요한 의미를 가진다.

OMG IDL은 서비스를 제공하는 분산 객체의 영역과 잠재적인 클라이언트에 대해 인터페이스를 명시할 수 있도록 한다. 즉 분산 객체의 구현자는 이 객체의 IDL을 통해 객체의 포맷을 제공하게 되고 결국 이 분산 객체가 제공하는 서비스를 이용할 잠재적인 클라이언트가 이용하게 되는 것이다

이러한 CORBA의 IDL 언어 바인딩을 그림으로 설명하면 <Figure 6>과 같다. 그림에서와 같이 클라이언트와 서버는 ORB라고 하는 CORBA의 객체 버스에 연결되어 각각 프로그래밍 언어에 상관없이 IDL이라는 언어 바인딩을 통해 통신하고 있다. 클라이언트-서버 각각은 특정 프로그래밍 언어로 구현되어 있다고 하더라도 클라이언트는 IDL 바인딩을 통해 투명하게 서버가 제공하는 서비스를 제공받을 수 있다(OMG, 1998).

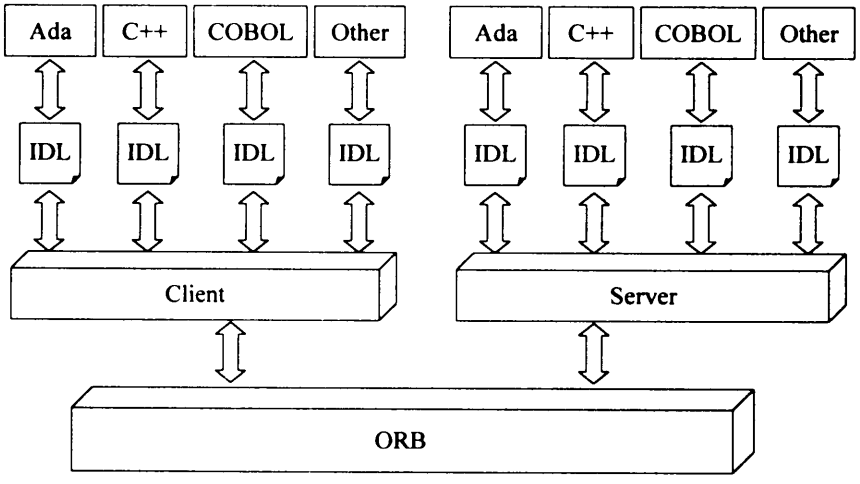

Figure 6. CORBA IDL binding

CORBA 분산 객체는 독립성(autonomous)과 협동성(collaboration) 기능을 갖는데 이 두 가지 기능은 다소 상반되는 개념으로 기존의 미들웨어는 두 가지 기능을 동시에 만족시키는데 한계가 있었지만 CORBA는 분산 컴퓨팅 기술과 객체 지향 기술을 통합함으로써 두 가지를 동시에 제공하고 있다(Farley, 1998).

이렇듯 CORBA의 각 분산 노드상의 객체들은 독립적으로 자신의 계산 작업을 수행하면서 각각 클라이언트와 서버의 기능을 하고 있는데 CORBA의 궁극적인 목표는 분산 객체간의 상호작용 뿐만 아니라 협동성을 추구하는 것이다. 기본적으로 네트워크 프로그래밍은 서버의 위치, 이기종간의 바이트 순서, 메시지 이동 등을 고려해야 하는 시스템 프로그래밍의 성질을 가진다. CORBA ORB는 이러한 여러 작업을 시스템이 자동으로 처리하도록 함으로써 지역적 투명성은 물론이고 원격 투명성을 가진다.

함수 호출의 다형성(polymorphism)은 CORBA가 객체 지향 기술을 기반으로 하기 때문에 가능한 성질로 CORBA ORB는 호출자로 하여금 원격 함수를 호출하는데 있어서 단순히 원격 함수를 호출할 뿐 아니라 특정 객체에 대한 함수를 호출할 수 있도록 한다. 이 방법은 기존의 원격 함수 호출 방법과 비교할 수 있다. <Figure 7>과 같이 RPC 방식은 단순히 원격 함수를 호출할 수 있지만 ORB 방식은 다형성을 이용해 같은 함수를 호출하더라도 객체의 타입이 다르면 다른 효과를 낼 수 있다.

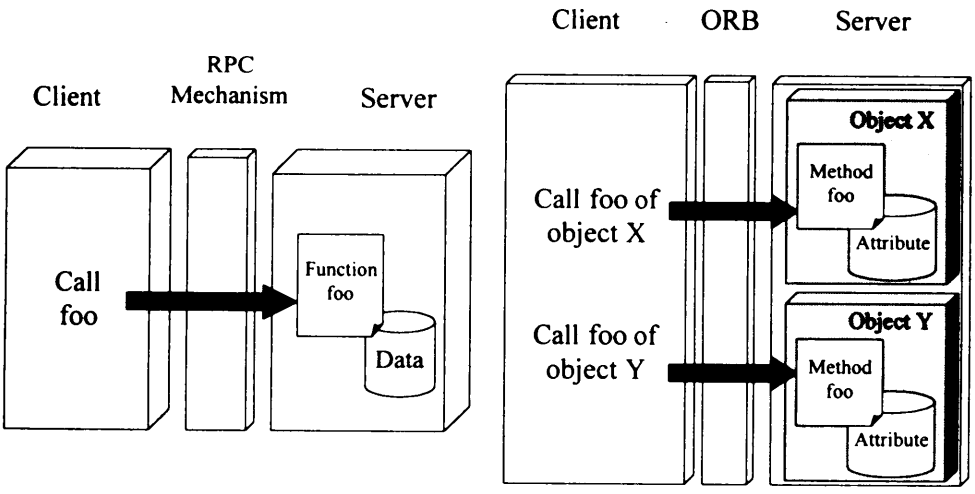

Figure 7. ORB vs. RPC

RPC와 같은 통신 프리미티브들의 시간 제약 조건을 만족시키기 위해서는 우선적으로 시간 제약조건을 표현할 수 있어야 하며 시스템은 이에 의해 프리미티브들이 생성하는 메시지들을 응용의 요구에 맞추어 스케줄하여야 한다. 실시간 교환의 한 예로서 실시간 CORBA는 동적 CORBA 환경에서 분산 객체들간의 상호 작용에 있어서 종단간 시간 제약조건을 지원하도록 설계되었다(Schmidt 등 1998). 동적 CORBA 시스템에서 전역적 우선순위를 기반으로 한 스케줄링을 전역적 우선순위를 기반으로 한 스케줄링을 통한 시간 제약조건을 수행하는데 대하여 최선책을 구현하지만 경성 실시간 보증을 제공하지는 않는다.

실시간 CORBA는 CORBA 시스템에서 종단간의 실시간 제약조건을 만족시키며 수행시킬 수 있어야 한다. 클라이언트가 서버로부터 제한시간 내에 get 함수를 수행하는 시나리오를 생각해보자. 상호작용이 이루어지려면 클라이언트는 요청 메시지에 시간 제약조건을 표현해야 하며 CORBA 시스템은 시간 제약조건을 수행할 수 있는 ORB와 객체 서비스를 제공해야 한다. 또한 통신하는데 사용되는 클라이언트와 서버 노드의 하부 운영 체제는 실시간 제약조건을 수행할 수 있어야 한다.

Iona Technologies는 실시간 운영체제에 비실시간 ORB를 인스톨함으로써 실시간 CORBA에 대한 접근을 시도한 바 있으며 Lynx와 VXWorks 실시간 운영체제에 비실시간 ORB를 구현하여 Real-Time ORB를 발표했다(Wolfe 등 1997). 그러나 이러한 ORB는 운영체제의 대부분의 실시간 특성을 이점으로 활용할 수 없으며 분산 시스템에서 종단간 시간 제약조건을 만족시키기에는 불충분하다.

