큐를 이용한 RPC 알고리즘 설계와 성능 평가

윤 동 식[†] · 이 병 관[††]

요 약

본 논문에서는 UNIX System V하에서 마이크로, 워크스테이션 그리고 퍼스널 컴퓨터 다수에서 큐를 이용한 원격 프로시져 호출(Remote Procedure Call:RPC)알고리즘을 설계하고 성능 평가 하였다. 기존 RPC 호출 방식으로 원격 프로시쥬어를 호출한 후에는 반드시 반환이 이루어져야만 다음 명령을 수행할 수 있기 때문에 반환이 이루어지지 않으면 무한정 대기하는 현상이 발생한다. 이 단점을 해결하기 위해 본 논문에서는 큐를 이용한 메세지 전송 방식으로 보안하여 위에서 제시한 문제점을 해결하여 순차적으로 프로세싱할 수 있는 형태로 변경된 원격 프로시쥬어 호출 알고리즘을 설계 구현하였다.

Design and Performance Evaluation of a RPC Algorithm using Queue

Dong Sic Yun[†] · Byung Kwan Lee[††]

ABSTRACT

This paper presents the design technique and the performance evaluation of RPC(remote procedure call) algorithm using queue on the micro computer, workstation and PC's under UNIX System V, The existing RPC algorithm is after calling remote Procedure, it must has be returned in order to run next instructions. But if it is not done so, it must wait indefinitely. To solve this problem, a message transfer method which uses queue is adopted. And the improved RPC algorithm that can process in sequence is designed and implemented.

1. 서 론

컴퓨터 및 통신 기술의 발달에 따라 증가되는 정보량을 기존의 중앙집중식 단일 처리 방식으로 서비스하기에는 한계가 있어 점차 지리적으로 분산되어 처리되는 분산 자료 처리(Distributed Data Processing: DDP)방식으로 변화되고 있다[1]. 분산 시스템에서 운영되는 운영체제를 분산 운영체제(Distributed Operating System:DOS)라 하는데, 분산 운영체제의 일반적인 목적은 다음과 같다. 첫째, 다중프로세서들이 시스템의 작업부하를 상호 협력하여 처리해서 시스템의 성능을 극대화하는 것이다. 둘째, 사용자 프로세스들에게 공정성을 보장하는 것이며, 세째, 시스템의 신뢰성과 결함허용율을 높이는 것이며, 네째, 전체 통신 시간을 줄여 동시에 많은 사용자가 사용할 수 있게 하는것이다.

† 정 회 원:아세아 항공 전문학교 정보통신학과 주임 교수
†† 정 회 원:관동대학교 이공대학 전자계산공학과 부교수
논문접수:1995년 10월 11일, 심사완료:1996년 2월 21일

통신 네트워크로 연결된 컴퓨터들의 집합인 분산 시스템은 각각의 컴퓨터들을 서로 효과적으로 협력시키기 위한 전체적인 통제 시스템을 필요로 하며, 이를 실행해 주는 소프트웨어인 분산 운영 체제를 구현하기 위해서는 필수적으로 프로세스간의 정보교환을 위한 통신 메카니즘이 필요한데 이것이 IPC(Inter Process Communication)이다. 이 IPC 기능은 분산 운영 체제의 구조 및 성능에 커다란 영향을 주는 것으로 우수한 IPC 메카니즘은 전체 시스템의 효율을 증진시킨다[5].

OSI(Open System Interconnection) 참조 모델의 응용 계층은 응용 프로세스 사이의 통신을 제공함으로써 응용 프로세스에게 분산 처리 시스템에 존재하는 여러가지 서비스를 사용할 수 있게 한다. 이러한 서비스 중 응용 프로세스가 원격 노드에 존재하는 프로시쥬어를 사용함으로써 수행을 분산시킬수 있게 하여 주는 응용 서비스를 RPC(Remote Procedure Call)라고 한다[6][7][8][9].

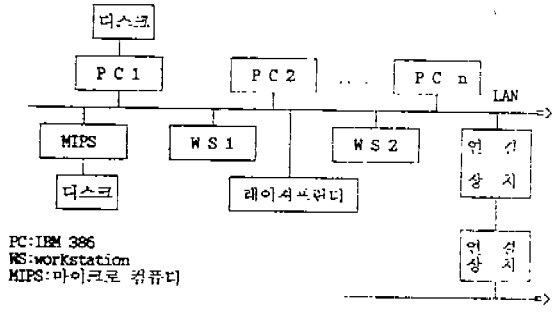
RISC 방식의 MIPS에 SUN 워크스테이션과 PC를 ethernet로 연결 된 것을 이용하여 본 논문에서는 UNIX system V의 기존 RPC 호출 방식의 단점, 즉 원격 procedure를 호출한 후에는 반드시 반환 하여야만 다음 명령을 수행할 수 있기 때문에 반환이 이루어지지 않으면 무한정 대기하는 현상이 발생한다. 이 단점을 해결하기 위해 본 논문에서는 큐를 이용한 메세지 전송 방식으로 보완하여 위에서 제시한 문제점을 해결하여 순차적으로 프로세싱할 수 있는 형태로 변경된 RPC 알고리즘을 설계 구현하였다.

2. 분산화일시스템과 원격프로시듀어호출

2.1 분산 화일 시스템과 환경

본 시스템의 운영 환경은 소형컴퓨터에서 이기종간에 이루어지는 분산 화일 시스템을 구축하기 위하여 다수의 퍼스널 컴퓨터(PC)와 SUN 워크스테이션 그리고 RISC방식의 CPU를 사용한 마이크로 컴퓨터인 MIPS을 IEEE 802 Ethernet을 사용한 LAN에 의해 상호 연결하였다. 이들 기계들은 모두 UNIX SYSTEM V를 운영체제로 사용했다.

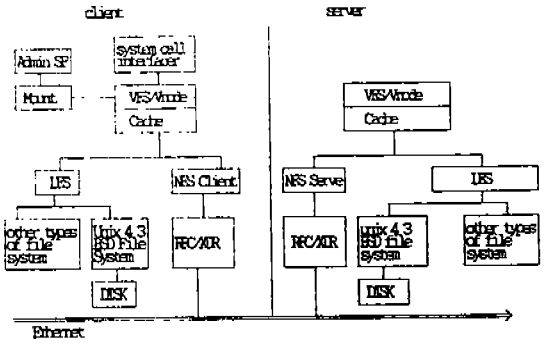
지역적으로 분산되어 있는 컴퓨터들을 이용하여 분산처리 시스템을 구성하므로 분산 화일 시스템에서 발생되는 혹은 시스템 설계시 고려되어야 할 사항은 다음과 같다. 첫째, 각각의 분산된 컴퓨터에 할당된 화일 시스템 구조의 차이와 기존의 화일 시스템과의 호환성 유지에 고려를 해야 한다. 둘째, 시스템을 사용하는 사용자들에게 분산된 화일을 마치 자신의 머신에서 사용하는 것 같이 원격 프로시쥬어 호출(remote procedure call)시에 반드시 투명성(tranparency)이 제공되어야 한다. 셋째, 각각의 머신을 연결 시키는 통신 장비와 논리적으로 연결 시키는 프로그램이 제공되어야 한다. 넷째, 화일 시스템의 설계에서 중시되어야 할 점은 화일 시스템 사용시 신뢰도 및 효율성이며, 화일 시스템의 확장성 또한 고려되어야 할 사항들이다.



(그림 1) 시스템 운영 환경
(Fig. 1) System Operating Environment

본 화일 시스템의 논리적인 구성요소에는 메세지 시스템 호출(message system call)에서의 큐(Queue) 자료구조 형태를 응용하여 원격화일 시스템(Remote File System)에 있는 화일을 지역시스템에 있는 화일과 같이 호출할 수 있도록하는 확장된 RPC와 데이타 전송에 있어서 내부 데이타를 통신될 수 있는 형태로 변환하여 전송할 수 있는 형태로 만들어 주는 XDR (eXternal Data Representation), 그리고 노드들의 상태와 네트워크의 상태를 일정한 시간마다 모니터링하여 RFI(Remote File Interface)에게 알려 주는 daemon 프로세스인 상태 모니터(status monitor)와 화일들의 lock 정보와 캐싱에 사용되는 페이지를 관리하는 Lock 관리자(Lock manager), 페이지 관리자(page manager)등으로 구성되었다.

다음의 (그림 2)에서 보는 바와 같이 클라이언트와 서버 부분으로 시스템을 구분 지을 수 있다. 사용자로부터 화일의 요구가 있을 시에는 클라이언트의 VFS/Vnode는 화일 요구에 따라 지역 화일과 원격 화일로 구분지어진다. 이 구분에서 지역화일인 경우에는 LFS(Local File System)에 의해 수행되어지고 그렇치 않을 경우에는 NFS(Network File System)에 의해 수행되어지게 된다. NFS는 일단 먼저 사용자가 요구하는 화일을 캐쉬 메모리에서 검색이 이루어진 후에 캐쉬 메모리 상에서 요구된 화일의 페이지를 가지고 RFI로 넘어간다. RPC/XDR은 수신한 함수의 인자를 메세지화하여 네트워크을 이용하여 서버로 보내진다. 메세지를 받은 서버의 RPC/XDR는 서버의 RFI의 해당 시스템을 호출(invoke)하여 인자를 할당한다. 호출된 시스템 콜은 서버의 VFS/Vnode를 통해 지역 화일 시스템인 NFS를 사용하여 화일 요구에 대한 처리를 한 후 결과를 역으로 사용자에게 반환한다.



(그림 2) 분산 화일 시스템 구성도
(Fig. 2) VDFS Configuration

2.2 RPC 통신 모델

분산 처리 시스템을 이루는 노드 사이에서는 서로의 정보를 교환할 수 있는 통신 방법이 존재하게 되는데 이러한 통신 방법의 유연성과 성능은 분산처리 시스템의 성능을 결정한다. 통신 방법의 유연성과 성능은 분산 프로그램을 작성하는 프로그래머가 직접 이용할 수 있는 통신에 관한 동작의 집합과 이러한 동작들을 이용하는 방식에 의하여 결정되게 된다. 통신에 관한 추상적인 집합(abstract set of operations)을 정의하고 이러한 동작들을 이용하여 분산 프로그램(distributed program)을 작성할 수 있는 방법을 제공하는 것을 통신 모델이라 하며 통신 시스템이 자신의 서비스를 이용할 수 있도록, 제공하는 기본적인 명령문을 프리미티브(primitive)라고 한다.

이러한 통신 모델은 메세지를 수신(receive)하고 송신(send)하는 형태로 통신하는 메세지 기본 모델과 프로시쥬어를 호출하는 형식으로 통신하는 RPC기본 모델의 두가지로 나누어 생각할 수 있다.