Lockheed Martin 등에 의해 개발된 ORB를 포함하여 여러 프로젝트들은 불필요한 부분을 모두 제거하고 빠르면서도 기존의 ORB 버전인 실시간 ORB를 구현하고자 하였다. 따라서 CORBA의 동적 호출 인터페이스와 같은 특성을 제거하고 클라이언트와 서버와의 고정된 종단간 연결을 가능하게 하였다. 이러한 특성은 실시간 CORBA 시스템에 필수적인 요소이지만 종단간 시간 제약 조건을 예측하기에는 불충분하다.

CORBA 시스템에 종단간 시간 제약조건의 표현과 수행을 구체화한 첫 번째 프로젝트는 MITRE에서 Peter Krupp와 Bhavani Thuraisingham 팀에 의해 설계되었다(Thuraisingham 등 1994). 이 작업은 명령 및 제어 시스템에 실시간 CORBA를 사용할 수 있게 하고 Xerox로부터 Lynx 실시간 운영체제까지 ILU ORB를 연결시킴으로써 원형화시켰다. 그리고 rate-monotonic과 deadline-monotonic 기법을 지원하는 분산 스케줄링 서비스를 제공한다.

Washington 대학에서는 TAO(The ACE ORB)라 불리는 실시간 CORBA를 위한 고성능 종단시스템 구조를 개발하고 있다(Harrison 등 1998 ; Schmidt 등 1998). 고성능의 실시간 ORB를 설계하는데 필수적인 성능 최적화가 목적인데 스케쥴링 작업은 경성 실시간 시스템에 초점을 두었다. TAO 시스템은 하부 운영체제와 네트워크가 실시간 보장을 하는 자원 스케쥴링 기법을 제공하며 CORBA 이벤트 서비스에 실시간 능력을 포함시켰다.

CHORUS/COOL ORB는 Chorus 시스템에 의해 개발된 융통성있는 실시간 ORB이다(Wolfe 등 1997). 융통성 있는 바인딩 구조, 최소의 ORB에 최소의 CORBA 제공, 실시간 운영 환경을 제공하는 것이 목적이다. COOL ORB 그 자체로만은 많은 실시간 특성을 제공하지 못하고 CHORUS 운영체제에 의존하는 편이지만 운영 체제의 오버헤드를 줄여준다는 장점을 가지고 있다.

Ⅳ. 실시간 RPC 성능 향상 기법

1. 분할 기법

실시간 RPC 성능을 향상시키기 위한 본 논문의 기본 아이디어는 이중화된 연결 구조를 이용하는 것이다. 이 구조로서는 FIFO 스케쥴링 기능을 갖고 있는 ATM 지역망 (Newman, 1994), 이더네트와 같은 비실시간 네트워크, 부하제어 서비스 클래스를 지원하는 인터넷의 하부구조 등이 포함되는데 이 구조에서 네트워크 혹은 연결을 FIFO 큐로 가정하면 <Figure 8>과 같이 단순화된다.

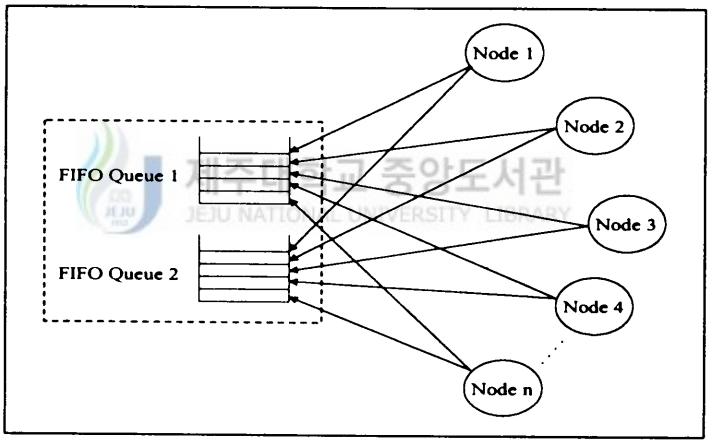

Figure 8. Nodes connected to the dual FIFO queues

부하가 낮은 큐 혹은 네트워크는 큐에서 대기하는 시간이 짧으므로 메시지의 지연시간이 단축되어 종료시한 만족가능성이 높다. 따라서 두 네트워크의 부하를 차별화하고 RPC 트랜잭션에 속한 메시지의 여유시간에 따라 종료시한이 촉박한 경우에 부하가 낮은 네트워크를 통하여 전송하게 한다면 시스템 전체적인 RPC 종료시한 만족도를 높일 수 있다. 결국 RPC 트랜잭션에 속한 메시지의 분배정책(dispatch strategy)이 비실시간 네트워크의 예측 불가능성을 극복하는데 영향을 줄 수 있다. 본 논문에서는 분배정책에 초점을 둔 세가지 분할 기법, 즉 이더네트 적용방안의 여유시간 분할 기법, 타입 분할 기법과 부하제어 서비스 적용방안의 여유시간 분할 기법을 제안하고 그 성능을 평가한

다. 제안된 기법은 각 RPC 메시지로 하여금 여유시간에 따라 하나의 연결을 선택하게 함으로써 각 RPC 트랜잭션에 여유시간에 따른 우선순위를 부여할 수 있으며 분할 기준치의 효율적인 선택에 의해 종료시한 만족도와 같은 실시간 성능의 향상을 기할 수 있다.

각 큐를 통한 RPC 전송에 있어서 주요 특성은 다음과 같다. RPC가 3단계의 수행 단계를 거치게 되므로 트랜잭션 절차 수행 중간에 종료시한을 놓칠 수가 있다. 예를 들어 네트워크 큐나 서버 큐에서 대기중인 시각에 이미 종료시한을 초과할 수 있는데 이런 경우 결과가 대개 무의미해 지므로 이미 종료시한을 놓친 트랜잭션의 남아있는 단계를 수행하는 것은 필요없이 네트워크 시간이나 서버에서의 CPU 시간을 낭비할 수 있으며 다른 요청에 대한 수행을 연기시키게 된다. 그러므로 서버가 그러한 요청을 실행하지 못하도록 해야 한다. 게다가 MAC 계층의 프로시저를 수행하기 전에 네트워크 큐에서 종료시한을 초과한 메시지를 기각시킬 수 있다(Lee 등 1997). 따라서 종료시한을 놓친 메시지의 전송을 어느 경우에라도 피할 수 있다.

두 개의 메시지 큐에 연결되어 있으므로 메시지가 도착하면 <Figure 9>에서와 같이 여유시간 분포에 따라 여유시한이 촉박한 c 영역에 속하는 메시지를 한 쪽 큐에 삽입시키고 여유시한이 촉박하지 않은 $(1-c)$ 영역에 속하는 메시지는 다른 쪽 큐에 삽입시킴으로써 구분할 수 있다. 따라서 분할 기준치(chop value) c를 기반으로 네트워크를 선택하는데 여유 시간이 B. Gao 등의 연구에서처럼 S_{min}에서 S_{max}까지 균일하게 분포한다고 가정한다면(Gao 등 1996), 여유시간 분포상 0부터 c 범위에 해당하는 RPC 트랜잭션에 속한 메시지들은 부하가 낮은 네트워크 1을 통해 전송하고 나머지 $(1-c)$에 해당하는 RPC 트랜잭션에 속한 메시지들은 네트워크 2를 통해 전송하도록 분배한다. 여유시간이 촉박한 메시지가 부하가 낮은 네트워크를 통해 전송되므로 여유시간 분할 방식은 메시지에 대한 종료시한을 만족시킬 가능성을 높인다.