2.2.1 메세지 기본 모델

메세지 기본 모델에서는 상호 작용하는 여러개의 프로세스로 이루어진 모임이 존재할 때 각 프로세스 사이에 통신은 메세지를 보내는 명령인 send와 메세지를 받아들이는 receive라는 두개의 프리미티브를 이용하여 메세지를 전송함으로써 통신이 이루어 진다. 이때 send의 인자로는 전송되어 질 메시지와 메세지를 받아야 할 프로세스의 주소가 사용되며 receive의 인자로는 보내는 프로세스의 주소와 보내어진 메세지가 사용된다. 프로세스가 하나이상의 응답이 발생하기를 기다리며 수행을 멈추고 있는 것을 블럭(block)되었다고 하는데, send 프리미티브는 전송된 메세지에 대한 응답이 올때까지 블럭되어 질 필요는 없다. 목적 노드에 전달된 메세지는 receive프리미티브의 수행에 의하여 목적 노드로 받아 들여지며 프로세스에 의하여 번역 되어지고 수행된다. 메세지 기본 모델에서는 각 응용 프로세스 사이의 통신이 가변적인 크기를 갖는 메세지를 중심으로 이루어지므로 응용 프로세스에게 높은 유연성을 제공하게 된다. 또한 메세지 모델에서는 메세지의 의미가 통신하는 프로세스 사이의 프로토콜에 의하여 결정되므로 프로그래머는 분산 프로그램의 작성이 용이하지 않다.

2.2.2 RPC 기본 모델

RPC 기본 모델은 프로그래머에게 원격 프로시쥬어(remote procedure)를 국부적 프로시쥬어(local procedure)를 사용하는 것과 같은 방법으로 사용할 수 있게 함으로써 프로세스간의 통신을 제공한다. 따라서 RPC의 동작은 동작방법이 정의되어 있는 원격의 프로시쥬어에 필요한 인자(argument)를 전송함으로써 원격 노드로 수행이 분산된다. 원격 노드에서 원격

프로시쥬어에 의하여 수행된 결과 값은 원격 프로시쥬어에 전송된다. 이러한 형태는 프로그래머가 네트워크에 대한 아무런 고려를 하지 않아도 되므로 프로그래머의 관점에서는 매우 간단하고 편리하게 분산 프로그램을 작성할 수 있다. 또한 RPC는 프로시쥬어에 호출의 형태를 하고 있으므로 원격의 프로시쥬어에서 반환되지 않으면 다음의 명령(instruction)을 수행할 수 없다. 따라서 RPC는 원격 프로시쥬어의 결과치가 도착할 때까지 수행되지 않는데 이것은 원격 프로시쥬어를 호출한 프로세스의 블럭을 유발한다.

RPC는 메세지 기본 모델에 비하여 하나의 요구에 대한 응답에 대한 시간인 호출시간(latency time)이 짧은 것을 기본 성격으로 한다. 따라서 요구-응답 응용(request-response application)에 적합하다. 연결의 설정은 호출시간을 짧게 하는데 많은 부담을 주므로 RPC는 트랜스포트와 네트워크 계층이 연결 지향 서비스(connection oriented service)보다는 비 연결지향 서비스(connectionless service)를 제공하는 상황에서 구현되어지는 경우가 많다.[6][7] 따라서 RPC는 많은 양의 데이타를 한꺼번에 보내는데 사용하기에는 부적합하다.

RPC는 원래 프로시쥬어 호출에서 개념을 가져온 것이어서, 한개의 제어 흐름 만을 존재시키므로 원격 프로시쥬에 대한 호출과 복귀를 동기시키고 있다. 이러한 동기적 원격 프로시쥬어 호출의 모델은 순차 수행 모델에는 적합하나, 병렬 수행 모델에는 적합하지 못하므로 분산 환경에 내재되어 있는 병렬성은 충분히 이용하지 못한다. 이러한 문제를 해결하여 분산 환경에 내재된 병렬성을 이용하고자, RPC 개념의 장점을 살리면서 제어의 이전과 복귀를 동기시키지 않는 비동기적 프로시쥬어 호출이 다양하게 제안, 구현되어져 왔다.

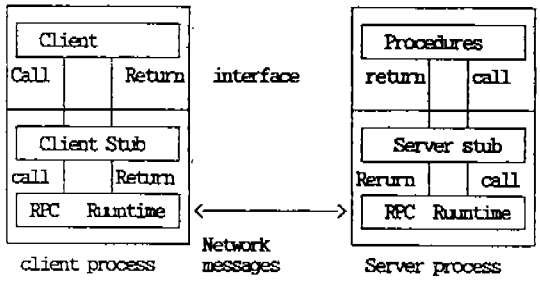
2.3 RPC의 구성요소

RPC는 네트워크 상의 다른 프로세스와 통신하기 위하여 프로시쥬어 호출의 형태를 사용하므로 RPC가 구현되어지는 프로그램 언어의 성격에 따라 실제적인 구조가 달라질 수 있다. 따라서 프로그램 언어는 중요한 요소 중 하나이다.

하나의 프로세스가 원격 프로시쥬어를 사용하기 위하여는 호출하기 위한 프로시쥬어의 네트상의 위치와 그 프로시쥬어의 인터페이스의 정의를 알아야 한다. 따라서 이러한 정보를 가지고 있는 네트상의 데이타베이스(database)가 필요한데, 이것을 RPC 디렉토리 서버(RDS:RPC directory server)라고 명명하여 사용하기로 한다. RDS로 부터 원하는 정보를 얻어와 실제로 사용할수 있도록 결합하는 과정을 바인딩(binding)이라 한다.[10]

RDS는 네트상에 잘 알려진 주소를 가지고 있으므로 RDS에 접근하기 위한 별도의 바인딩은 필요하지 않다.

RPC에서 또 하나의 중요한 요소는 원격 프로시쥬어를 고객에게 제공하는 RPC 서버[10]이다. RPC서버는 하나 이상의 원격 프로시쥬어를 자신의 루틴 안에 포함하고 원격 프로시쥬어의 인자와 결과에 대한 정보와 원격 프로시쥬어 장소에 대한 정보를 네트워크상의 RPC 디렉토리 서버에 등록 함으로써 원격 프로시쥬어를 사용할 수 있는 방법을 제공한다. RPC 서버는 고객 프로세스의 요구에 의하여 자신이 가지고 있는 프로시쥬어 중 하나를 수행하여 그 수행의 결과 값을 반환 하는 역할을 한다. (그림 3)는 RPC서버의 예를 나타낸다.