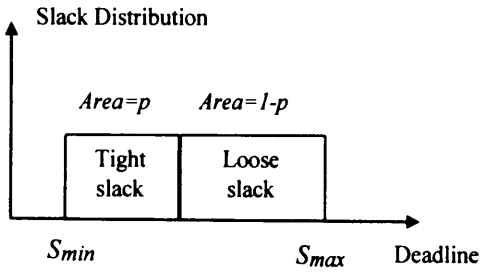

Figure 9. Slack partition

2. 이더네트 적용방안

이더네트와 같은 비실시간 네트워크는 메시지들이 FIFO 순으로 전송되고 각 메시지에 대한 MAC 수행시간에 대한 사전 예측이 불가능하므로 보장에 의한 스케쥴 역시 불가능하다. 요청, 실행, 결과전달의 전체 수행 시간 중 네트워크를 거치는 요청, 결과 전송하는 MAC 시간에 좌우되어 MAC 과정이 전체 RPC 수행 과정을 연장시키게 된다. 그리고 MAC의 특성상 네트워크 카드로 전송된 메시지에 대해서는 컴퓨터(CPU)에서 통제가 불가능하여 오직 MAC 시작 이전에만 기각이 가능하다.

이중 네트워크에서의 실시간 RPC 스케쥴링 기법에서는 각 노드가 <Figure 10>에서 보는 바와 같이 두 개의 네트워크에 연결되어 있다. 이중 네트워크는 실시간 네트워크는 아니지만 일반적인 표준 네트워크로 FCFS 순에 따라 메시지를 전송하는 이더네트가 여기에 속한다. 메시지가 노드에 생성되면 두 개의 FCFS 큐중에 적당한 네트워크를 통해 전송되도록 한 쪽 큐에 추가된다. 메시지가 큐의 첫 번째에 위치하게 되면 MAC 계층 프로토콜, 즉 이더네트에서는 CSMA(Carrier Sense Multiple Access)/CD(Collision Detect)에 의해 전송되는데 전송 지연시간은 기하학적인 분포를 따른다(Dao 등 1991). 즉 k 지연시간을 가진 메시지의 확률은 P를 네트워크 부하요소, 즉 한 노드가 전송할 확률이라 하고 네트워크에 k개의 노드가 있다고 할 때

$$\Pr(k) = P \cdot (1-P)^{k-1} \qquad (1)$$

와 같이 표현된다(Lee 등 1990). 이러한 확률식에서 평균 지연시간은 추정이 가능하지만 최대 전송시간은 무한대이므로 지연시간은 일반적인 네트워크에서 미리 예측이 불가능하다. 메시지는 큐간에 메시지의 이동이 불가능하며 어느 네트워크에 어떤 메시지들을 분배할 것인지가 종료시한 만족도에 있어서 가장 영향을 주는 문제이다.

<Figure 10>은 이중의 네트워크에 연결된 각 노드가 두 개의 FIFO 큐를 가지고 있어 요청 메시지가 도착하면 둘 중 하나의 큐에 삽입되어 네트워크로 전송되는 이중의 비실시간 네트워크를 보여준다.

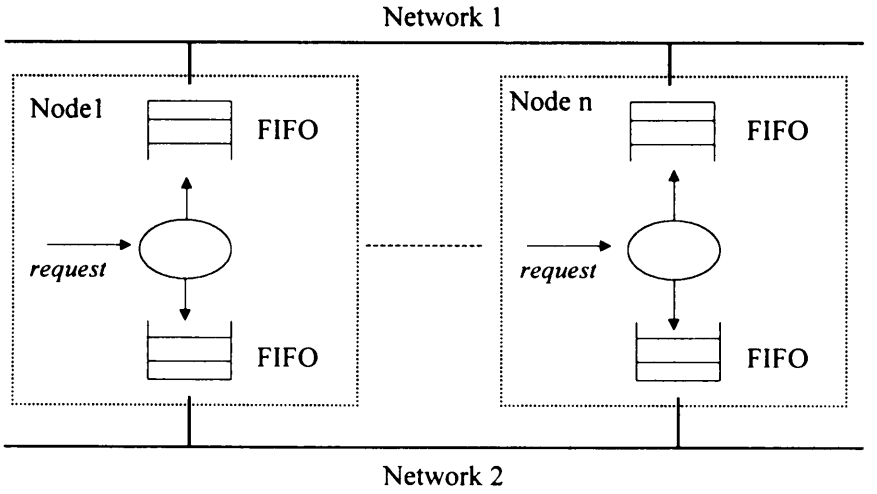

Figure 10. Dual non-real-time network

1) 여유시간 분할

각 노드에 있어서 요청 혹은 결과 메시지가 생성되었을 때 두 연결 중 하나를 선택하여 전송되어야 하는데 분배하는 기법으로서 균등 분할(even partition) 방식과 여유시간 분할(slack partition or chop partition) 방식을 고려할 수 있다.

균등 분할 방식은 메시지를 두 연결에 똑같은 확률로 할당하여 충분한 네트워크 부하에서 메시지의 전송시간을 반으로 감소시킨다. 반면 여유시간 분할 방식은 각 RPC 여유시간에 따라 분배하는데 여유시간은 일반적으로 작업의 종료시한과 실행시간 사이의 차이로서 정의되며 작업의 시간 제약조건을 만족시키기 위해서는 늦어도 여유 시간 이내에 작업의 수행이 시작되어야 한다.

균등 분할 기법은 평균적으로 각 연결 큐의 평균 길이와 전송시간이 동일할 뿐 아니라 두 연결의 RPC 부하를 같게 하므로 단일 연결로 모든 대역폭을 할당받는 경우와 동일하게 되는데 균등 분할 방식은 모든 RPC 트랜잭션의 여유시간이 같을 때에 최선책이 된다. 그러나 각 RPC 메시지는 다른 여유시간을 갖기 때문에 촉박한 종료시한을 갖는 RPC가 생성한 메시지는 신속히 전송되어야 하는 반면 여유시간이 충분한 RPC에서 생성된 메시지는 전송시간이 여유시간 범위 내에서 연장되어도 종료시한을 만족시킬 가능성이 높다.

여유시간 분할 방식은 각 RPC 메시지를 트랜잭션의 여유시간에 따라 각 연결에 분배하며 RPC의 여유시간을 계산하는데 필요한 수행시간은 서버에서의 수행시간과 네트워크 전송 시간의 합으로 구성된다. 서버에서의 수행시간은 사전에 예측이 가능하거나 최소한 최악의 수행시간에 대해 계산할 수 있는데 반해 네트워크에서의 수행시간은 정확한 예측이 불가능하므로 메시지의 길이에 의해 결정되는 순수한 전송시간으로 설정한다. 결국 같은 RPC에 속한 요청과 응답 메시지는 같은 연결을 통해 전송된다. 여유시간 분할 방식은 RPC 트랜잭션들의 여유시간 분포가 사전에 알려져 있다는 가정 하에 한 연결의 부하를 낮게 유지하여 여유시간이 촉박한 메시지들을 전송하고 또다른 연결에는 종료시한 만족에 있어서 비교적 여유가 있는 메시지를 전송한다.

2) 타입 분할

타입 분할(type partition)은 여유시간 분할 방식에서 결과 메시지에 높은 우선순위를 부여하는 분배 정책이다. 요청 메시지와 결과 메시지에 똑같은 우선순위를 부여하여 구분없이 네트워크에 분배한 여유시간 분할과 달리 타입 분할에서는 좀 더 많은 결과 메시지가 낮은 부하 네트워크로 전송될 수 있도록 한다. 이는 결과 메시지 전달에서의 종료시한 초과가 요청 메시지 전달에서의 전송시간 연장보다 전체적으로 성능에 주는 영향이 크기 때문이다. 따라서 응답 메시지는 비율 p에 대해 낮은 부하 네트워크로 전달하고 비율 $(1-p)$에 대해서는 높은 부하 네트워크로 전달하는 기법이다.

요청메시지의 수를 N_q, 응답메시지의 수를 N_p라 하고 낮은 부하 네트워크를 통해 전송되는 요청 메시지의 분할비를 q, 응답 메시지의 분할비를 p라 하자. M_q와 M_p를 요청 메시지와 응답 메시지의 평균 크기라 하고 c(partition factor)가 여유시간에 따른 분할기준치를 나타낸다고 하면 다음과 같은 식을 얻을 수 있다(이, 2000).

$$N_q q M_q + N_p p M_p : N_q(1-q)M_q + N_p(1-p)M_p = c : 1-c \qquad (2)$$

식(2)에서 실시간 시스템이 종료시한 만족도를 충분히 크게 함에 따라 요청 메시지와 응답 메시지의 수는 거의 같아진다. 즉 N_p와 N_q의 비율이 1에 가까워진다. 따라서 식(3)이 식(2)에서 유도될 수 있다.

$$q+pm : 1-q+(1-p)m = c : 1-c \quad (3)$$

이 때 m은 $\frac{M_p}{M_q}$로 계산되는 평균 메시지 크기의 비율이다. 식(3)를 풀면 식(4)에 보여지는 바와 같이 p와 q 사이의 관계가 성립된다.

$$q = (1+m)c - pm \quad (4)$$

결과적으로 스케줄러가 낮은 부하 네트워크로 전송되는 응답메시지의 분할비 p를 결정하면 q가 계산된다. p의 하한은 c가 되며 이때 타입 분할은 여유시간 분할과 동일하게 된다. 종료시한 만족도를 최대화시키는 c와 p의 값은 실험적으로 또는 주어진 네트워크 인자에 대한 분석적 모델을 통하여 구할 수 있으며 구해진 값은 네트워크 인자의 변화에 따라 조정될 수 있다(Gao 등 1996).

3. 부하제어 서비스 적용방안

실시간 응용이 교환하는 메시지는 시간 제약 조건을 갖는데 이를 만족시키기 위해서는 응용의 시간제약 요구나 서비스의 질 요구에 따라 응용의 수행에 필요한 자원을 할당하여야 한다. 자원의 사전 할당에 의해 일정 수준의 종료시한 만족도를 보장하게 되는데 보장 과정은 네트워크나 운영체제 등 하부 플랫폼의 특성에 따라 최선노력 수준에서부터 결정적인 수준까지 제공될 수 있다. 현재 인터넷의 하부구조는 결정적인 수준보다는 최선 노력 수준으로 실시간 응용을 지원할 수 있으며 부하제어 서비스는 자원 할당 방식의 한 예로서 실시간 연결에 대한 용량제어 기능을 제공하여 경로상의 요소들의 부하를 적정하게 유지할 뿐 아니라 자원을 사전에 할당함으로써 일부 요소가 과부하 상태가 될 지라도 연결 설정 과정에서 승인된 서비스를 제공할 수 있도록 한다.

1) 여유시간 분할

인터넷과 같은 최선 노력 수준으로 실시간 응용을 지원하는 하부구조에서 RPC 메시지들의 종료시한 만족도를 개선하기 위해서는 메시지의 종료시한을 고려한 스케줄링이 필요하여 종료시한이 촉박한 RPC 메시지는 종료시한에 여유가 있는 메시지 보다 신속히 전송이 완료되어야 한다. 그러나 하부 네트워크가 단순히 FCFS 순으로 전송한다면 이와 같은 우선순위를 부여할 수 없다. 네트워크의 부하가 낮을수록 메시지의 전송 시간은 단축되므로 이중화된 네트워크 연결을 통해 두 연결의 부하나 대역폭을 다르게 유지하고 낮은 부하의 연결을 통해 촉박한 시간 제약조건을 갖는 RPC 메시지를 전송한다면 종료시한에 따른 우선순위를 부여할 수 있게 되어 종료시한 만족도를 개선할 수 있다.

연성 실시간 시스템을 구축함에 있어서 객체들간의 실시간 상호작용을 지원하기 위해 부하제어 서비스 클래스의 이중화된 그룹 연결을 기반으로 실시간 RPC의 종료시한 만족도를 개선할 수 있다(이, 2000). <Figure 11>은 일반 네트워크에 연결된 각 노드가 이중화된 그룹으로 연결되어 부하제어 서비스를 제공할 수 있도록 하는 그룹 연결 설정이다.

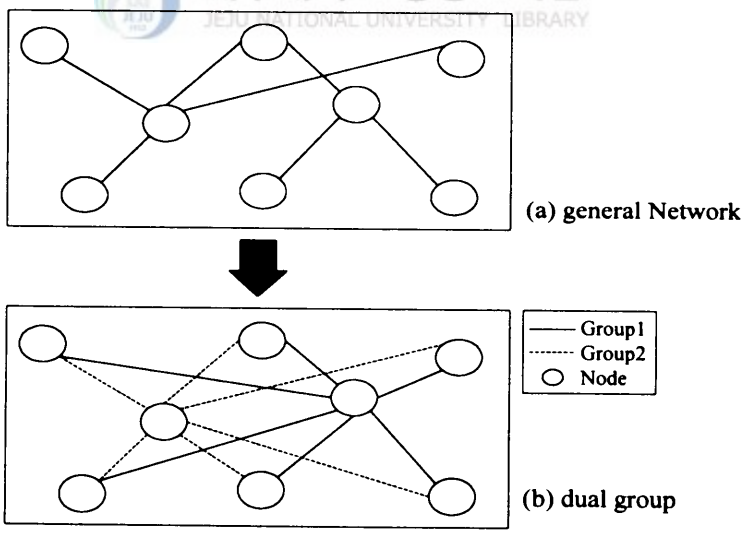

Figure 11. Network with the dual group connection

종료시한을 만족시키기 위해서는 메시지 교환에 필요한 대역폭을 사전에 할당받아야

하는데 이를 위해서는 RPC를 교환하는 노드들간에 그룹 연결이 설정되어 있어야 한다. 그룹 연결을 설정함에 있어서 부하제어 서비스는 연결에 개입된 네트워크 요소들의 자원을 할당받는다. 이때 상이한 경로를 이용하여 두 개 이상의 연결을 설정할 수 있을 뿐 아니라 각 경로의 대역폭도 다르게 할당받을 수 있다. 특정 경로를 통한 연결이 응용이 요구하는 대역폭을 모두 할당할 수 없는 경우도 발생하며 이 경우 충분한 대역폭을 할당받기 위해서는 또다른 그룹 연결을 설정하여야 한다. 결국 RPC를 교환하는 노드들간에 다수의 그룹 연결을 설정하여 대역폭을 할당받는다. 하나의 RPC 트랜잭션은 하나 이상의 노드가 개입된 일련의 수행 과정을 포함하므로 트랜잭션 수행 중에 종료시한을 초과할 수 있는데 이 경우 네크워크 큐에서나 서버 큐에서 사전에 종료시한 초과 여부를 판단하여 초과한 트랜잭션을 기각시켜야 한다.

이중 부하제어 서비스 연결을 통한 여유시간 분할에서 여유시간 분포상 0부터 c 범위에 해당하는 메시지들은 그룹 연결 1을 통해 전송하고 나머지 *(1-c)*에 해당하는 메시지들은 그룹 연결 2를 통해 전송하도록 분배한다(Gao 등 1996).

4. 분할 기준치 설정 방안

여유시간 분할 방식은 분할 기준치의 값에 의해 그 성능이 좌우된다. 만약 시스템의 인자들이 변하지 않는 정적인 특성을 갖고 있다면 사전에 다양한 모의실험이나 환경실험을 수행하여 인자들과 최적의 분할 기준치에 대한 관계를 통계적인 모델에 의해 구한 후 시스템 운영 중에는 이 모델에 의해 분할 기준치를 설정할 수 있다. 반면 시스템의 인자들이 동적으로 변화한다면 분할 기준치를 이에 따라 같이 변경하여야 하는데 이러한 동적 적응을 위해서는 네트워크 상에 조정자(coordinator) 노드가 필요하다. 조정자 노드는 네트워크 상의 다른 노드들로부터 주기적으로 RPC 부하와 같은 인자들과 현재의 종료시한 만족도와 같은 상태 정보들을 수집하여 새로운 분할 기준치를 설정하고 이를 방송에 의해 다른 노드들에게 알려 분할 기준치를 갱신하도록 한다. 또 통계적인 실험과 분석에 의해 주어진 네트워크 인자들에 대해 효율적인 분할 기준치를 계산할 수 있는데 이는 다음 장에서 자세히 분석한다.