(그림 3) RPC 논리적 구조
(Fig. 3) Logical Structure of RPC

프로그래머는 네트워크에 대한 세부적인 사항을 고려하지 않아야 하므로 프로그래머부터 네트워크에 대한 자세한 정보를 은폐하기 위하여 사용되는 코드를 스터브(stub)라 한다. 고객 스터브는 원격 프로시쥬어를 요구하는 고객(Client)에게는 실제 원격 프로시쥬어처럼 보이며 원격 프로시쥬어의 스터브는 원격 프로시쥬어에 대한 서비스를 제공하는 RPC 서버

에게는 고객처럼 보이게 된다. 스터브는 네트워크 프리미티브를 이용하여 네트워크에 대한 동작을 하는 루틴으로 이루어져 있으며 상위 계층으로부터 제공받은 인자를 통신에 적합한 형태로 바꾸기 위하여 정렬화/역정렬화(marshall/ unmarshall) 네트워크 프리미티를 사용한다.[10] 스터브는 일반적으로 다음의 형태를 취한다. 스터브를 설명하기 위하여 사용하는 인자와 프로시쥬어를 다음과 같이 정의한다 :

〈인자 정의〉

pname : 원격 프로시쥬어의 기호적 이름을 나타낸다.

address : 원격 프로시쥬어의 주소를 나타내는 인자이다.

r_pname : 실제적인 원격 프로시쥬어의 이름을 나타내는 인자이다.

interface_description : 인터페이스 정의를 나타내는 인자이다.

argument : 원격 프로시쥬어의 인자를 나타내는 인자이다.

〈프로시쥬어의 정의〉

import(r_pname, interface_description, adress);

원격프로시쥬어의 정의를 고객에게 알리기 위하여 pname의 인터페이스 정의와 원격 프로시쥬어의 주소를 네트워크상의 데이타베이스에 알리는 프로시쥬어이다.

marshall(arguments, interface_description);

arguement를 네트워크를 통하여 전송하기 위하여 인터페이스 정의를 이용하여 정렬화하는 프로시쥬어이다.

unmarshall(arguments, interface_description);

네트워크를 통하여 전송받은 argument를 국부적 형태의 데이타로 변경하기 위하여 인터페이스 정의를 이용하여 역 정렬화하는 프로시쥬어이다.

transmit(address, message[message]);

실제적으로 네트워크를 통하여 전송하기 위하여 네트워크 프리미티브를 이용하여 주소 메시지를 전송하는 프로시쥬어이다. 네트워크가 신뢰성있는 통신을 제공하여 모든 메시지가 순서대로 목적지에 도

```
procedure pname(argument);
{
    import(r_pname,interface_description,address) ;
    marshall(argument,interface_description) ;
    transmit(address,r_pname,argument);
    while(not result_arrived);
    receive(result);
    unmarshall(result,interface_description);
    return ;
} ;
```

(그림 4) 프로시쥬어 1 고객 스터브의 형태
(Fig. 4) Procedure 1 Client stub type

```
procedure SERVER_stub;
{
    registration(r_pname, interface_description, address);
    loop {
            while(not arrived_request);
            receive(address, r_pname, argument);
            unmarshall(argument, interface_description);
            local_call(r_pname, argument);
            marshall(result, interface_description);
            transmit(address, result) ;
 };
};
```

(그림 5) 프로시쥬어2 서버 스터브의 형태
(Fig. 5) procedure 2 server stub type

착 한다고 가정한다.

receive(address, message[message]);

네트워크 프리미티브를 이용하여 자신의 주소에 대한 메시지를 전송받는 프로시쥬어이다. 네트워크가 신뢰성 있는 통신을 제공하여 모든 메시지가 순서대로 목적지에 도착한다고 가정한다.

(그림 4)은 클라이언트 스터브이다.

먼저 원격 프로시쥬어의 기호적 이름을 이용하여 네트워크상의 데이타베이스를 접근하여 원격 프로시쥬어의 정의와 주소를 알아낸다.

(그림 5)의 프로시쥬어는 서버의 스터브이다. 서버는 자신이 처음 수행되면 자신이 가지고 있는 원격 프로시쥬어의 인터페이스 정의와 주소를 네트워크에 알리고 요구가 있을때까지 기다리게 된다. 서버는 끝나지 않는 프로시쥬어로써 하나의 서비스 이후에 다시 같은 동작을 반복하게 된다.

정렬화는 스터브에 의해 이루어지며 국부적인 데이타 표현 형태를 갖는 인자와 결과 값을 인터페이스 기술언어에 의하여 기술 되어진 추상적인 인터페이스를 이용하여 네트워크 전송을 위한 전송 구문(transfer syntax)의 형태로 바꾸어 주는 역할을 하는 것을 말하며, 역정렬화는 네트워크를 통하여 받은 정렬화된 인자와 결과값을 추상적인 인터페이스 정의를 이용하여 국부적인 데이타 표현으로 바꾸어 주는 역할을 하는 것을 말한다. 이때 각 인자와 결과값을 정렬화 할때는 인터페이스 정의에 준하여 정렬화 한다.

2.4 RPC의 동작방법

일반적으로 RPC가 동작하는 순서는 다음과 같다. 먼저 원격 프로시쥬어에 대한 서비스를 제공하는 RPC 서버가 수행되면서 자신의 존재를 네트워크 상의 다른 시스템에 알리게 되는데 이것을 등록(registration)이라 한다. 이것에 의하여 다른 프로세스는 RPC 서비스의 존재와 사용방법을 알게 된다. 둘째, 고객 프로세스는 고객의 스터브를 국부 프로시쥬어 호출의 형식을 이용하여 사용하게 된다. 이때 스터브는 실제적인 서버 프로시쥬어로 보이게 된다. 스터브는 고객으로부터 전달받은 인자를 전송에 적합한 형태로 정렬화하여 메세지를 만들게 된다. 세째, 원격 프로시쥬어를 사용하고자 하는 프로세스는 원격 프로시쥬어에 대한 위치와 인자에 대한 정보를 RPC 디렉토리 서버를 이용하여 얻게 되는데 이러한 일을 임포트(import)라고 한다[7][22]. 이때 얻어진 정보를 이용하여 원격 프로시쥬어를 올바르게 부를 수 있다. 네째, 이렇게 만들어진 메세지는 고객 스터브가 국부 커널(local kernel)에게 네트워크를 통하여 원격 시스템(remote system)으로 전달되기 위한 시스템 호출(system call)를 함으로써 국부 커널에 전달 되어 진다. 다섯째, 네트워크 메세지는 연결지향 또는 비 연결지향 프로토콜(protocol)을 이용하여 원격 시스템으로 전달되어진다. 여섯째, 서버 스터브는 고객의 요구를 처리하기 위하여 원격 시스템에 존재하면서 메세지를 기다리고 있다. 메세지가 도착하면 스터브는 인자를 역 정렬화(unmarshall)를 통하여 원격 시스템의 데이타 형태로 바꾸어 주게 된다. 일곱째, 서버 스터브는 실제적인 서버 프로시쥬어를 국부 프로시쥬어 호출을 통하여 수행시키게 된다. 이때 역 정렬된 인자를 넘겨준다. 여덟째, 서버의 프로시쥬어가 수행되어지고 난 다음 서버 프로시쥬어는 결과 값을 반환하고 종료된다. 아홉째, 서버 스터브는 이 결과 값을 정렬하여 네트워크 루틴(routine)에 대한 시스템 호출을 이용하여 고객 프로세스로 전송하게 된다. 열번째, 고객 스터브는 커널을 통하여 메세지를 받는다. 마지막으로 스터브에 의하여 결과 값을 역 정렬한 후에 고객에게 일반적인 프로시쥬어 리턴을 통하여 전송되게 된다.

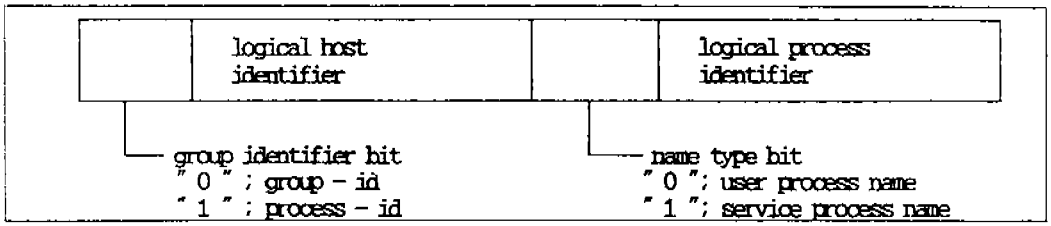
3. 확장된 RPC/XDR 기법

3.1 IPC 페이지 구조

IPC는 최대의 성능을 나타내기 위해 그 환경에 적합하게 설계해야 하며 신뢰성 있고 신속한 처리가 가능한 프로토콜을 가지고 운영되어야 한다.

두 프로세스 사이의 통신은 직접 또는 간접으로 전송될 수 있다. 프로세스는 어떤 서비스나 기능을 제공받기 위해서 프로세스 식별자(identifier)를 알아야 하며 이를 위해 이름(name) 서버는 사용자 프로세스와 서비스 프로세스간의 동적 바인딩(dynamic binding)을 제공할 수 있어야 한다.

특히 직접 방식의 IPC는 명시(explicit)한 전역 처리 식별자(unique global process identifier)를 정의할 수 있어야 하므로 좋은 성능의 이름 서버가 요구된다.



(그림 6) 이름 구조
(Fig. 6) Name Structure

이러한 명시한 pid의 표현은 Kernel로부터 지역(local)프로세스와 원격(remote)프로세스를 쉽게 확인할 수 있어야 하며 프로세스가 존재하는 호스트까지도 확인할 수 있어야 한다.

이름을 크게 나누어 사용자 프로세스 이름과 서버 이름으로 구분할 수 있으며 서버 이름은 시스템에 따라 일반적인 이름(generic name)을 이루므로 이름 공간에 저장된 프로세스의 이름을 두 종류의 프로세스 군으로 분리할 수 있어서 빠른 이름 엑세스를 가능하게 해준다.

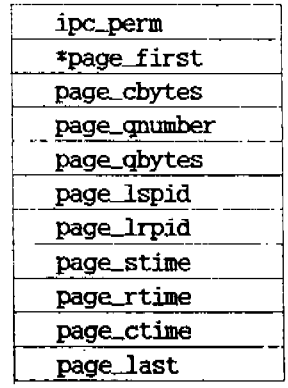
(그림 6)은 프로세스의 이름 구조를 나타낸다.

분산 운영체제에서 일반적인 페이지 패싱은 송신 프로세스의 주소 공간(address space)에서 수신 프로세스의 주소공간으로 페이지를 복사 함으로써 이루어진다. 이 때 프로세스간의 성공적 페이지 전송을 위해 요구되는 일련의 규칙들을 통신 프로토콜이라 하며 이러한 통신 규약은 시스템의 특성에 따라 설계자에 의해 선별 된다.

3.2 확장된 RPC 구조

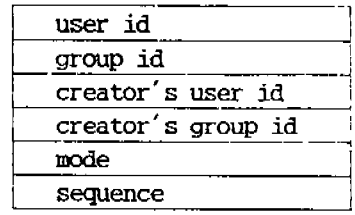
RPC기법을 통한 메세지는 프로세스간의 정보 교환을 위한 구조화된 정보이다. 메세지 구조에서 페이지를 어떠한 크기 형태로 둘 것인가와 큐에 어떠한 데이타 타입이 들어갈 것인가 하는 것이다. 페이지의 길이에 따라 고정길이, 가변길이, typed 페이지로 분류할 수 있으며 일반적으로 느슨하게 연결된 시스템 환경에서는 가변길이 페이지를 이용하는데 실제적인 구현은 복잡하지만, 프로그래밍이 간단하다.