V. 성능 평가

1. 개요

본 절에서는 제안된 성능 향상 기법에 따른 RPC 스케줄링 기법을 SMPL을 사용한 모의실험을 통하여 성능을 측정한 결과를 제시하고 분석한다(MacDougall, 1987). 모의실험에 있어서 주요한 가정은 다음과 같다. 시스템에서 모든 RPC 트랜잭션들은 동일한 중요도를 가지며 종료시한 만족도가 가장 주요한 시스템 성능 요소라고 가정한다. 또한 각 RPC들은 다른 RPC 트랜잭션과 독립적으로 수행되며 또한 서버는 EDF 정책에 따라 수신된 요청을 스케줄하고 수행한다고 가정한다. 성능 목표는 종료시한 내에 RPC를 수행함으로써 얻어지는 전체 종료시한 만족도를 최대화시키는 것이다.

모의실험은 10 Mbps를 할당받은 이중의 연결에 기반하고 있으며 실험의 단순화를 위해 추가적인 분할(fragmentation)이나 재조립(reassembly) 과정이 없이 하나의 메시지를 통하여 전송될 수 있도록 어떤 메시지도 하나의 MAC 프레임을 초과할 수 없도록 한다. 요청 메시지의 평균 길이는 500 바이트, 반면 응답 메시지의 평균 길이는 1000 바이트로 설정하여 지수 분포를 따르도록 하였다. 그리고 네트워크상에는 RPC 트래픽만이 존재한다고 가정하며 다른 트래픽이 존재하는 경우에도 두 개의 네트워크의 부하를 차별화시키는 것이 가능하다.

Table 3. Default values of the simulation

average interarrival time	0.0008
chop value	0.45
minimum slack ratio	2
maximum slack ratio	32
service time	0.0001

RPC의 종료시한 만족도에 영향을 미치는 평균 도착간격 시간, 평균 서비스 시간, 분할기준치, 여유시간 비율 등을 포함하는 다양한 네트워크 인자들이 있는데 모의실험에서는 그러한 각각의 인자에 따른 종료시한 만족도를 측정한다. RPC의 평균 도착간격 시간은 0.0008초, 평균 서비스 시간은 0.0001초, 최소 여유시간의 비율을 2, 최대 여유시간의 비율은 32로 <Table 3>에 보여지는 고정된 디폴트값을 기반으로 다양한 인자값에 대한 종료시한 만족도를 측정하였다. 이때 평균 도착간격 시간과 평균 서비스 시간에 대한 시간 단위는 1초이고 지수적으로 분포한다고 설정한다.

2. 이더네트 적용방안

1) 여유시간 분할

여유시간 분할에서는 평균 도착시간 간격에 따른 종료시한 만족도, 여유시간 비율에 따른 종료시한 만족도, 분할 기준치에 따른 종료시한 만족도와 평균 서비스 시간에 따른 종료시한 만족도에 대한 실험결과를 분석한다.

첫 번째 실험에서는 평균 도착시간 간격을 변화시켰고 이 값은 RPC 부하를 나타낸다. <Figure 12>에 보여지는 바와 같이 평균 도착시간 간격이 증가함에 따라 여유시간 분할 기법의 종료시한 만족도는 균등 분할 기법보다 최대 7 %까지 향상된다. 그러나 RPC 부하가 심해지는 경우 균등 분할 기법이 평균 네트워크 큐의 길이를 줄이게 되어 더 나은 성능을 보인다. 다른 분할치를 사용한다면 교차점은 바뀌게 되며 최적의 분할치는 다양한 인자값에 따른 RPC 부하에 대한 실험을 통해 얻어진다.

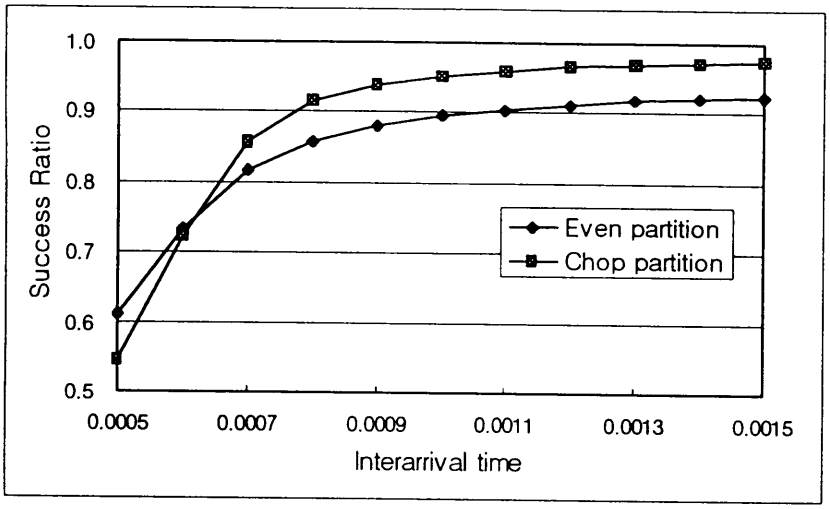

Figure 12. Deadline meet ratio vs. arrival rate

<Figure 13>은 최대 여유시간 비율에 따른 종료시한 만족도를 보인다. 최대 여유시간 비율을 평균 서비스 시간의 10배부터 100배까지 변화시켰을 때 두가지 기법이 모두 여유시간이 증가함에 따라 더 나은 성능을 보이는 것은 당연하다. 여유시간 분할은 여유시간의 모든 범위에 걸쳐 균등분할보다 종료시한 만족도가 높으며 여유시간이 증가함에 따라 향상 정도가 조금씩 더 증가한다.

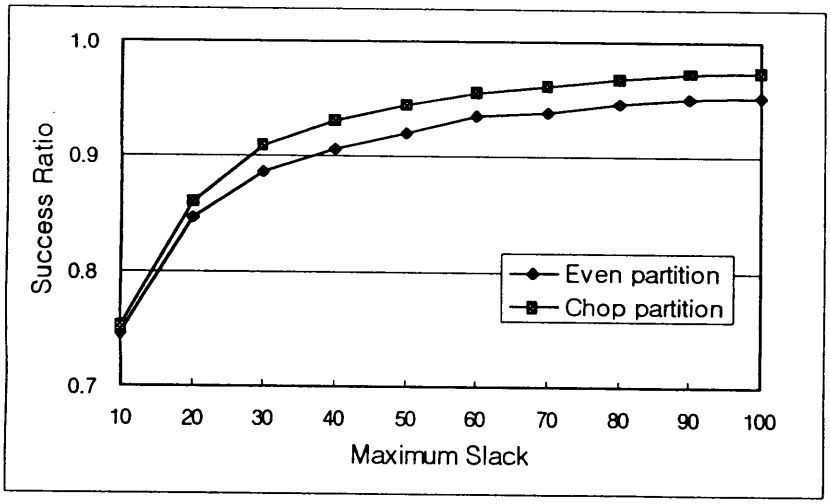

Figure 13. Deadline meet ratio vs. slackness

<Figure 14>는 주어진 인자 집합에 대한 분할 기준치의 영향을 나타낸다. 0.33 이상에서 0.5 사이의 분할 기준치에서 여유시간 분할 기법이 균등 분할 기법보다 종료시한 만족도를 더욱 향상시키며 0.45 부근의 분할 기준치에서는 두 기법에 따른 성능 차이를 최대화함을 알 수 있다. 분할 기준치는 RPC 부하가 증가함에 따라 0.5에 근접하는 경향이 있다.