그러나 본 논문에서는 구현상에 단순화를 위해 고정길이 페이지를 채택하였으며 페이지를 header 부분과 데이타 부분으로 나누어 다음과 같은 구성요소를 설계 하였다. 페이지 큐(page queue)의 자료구조는 다음과 같다.



(그림 7) 페이지 큐 자료구조
(Fig. 7) Page Queue Data Structure

페이지 큐에서는 ipc_perm과 큐의 처음과 끝 그리고 큐의 현재의 크기와 페이지 갯수, last pagesnd, last pagercv와 페이지의 송수신 시간과 교환 시간을 갖는다. 또한 ipc_perm의 자료구조는 (그림 8)과 같다.



(그림 8) ipc_perm 자료구조
(Fig. 8) ipc_perm Data Structure

여기에서는 user와 group의 자료구조를 제시하였다. 본 논문에서는 기존의 RPC를 이용한 시스템 호출에 메세지 방식의 큐를 채택하여 기존에 RPC가 가지고 있던 원격 프로시져를 호출한 프로세스의 블럭(block)을 유발하는 것 즉, 원격 프로시져에서 반환되지 않으면 다음 명령을 수행할 수 없는 점을 개선하였다. 호출된 순서에 의해 순차적으로 프로세스 할 수 있는 형태를 갖게 함으로써 현 RPC호출방식에 큐 구조를 첨가하여 고객이 시스템을 호출할 수 있는 알고리즘을 제시하였다.

아래의 (그림 9)은 RPC 방식의 각 함수들의 호출과 자료 구조를 나타냈다.

그리고 (그림 10)은 페이지 구조이며, (그림 11)은 ipc-perm 자료구조를 나타낸다. 각 변수의 역할은 (표 1)에 나타나 있다.

```
  struct page
     {
struct   page     *page_next;     /* ptr to next page on q */
long     page_type;               /* page type */
short    page_ts;                 /* page text size */
short    page_spot;               /* page text map address */
     };
```

(그림 9) 페이지 구조
(Fig. 9) Page Structure

```
struct msqid_ds
{
   struct         ipc_perm;        /* operation permission struct */
   struct  page    *page_first;    /* ptr to first page on q   */
   struct  page    *page_last;     /* ptr to last page on q    */
   ushort         page_cbytes;     /* current # bytes on q       */
   ushort         page_qnum;       /* # of page on q */
   ushort         page_qbytes;     /* max # of bytes on q */
   ushort         page_lspid;      /* pid of last pagesnd */
   ushort         page_lrpis;      /* pid of last pagercv */
   time_t         page_stime;      /* last pagesnd time */
   time_t         page_rtime;      /* last pagercv time */
   time_t         page_ctime;      /* last change time */
};
```

(그림 10) 페이지 큐 자료구조
(Fig. 10) Page Queue Data Structure

```
struct ipc_perm
     {
          ushort      uid;        /* owner's user id */
          ushort      gid;        /* owner's group id */
          ushort      cuid;       /* creator's user id */
          ushort      cgid;       /* creator's group id */
          ushort      mode;       /* access modes */
          ushort       seq;       /* slot usage sequence number */
          ushort       Key;       /* Key */
     };
```

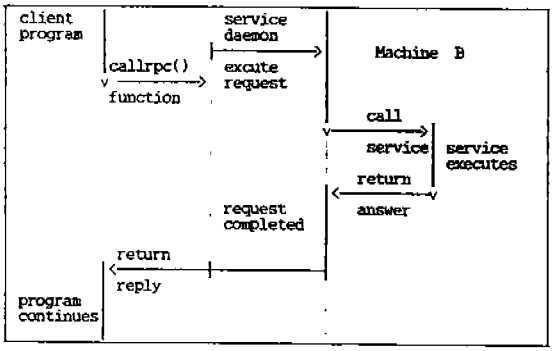
(그림 11) ipc_perm
(Fig. 11) ipc_perm data structure

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds  *buf;
```

(그림 12) RPC 제어 큐
(Fig. 12) Controlling RPC Queues

〈표 1〉 페이지 호출 변수
〈Table 1〉 Page call variable

호출 변수	변 수 기 능
key	요구되어진 key 값을 전송
opperm	요구된 작동 수락된 것을 저장
flags	요구된 제어 명령을 저장
opperm_flags	메세지 flags 인자를 전송하기 위한 시스템호출에 사용
msqid	시스템 호출이 되었을 때 메세지 큐 id의 수를 반환, 호출이 되지 않으면 에러코드 (−1)를 반환.



(그림 13) Remote Procedure Call의 통신 프로토콜
(Fig. 13) Communication protocol of RPC

3.2.1 페이지 관리 알고리즘

페이지는 슬롯 통신, 시스템제어기 입출력용으로 구분하여 페이지 프리 헤드(free head)는 할당 포인터와 반환 포인터로 구성되는데 할당 포인터에서 페이지 할당이 이루어지고 반환 포인터에 페이지 반환이 된다.

```
/* page manager algorithm */
 page assign()
{
   page free head create;
  loop:
      page assign for lock;
      if ( assign pointer := page)
{
        delete page(1);
         assign for unlock;
         initialize  deleted page;
         page return;
       } else
{
          page return for lock;
          if (return pointer:=page)
{
             return pointer <=> assign pointer;
             return ,assign for unlock;
             goto loop;
          }
          return ,assign for unlock;
          sleep call;
          goto loop;
          page return()
          {
            page free head create;
            page return for lock;
            return pointer page ;
            return for unlock;
            if (assign hold process)
                wakeup call;
          }
   }
}
```

(그림 14) 페이지 관리자 알고리즘
(Fig. 14) Page manager algorithm

3.2.2 페이지 전송 알고리즘

페이지 전송은 요구의 경우와 서비스의 완료의 경우가 달리 수행된다. 요구의 경우에는 목적 슬롯의 요구 채널에 페이지를 연결하여 목적 슬롯에 인터럽트를 전송하므로써 페이지 전송이 되며, 서비스 완료의 경우는 요구시에 접수한 페이지를 서비스 완료 또는 기타 사항으로 반송하는 페이지 전송으로서 원시 슬롯에 인터럽트를 전송하므로 페이지가 전송되는 것을 말한다. 페이지 전송은 하나의 페이지마다 대상 슬롯에 전송하는 것이 아니고 일정 갯수 또는 일정 시간 등의 변수에 의하여 여러개의 페이지를 묶어서

하나의 인터럽트로 전송하는 기법을 사용한다. 그러나 실시간 처리를 위한 페이지 전송이 요구 되면 그 페이지를 채널의 처음 위치에 연결하는 즉시 대상 슬롯에 전송한다.

3.2.3 페이지 송수신 알고리즘

페이지 수신은 요구에 대한 수신과 서비스 완료에 의한 수신이 있다. 요구에 대한 수신 과정은 요구 채널에 연결된 페이지를 떼어와서 전처리 하고 명령 큐에 삽입한다. 페이지 수신은 한꺼번에 채널에 연결된 페이지를 떼어와서 전처리를 하고 명령큐에 삽입한다. 페이지 수신은 한꺼번에 채널에 연결된 페이지를 떼어와서 지역 메모리에 가져온 후 접속된 순서대로 수신하는 기법을 사용한다. 다중 채널에서 각 인터럽트 벡터는 채널 헤드에 기록된 요구 벡터가 슬롯 통신에 대한 채널 헤드에 기록된 요구 벡터가 슬롯 통신에 대한 벡터이고, 그 다음의 입출력 장치 채널들은 하나씩 증가하는 벡터값을 가지므로 수신시 입력되는 벡터의 offset은 지역 메모리의 채널 위치와 일치됨으로서 수신 할 페이지를 쉽게 가져올 수 있다. 서비스 완료에 의한 수신 과정도 요구에 대한 수신과 유사한 방법으로 시행되지만 프로세스의 특성에 따라 다소 차이가 있다.