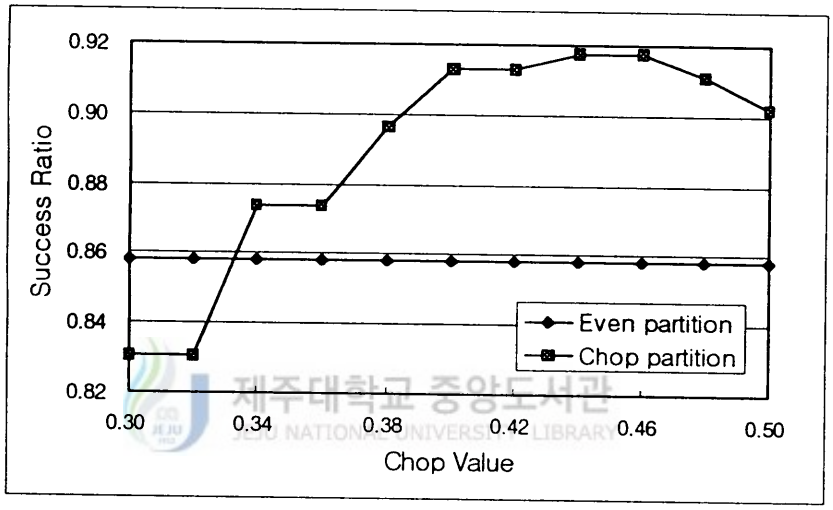

Figure 14. Deadline meet ratio vs. chop value (for the Ethernet)

<Figure 15>는 평균 서비스 시간에 따른 종료시한 만족도를 나타낸다. 이 실험에서는 서비스 시간을 0부터 0.004까지 변화시켰다. 서비스 시간이 증가할수록 메시지 전송 시간은 줄어들며 RPC 고부하 상황과 동일하게 되어 종료시한 만족도를 감소시키지만 여유시간 분할 기법이 균등 분할 기법보다 약간의 성능을 향상시킴을 알 수 있다. 네트워크 부하가 심해져 0.8 아래로 떨어질 때 네트워크는 더욱 높은 대역폭을 갖도록 업그레이드되어야 한다. 개방 시스템의 경우에는 부하 제어 서비스에서처럼 수용 가능한 수준에서 네트워크 부하를 유지하는 것이 필수적이다(Wroclawski, 1997).

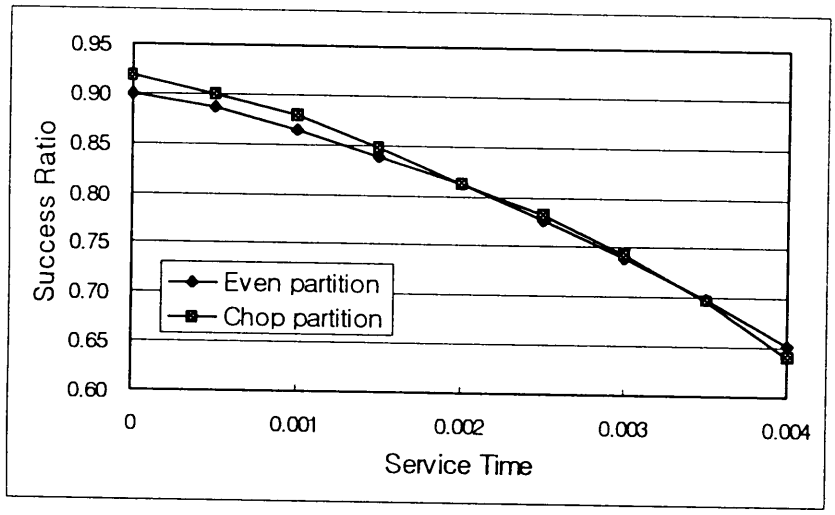

Figure 15. Deadline meet ratio vs. service time

이중의 비실시간 네트워크인 이더네트에서의 적용방안으로 제안된 여유시간 분할 기법은 여유시간에 따라 메시지를 전송함으로써 연성 실시간 RPC의 종료시한 만족도를 높인다. 여유시간 분할 기법은 여유시간이 촉박한 트랜잭션에 속하는 메시지는 부하가 낮은 네트워크를 통하여 전송되는데 반해 여유시간이 촉박하지 않은 메시지는 다른 네트워크를 통해 전송시켰다. 그렇게 함으로써 여유시간 분할 기법 제안은 평균 도착간격 시간, 여유시간 비율, 분할기준치, 평균 서비스 시간과 같은 네트워크 인자에 대해 실험한 결과 비실시간 네트워크의 예측불가능성을 효율적으로 극복하여 성능향상을 가져온다는 사실을 보였다.

2) 타입 분할

응답 메시지의 분실은 네트워크 대역폭이 요청 메시지를 전송하는데 낭비가 될 뿐만 아니라 서버의 계산 시간이 요청된 프로시저 호출을 실행시키는 데 허비되었기 때문에 전체적인 종료시한 만족도에 상당한 해를 끼친다. 이러한 낭비는 다른 RPC 트랜잭션의 종료시한 만족 가능성을 감소시킬 수 있다. 게다가 분할기준치의 선택이 실시간 성능에 결정적이지만 이 값은 많은 다른 인자값에 의존하기 때문에 최적의 분할치를 결정하는 것은 어려운 일이다.

응답 메시지에 우선순위를 둔 타입 분할 기법은 서버와 네트워크 사이의 낭비시간을

감소시켜 여유시간 분할 기법보다 크게 성능향상을 보이지는 못하지만 종료시한 만족도를 증가시키는 결과를 보였다. 여유시간 분할에서 분할기준치가 0.45인 경우 가장 성능향상을 보였으므로 전체 메시지의 분할 기준치를 0.45로 고정시켜서 응답 메시지의 분할 기준치에 따른 모의실험을 수행하였다. <Figure 16>에 보여지는 바와 같이 분할 기준치가 0.45이고 응답메시지의 비율이 0.48일 경우 2.5 %까지 성능 향상을 가져올 수 있음을 알 수 있다. 그리고 기본 인자값에 대한 결과뿐만 아니라 다른 인자값에 대해서도 <Figure 16>에 보여지는 것과 유사한 패턴의 결과를 얻는다.

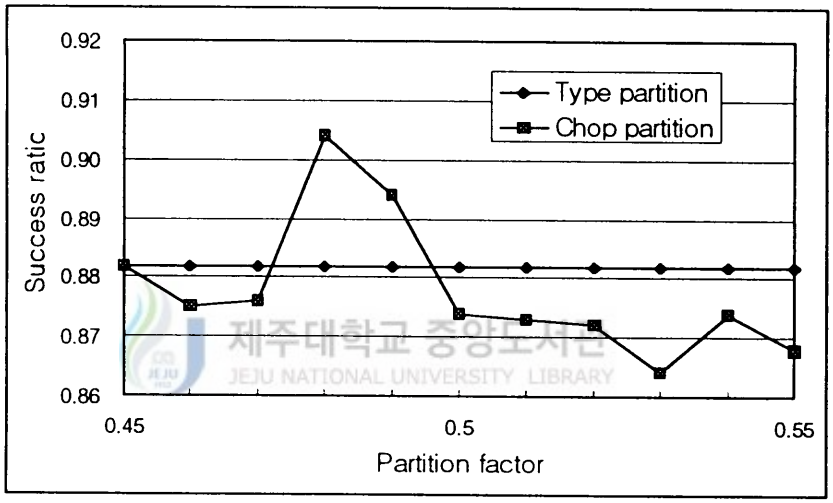

Figure 16. Success ratio according to partition factor

3. 부하제어 서비스 적용방안

1) 여유시간 분할

일반 네트워크 부하제어 서비스 적용방안의 여유시간 분할 기법에서는 5.5 Mbps와 4.5 Mbps를 할당받은 두 연결에 있어서 분할 기준치의 영향을 살펴보았는데 그 결과는 <Figure 17>과 같다.