```
/ * page transfer algorithm */
 page send timer()
{
   channel lock;
   if (channel= page )
   {
      channel unlock;
      interrupt transfer;
   }
  page send()
  {
    channel head create;
    page types for channel create;
    channel unlock;
    channel page insert;
    channel page counter increse;
    channel unlock;
    if (realtime page || transmit page numbers)
     {
       if (timer active)
          timer active out;
        interrupt transfer;
      }
      if (timer not active)
         timer active ;
  }
}
```

(그림 15) 페이지 송신 알고리즘
(Fig. 15) Page send algorithm

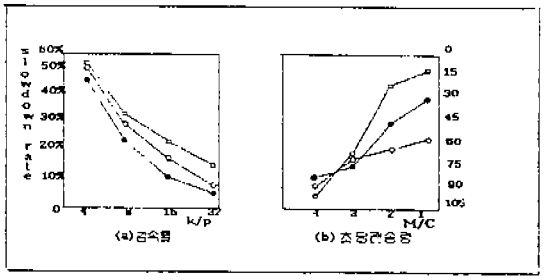
```
page recieve()
 {
    channel head create;
    vector number for channel decision;
    channel unlock;
    page delete pointer of channel <= tmp;
    page field of channel initialize;
    channel unlock;
    for
     {
       a queue insert page ;
     }
}
```

(그림 16) 페이지 수신 알고리즘
(Fig. 16) Page recive algorithm

4. 성능분석

본 절에서는 메세지 방식의 큐를 RPC에 이용하여 기존의 원격 호출방식과 메세지 방식을 마이크로 벤치마크 테스트로 실험하였다. (그림 17)은 메세지 방식과 기존 RPC방식과 큐(Queue)를 이용한 QRPC 방식의 성능 평가 테이블과 그래프이다. 먼저 성능 평가 테이블을 보면 테이블 (a)는 페이지를 이용하여 메세지 전송과 원격프로시져 호출의 통신 감속율을 비교한 테이블이다. 테이블에서도 나타나 있듯이 페이지의 단위를 높일 수록 통신 감속율은 향상 되어지는 것을 볼수 있다. 같은 용량의 페이지 단위에서도 큐를 사용한 RPC방식이 무한 대기 상태에 관계 없이 다음 데이타(page)를 전송하므로 메세지 전송 방식과 기존 RPC 방식보다 통신 감속율이 향상되어 진 것을 볼 수 있다. 또한 테이블 (b)는 캐쉬 메모리를 사용하여 초당 전송량을 평가한 테이블이다. 이 성능 평가에서 캐쉬 메모리의 크기는 Sprite Network System에서 사용한 크기인 1M, 2M, 3M, 4Mega Byte를 사용하여 평가 하였으며 테이블에서 나타나 있듯이 기존 RPC 방식보다 12%~14% 향상된 것을 볼 수 있다. 위의 두 테이블에 나타난 결과는 아래 그래프에 나타나 있다. 위의 성능 평가 도구로는 마이크로 컴퓨터의 벤치마크 시스템을 이용하였으며, 그래프 (a)의 가로 축은 전송에 사용된 페이지의 크기를 나타내며 세로축은 slowdown rate를 나타낸다. 그래프 (b)의 가로축은 캐쉬 메모리의 크기를 나타내며, 세로축은 초당 전송되는 크기를 나타낸다. 그래프에 나타난 것처럼 캐쉬 메모리의 크기를 변형하여 각 단계별로 측정하였으며, 또한 기존 RPC방식과 메세지 방식의 큐를

이용한 원격 프로시쥬어 호출방식을 평가한 그래프이다. 그래프에서 나타난 것처럼 전송되는 정보를 페이지 단위로 분할하여 캐쉬를 이용하여 큐 형태로 전송하므로 10%~15%의 성능향상을 가졌왔다. 큐를 이용하여 원격 프로시져 호출을 수행하므로 기존 원격 프로시져 호출 방식의 단점인 무한정 대기 상태가 발생하지 않으므로 호출 수행 속도는 향상되었다.



page size / 방 식	4K	8K	16K	32K
message	53%	32%	21%	15%
RPC	50%	28%	16%	9%
QRPC	43%	21%	10%	6%

(a) 감속률

Cache size / 방 식	1M	2M	3M	4M
message	13	34	71	97
RPC	60	67	82	91
QRPC	33	48	80	82

(b) 캐쉬 메모리 사용시 초당 전송율

(그림 17) 성능 평가도
(Fig. 17) Performance Evaluation grap.

5. 결 론

RISC 방식의 MIPS에 SUN 워크스테이션과 PC를 ethernet로 연결된 것을 이용하여 본 논문에서는 UNIX system V의 기존 RPC 호출 방식의 단점, 즉 원격 프로시쥬어를 호출한 후에는 반드시 반환 하여야만 다음 명령을 수행할 수 있기 때문에 반환이 이루어지지 않으면 무한정 대기하는 현상이 발생한다. 이 단점을 해결하기 위해 본 논문에서는 큐를 이용한 메세지 전송 방식으로 보안하여 위에서 제시한 문제점을 해결하여 순차적으로 프로세싱할 수 있는 형태로 변경된 RPC 알고리즘을 설계 및 구현하였다. 벤치마크 시스템을 이용하여 그래프에 나타난 것처럼 캐쉬 메모리를 변형하여 각 단계별로 측정하였으며, 또한 기존 RPC방식과 메세지 방식의 큐를 이용한 원격 프로시쥬어 호출방식을 평가한 그래프에서 나타난 것처럼 전송되는 정보를 페이지 단위로 분할하여 캐쉬를 이용한 큐 형태로 전송하므로 10%~15%의 성능향상을 가졌왔다.

끝으로 앞으로의 연구방향을 본고에서 설계된 것을 바탕으로 마이크로 컴퓨터, 퍼스널 컴퓨터, 워크스테이션을 연결하여 화일을 각각의 노드에 분산 시켜 완전한 분산 화일 시스템을 구현시키는 것과 캐쉬 블록 관리에서 발생하는 부하 균형과 서버 시스템의 고장에 대한 복구성, 교착상태 발생 문제에 대해서 더 많은 실험과 연구가 필요하다.

참 고 문 헌

[1] George.A.champine, "Distributed computer System" north-Holland, 1980.

[2] B.W.Lampson et al., "Distributed Systems Architecture and Implementation" Springer-Verlag, 1981.

[3] P.H.Enslow Jr., "What is a Distributed Data Processing System?", IEEE Computer, no. v1, 1981. 1.

[4] A.S and R.V.Renessee, "Distributed Operating System", ACM Computing Surveys Vol. 17, No. 4, 1985. 12.

[5] Yeshayahu Arsty, Hung-Yang Chang, "Interprocess Communication in the Charlotte", IEEE, 1987. 2.

[6] Sape Mulleder, "Distributed Systems", ACM Press, 1989.

[7] Coulouris and Doullimore, "Distributed systems: concepts and Design", Addison Wesley, 1990.

[8] Stephen G.Kocha, Patrick H. Wood, "Unix Networking", Pipeline Associates, Inc, 1989.

[9] Andrew D.Birrell and Bruce Jay Nelson, "Implementing Remote Procedure Calls", ACM

Trans, on Computer System, Vol 2. No. 1. Feb., pp. 39-59, 1984.

[10] E.Levy and A. Silberschatz. "Distributed File System: Concepts and Examples", ACM Computing Surveys, Vol. 22, NO. 4, Dec 1990, pp 321-374.

[11] G.Dickson, A.Lioyd, "Open Systems Interconnection", Prentice-Hall, 1992.

[12] R.Sandberg, "The Sun Network File system: Design, Implementation and Experience", Networks File Systems, USENIX Conference, UNIV. of Califonia Press, Summer 1987.

[13] Y.Parker. J-P. Banatre and M.Bozyigit, "Distributed Operating system: theory and Practice", Springer-Verlag, pp. 147-174, 1986.

[14] David R.Cheriton, "The V Distributed system", Communication of the ACM, Vol. 31, March, pp. 314-333, 1988.

[15] David K. Gifford and Nathan Glasser, "Remote pipes and procedure for efficient distributed communication", ACM Trans. on Computer Systems, Vol. 6, August, pp. 259-283, 1988.

이 병 관

1979년 부산대학교 졸업(학사)
1986년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1990년 중앙대학교 대학원 전자계산학과 졸업(이학박사)
1988년~현재 관동대학교 전자계산공학과 부교수.
관심분야: 소프트웨어 공학, 분산운영체제, multi-media

윤 동 식

1992년 관동대학교 정보처리학과 졸업(학사)
1994년 관동대학교 대학원 전자계산공학과 졸업(공학석사)
1994년~현재 원주전문대학, 동우전문대학, 삼척산업대학, 관동대학교 외래강사
1995년~현재 아세아항공 전문학교 정보통신 설비공학과 주임교수.
관심분야: 분산운영체제, 멀티미디어, 컴퓨터 네트워크