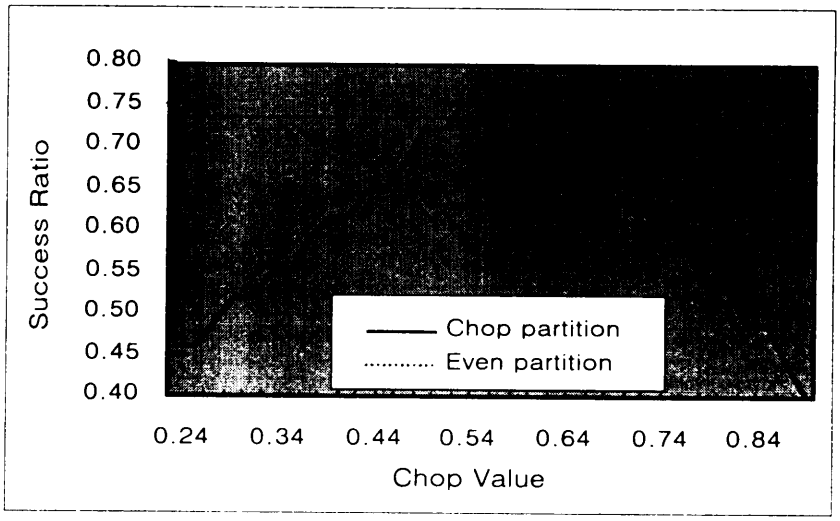

Figure 17. Deadline meet ratio vs. chop value (for the controlled load service)

<Figure 17>은 RPC의 기본 인자값에 대한 실험결과지만 다른 인자값들을 사용한 경우에도 그림과 같은 형태의 종료시한 만족도 패턴을 보인다. 위 실험은 결국 분할기준치의 선택이 단일 10 Mbps 연결과 균등분할에 비해 종료시한 만족도를 개선할 수 있음을 보인다. 대역폭을 변화시켜가며 향상도를 측정한 결과 2 %에서 5 %까지의 향상을 기할 수 있다.

제안된 스케쥴링 기법에 있어서 주어진 네트워크 인자에 대해 최적의 분할 기준치를 설정하는 것이 시스템의 실시간 성능에 가장 중요한 과정인데 이는 실험에 의한 통계분석에 의해 결정될 수 있다. 회귀분석은 본질적으로 독립변수라 불리우는 하나 또는 둘 이상의 변량들에 기초하여 종속변수에 미치는 영향력의 크기를 알아보려고 하는 분석기법이다. 이를 통하여 독립변수가 종속변수와의 관계를 파악하고 종속변수에 미치는 영향력의 크기를 측정하며 독립변수의 일정한 값에 대응되는 종속변수의 값을 예측할 수 있다. 따라서 회귀분석의 대상은 인과관계가 있는 두 변량간의 함수식이 된다고 할 수 있다. 회귀분석을 실시하는 주 목적이 종속변수를 설명할 수 있는 정확한 회귀방정식을 도출하는 것이라는 것은 너무나도 당연한 문제이며 이는 회귀식에 있어서 모수가 되는 α와 β를 정확히 도출해 낼 수 있어야 하는 것이다. 즉 $\hat{Y}_i = \hat{\alpha} + \hat{\beta} X_i$ 로 대변는 표본 회귀선에서 관측치들을 가장 잘 대표할 수 있는 $\hat{\alpha}$값과 $\hat{\beta}$값을 산출할 수 있어야 하는 것이다. 그런데 여기서 관측치에 가장 적합한 표본 회귀선은 실측치와 예측

치의 잔차항 합이 최소가 되는 회귀선이어야 한다. 여기서 잔차항 $e_i = Y_i - \hat{Y}_i$ 로 표현할 수 있으며 이러한 잔차항은 예측을 부정확하게 만드는 요인으로서 회귀식은 이러한 잔차항의 합이 최소가 되어야 할 것이다. 회귀선을 도출하기 위해서 사용하는 가장 일반적인 방법은 잔차항을 최소로 하는 최소제곱법(SS: Sum of Squares)으로 다음의 식으로 표현할 수 있다.

$$MIN \sum e_i^2 = MIN \sum (Y_i - \alpha - \beta X_i)^2 \quad (5)$$

위 잔차항식을 최소로 하는 표본 회귀계수를 구하기 위해 α와 β에 대해 편미분을 실시한다.

$$\frac{\Delta SS}{\Delta \alpha} = -2\sum_i (y_i - \alpha - \beta x_i) \quad (6)$$

$$\frac{\Delta SS}{\Delta \beta} = -2\sum_i X_i (y_i - \alpha - \beta x_i) \quad (7)$$

편미분 값을 0으로 하는 α, β의 각각에 대한 추정치를 $\hat{\alpha}$와 $\hat{\beta}$로 놓고 식을 정리하면 다음의 표본 회귀계수를 얻을 수 있는 식이 도출된다.

$$n\hat{\alpha} + \hat{\beta}\sum x_i = \sum y_i \quad (8)$$

$$\hat{\alpha}\sum x + \hat{\beta}\sum x_i^2 = \sum x_i y_i \quad (9)$$

여기서 $\hat{\beta}$는 표본 회귀선의 기울기가 되며 $\hat{\alpha}$는 절편이 된다(김, 1996).

SAS/STAT에는 회귀분석을 지원하는 절차로 GLM 절차, LIFEREG, NLIN, ORTHOREG, REG, RSREG 등이 있다. 식(10)은 한 네트워크의 대역폭 B_1이 5.5 Mbps에서 8.5 Mbps까지 변화함에 따라 최적의 성능을 보이는 분할 기준치 c를 통계적으로 추정한 결과이다(Frank, 1991).

$$c = 0.0914B_1 + 0.07419 \quad (10)$$

물론 다른 RPC 인자를 사용하면 이에 따라 상수값들이 변화한다(Lee 등 2000). 이와 아울러 <Figure 18>은 실험에 의해 찾아진 최적의 분할 기준치에 의해 추정된 값으로 분할 기준치를 설정하였을 때 생성간격 시간에 따른 종료시한 만족도를 보이고 있는데 성능의 차이는 최대 0.3 % 이내로서 통계적 분석에 의한 방식은 효율적으로 분할 기준치를 설정할 수 있다.

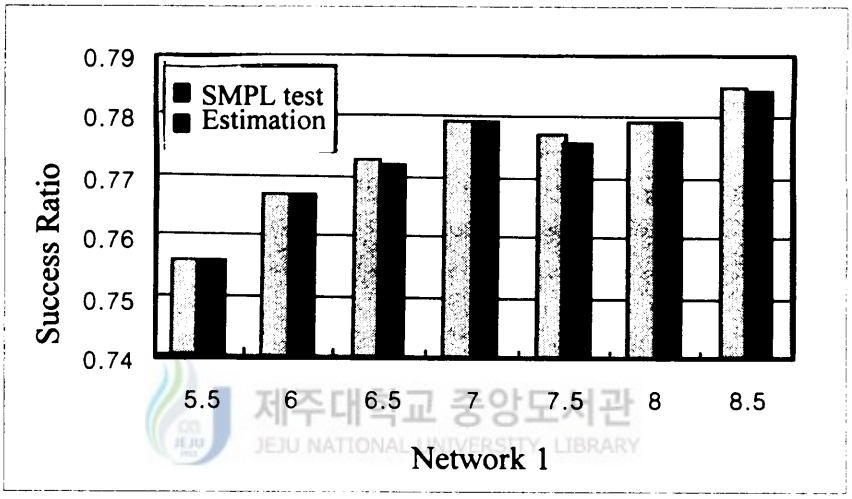

Figure 18. SMPL test vs. estimation

Ⅵ. 결 론

본 논문은 이중의 네트워크 혹은 연결을 기반으로 실시간 RPC 스케쥴링 기법을 제안하고 분석하였다. 제안된 기법들은 두 연결의 부하를 다르게 유지하고 메시지들의 여유시간과 타입에 따라 메시지들을 전송함으로써 연성 실시간 RPC의 종료시한 만족도를 향상시켰다. 여유시간이 촉박한 트랜잭션에 속하는 메시지에 한해서는 낮은 부하 네트워크로 전송하는 방식과 서버 수행시간의 낭비를 감소시키기 위한 타입 분할 방식 그리고 이중의 그룹연결을 통한 부하제어 기법을 제안함으로써 비실시간 네트워크의 예측불가능성을 효율적으로 극복할 수 있었다.

첫 번째 이더네트 적용방안에서는 여유시간 분할 기법과 타입 분할 기법을 제안하고 모의실험 결과를 분석했다. 여유시간 분할 기법은 RPC 트랜잭션의 여유시간에 따라 RPC 메시지들이 전송될 연결을 선택하는 기법으로 그 성능을 모의실험에 의해 측정한 결과 적정한 분할 기준치의 설정으로 균등분할 방식에 비해 RPC의 종료시한 만족도를 개선할 수 있음을 보였으며 이 분할 기준치는 통계적으로 최적치와 0.3 % 이내의 성능 오차 내에서 효율적으로 계산될 수 있었다. 타입 분할 기법은 RPC 트랜잭션의 여유시간에 따라 RPC 메시지들을 분배하는데 요청 메시지보다는 응답 메시지에 보다 우선순위를 두어 보다 많은 응답 메시지를 부하가 낮은 네트워크로 전송하는 기법이었다. 타입 분할의 경우에도 균등분할과 여유시간 분할에 비해 종료시한 만족도를 개선할 수 있음을 보였다.

두 번째 부하제어 서비스 적용방안에서는 여유시간 분할 기법을 제안하고 모의실험 결과를 분석했다. 이 때 RPC 트랜잭션의 여유시간에 따라 RPC 메시지들이 전송될 연결을 선택하는데 두 개의 그룹 연결을 이용하였다. 이중의 그룹 연결을 기반으로 부하제어 서비스를 제공하여 종료시한 만족도를 더욱 향상시킬 수 있었다.

비록 이중 네트워크의 분할기준치와 응답 메시지의 분할기준치가 실시간 성능에 결정적이긴 하지만 많은 인자에 의존하기 때문에 최적값을 구하기가 힘들다. 그러나 보다 다양한 RPC 인자와 대역폭이 다양하게 설정된 경우를 위하여 해석적으로 계산될 수 있어야 한다. 이러한 동적 적응을 위해서는 네트워크 상에 조정자 노드를 두어 네트워크 상의 다른 노드들로부터 주기적으로 RPC 부하와 같은 인자들과 현재의 종료시한 만족도와 같은 상태 정보들을 수집하여 새로운 분할 기준치를 설정하고 이를 방송에

의해 다른 노드들에게 알려 분할 기준치를 갱신하도록 하는 등의 기법이 필요하다.

Ⅶ. 참 고 문 헌

Arvind, K., K. Ramamritham and J. A. Stankovic. 1991. A local area network architecture for communication in distributed real-time systems. *Journal of Real-Time Systems*, 3 : 115-147.

Birrel, A. D. and B. J. Nelson. 1984. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1). 39-59.

Cheriton, D. 1988. VMTP: Versatile message transaction protocol specification RFC 1045.

Chung, S. K., E. D. Lazowska, D. Notkin, and J. Zahorjan. 1989. Performance implications of design alternatives for remote procedure call stubs. *Proceedings of Distributed Computing Systems*. 36-41.

Coulouris, G., J. Dollimore and T. Kindberg. 1994. *Distributed Systems: Concepts and Design*. Addison-wesley.

Dao, M. and K. J. Lin. 1991. Remote procedure call protocols for real-time systems. *Proceedings of IEEE Euromicro Workshop*, 216-223.

Deitel, H. M. 1994. *Operating Systems*. Addison-Wesley Publishing, Inc., 325-358.

Frank, D. C. 1991. *SAS Applications Programming: A Gentle Introduction*. PWS Co.

Gao, B. and H. Garcia-Molina. 1996. Scheduling soft real-time jobs over dual non-real-time servers. *IEEE Transactions on Parallel and Distributed Systems*, 56-68.

Farley, J. 1998. *JAVA Distributed Computing*. O'REILLY.

Harrison, T.H., C. O. Ryan, D. L. Levine and D. C. Schmidt. 1998. The design and performance of a real-time CORBA event service. *IEEE Journal on Selected Areas in Communications*.

Kaashoek, M. F., R. V. Renesse, H. V. Staveren and A. S. Tanenbaum. 1993. FLIP: An internetwork protocol for supporting distributed systems. *ACM Transactions on Computer Systems*, 11 : 73-106.

Lee, I. and S. Davidson. 1990. A performance analysis of timed synchronous communication primitives. *IEEE Transactions on Computers*, 1171-1131.

Lee, J. 1996. Design of a communication system capable of supporting real-time RPC. *4-th Real-Time Application Workshop*, 99-102.

Lee, J. and S. Park. 1997. Design of a DAVIC residential network based on ethernet. *Proceedings of Real-Time Computer System and Application*, 223-228.

Lee, J. and S. Kim. 2000. Design of a real-time RPC scheduling scheme over dual networks. *Proceedings of International Conference on Advanced Communication Technology*, 90-94.

MacDougall, M. H. 1987. *Simulating Computer Systems: Techniques and Tools.* MIT Press.

Newman, P. 1994. ATM local area networks. *IEEE Communication Magazine*, 86-98.

Object Management Group. 1998. The common object request broker: architecture and specification, 2.2.

Peterson, L., N. Hutchinson, S. O'Malley, and H. Rao. 1990. The x-kernel: A platform for accessing internet resources. *IEEE*. 23-33.

Ramamritham, K. and J. Stankovic. 1984. Dynamic task scheduling in hard real-time systems. *IEEE Software*, 65-75.

Schmidt, D., D. Levine and S. Mungee. 1998. The design and performance of real-time object request brokers. *Computer Communications*, 294-324.

Sinha, A. 1992. Client-server computing. *Communication of ACM*, 77-98.

Stevens, W. R. 1990. *UNIX Network Programming*. Prentice Hall, Inc.

Tanenbaum, A. S., R. V. Renesse, H. V. Staveren, G. J. Sharp, S. J. Mullender, J. Jansen and G. V. Rossum. 1990. Experimences with the Amoeba distributed operating system. *Communication of the ACM*, 33(12). 46-63.

Thuraisingham, B., P. Krupp, A. Schafer and V. Wolfe. 1994. On real-time extensions to the common object request broker architecture. *Proceedings of the Object Oriented Programming, Systems, Languages and Applications Workshop on Experiences with CORBA.*

Wolfe, V. F., L. C. DiPippo and R. Johnston. 1997. Real-time COBRA. *Proceedings of the Real-Time Technology and Applications.*

Wroclawski, J. 1997. *Specification of the Controlled-Load Element Service.* RFC 2211.

김종수, 이화진, 정종필. 1996. *SAS 바로쓰기.* 홍진출판사

이정훈, 강미경. 2000. Design of a real-time RPC scheduling scheme over dual non-real-time networks based on RPC semantic. *Joint Conference on Communication and Information.*

이정훈, 강미경. 2000. 이중 부하제어 서비스 연결을 이용한 실시간 원격 프로시저 호출의 성능향상 기법. 정보과학회 추계학술대회.