분산시스템 지원을 위한 그룹 RPC프로토콜의 설계와 검증

이 종 근[†] · 이 창 석[††] · 이 광 휘[†††]

요 약

RPC는 분산시스템에서 가장 많이 활용되고있는 통신메카니즘중의 하나이다. 본 연구에서는 동적 그룹 지원(Hierarchical Group Communication)을 이용하여 RPC의 효율성을 높히는 HGRPC(Hierarchical Group Remote Procedure Call)를 제안한다. HGRPC에서는 효율적인 동적 그룹 지원관리를 활용하여 call 프로세듀를 활용하기위하여 클라이언트 서버에 MonP(Monitor Process)를 두어 글로벌 서버의 상태를 모니터하여 글로벌 서버의 기능을 효율화 시켰으며 로칼 서버는 로칼 서버의 멤버들을 관리하므로 글로벌 서버와의 불필요한 통신부하를 효율적으로 관리하게하였다. 또한 제안시스템의 검증을 위하여 페트리 네트를 이용하여 그 성질들을 분석하고 효율성에 대하여 분석하였다.

Design and Verification of a Group RPC Protocol for A Support Distributed System

Jong-Kun Lee[†] · Chang-Seuk Lee[††] · Kwang-Hui Lee[†††]

ABSTRACT

RPC(Remote Procedure Call) is one of the very popular protocol in distributed application design. In this paper, we are proposed the HGRPC(Hierarchical Group Remote Procedure Call) based on the Hierarchical Group Communication management for improve the userability of RPC. Also, in HGRPC, an architecture of a monitor processor(MonP) is presented for distributed network management system managing communication network for distributed systems, specially for overlapping domain management. Based on these MonP and hierachoical communication managemnet concepts, we can reduce the communication load on communication network. Our proposed protocol has been verified by the Petri nets.

1. 서 론

분산시스템이란 여러개의 컴퓨터 환경과 통신망 속에서 시스템과 응용프로그램들이 분산되어 제반 자원의 공유와 제어를 행하는 시스템 형태로 널리 이용되고있다. 그러나 이러한 분산시스템의 환경에서는 자원의 공유나 제어를 통한 시스템의 성능을 향상시키는 것에 반하여 분산된 중복 데이타 처리에 따른 통신 처리의 과부하와 효율적인 통신 메카니즘의 활용이 주요한 문제점으로 대두 되고있다. 이러한 문제점을 해결하기 위하여 원격 프로시저 호출(RPC:Remote Procedure Call)이 제시되었는데[1], RPC는 분산처리 시스템과 네트워크 사이를 프로세스간의 통신 메카니

※이 연구는 1996년도 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음.

† 종신회원:창원대학교 전자계산학과 교수
†† 준 회 원:창원대학교 대학원생
††† 정 회 원:창원대학교 전자계산학과 교수

논문접수:1997년 9월 27일, 심사완료:1997년 12월 29일

증화하여 통신망의 과부하를 줄이고 단순성(simplicity), 투명성(transparency)의 특성으로 분산시스템 개발에 광범위하게 이용되고 있다. RPC 에서는 클라이언트에서 서버프로세서에로의 서비스 요구를 마치 클라이언트에서 일반적인 호출(Call) 프로세듀와 같이 동시적인 개념으로 구성되는 특성을 갖고있다. 이러한 RPC시스템으로는 SUN RPC와 XEROX RPC등이 있으며, 분산 환경에서는 OSFDCE의 RPC가 많이 사용되어지고 있다[2, 3, 4].

원격 프로시저 호출(Remote Procedure Call) 메카니즘은 호출자와 피호출자가 서로 다른(혹은 같은) 컴퓨터에 위치한 다른 프로세스간에 이루어 진다. 즉 호출당한 파라메터와 이에대한 회답 결과는 통신망을 통하여 호출자와 피호출자 사이에 교신된다. 지금까지의 RPC는 단순히 점대점(point-to-point)이나 일괄송신(broadcasting) 통신에 응용되고 있으며 그룹으로 형성된 통신체제에서는 응용되지 못하고 있다. 그동안 RPC에 그룹통신개념을 부가한 GRPC(Group Remote Procedure Call)에 관한 연구가 여러 학자들을 통하여 진행되어왔는데 특히 호출자에서 통신상의 장해가 발생하여도 피호출자의 경우 이에 대한 정보가 없어 피호출자가 계속해서 작업을 수행하는 에라복구 문제를 중심으로 이루어져 왔다[11, 17, 18]. 그러나 결과 처리에 대하여 호출된 모든 결과 중에서 어느 하나만 성공적으로 처리가 되어 수신이 되면 나머지는 무시하는 것이나 일단 통신망상태에서는 계속해서 호출된 결과가 글로발 서버에게로 수신되고 있는 것이다. 또한 실패인 경우 모든 결과가 실패일 경우에 한하므로 역시 클라이언트에서 서버의 모든 결과를 수신함으로 많은 통신 부하를 부담하게 한다.

지금까지 이러한 통신망에서의 부하를 관리하는 GRPC연구는 연구자의 견해로는 아직 발견하지 못하였다. 따라서 본 연구에서는 이러한 통신부하의 문제점을 해결하기 위하여 동적 그룹 지원 관리하는 효율적인 그룹RPC(HGRPC: Hierarchical Group Remote Procedure Call)를 설계 제안 한다. 동적 그룹지원[6]이란 클라이언트를 글로벌 서버로 간주하여 네트워크 전반의 제반 내용을 관리하고 각 로칼서버는 자신의 멤버들만을 관리하므로써 클라이언트에게 불필요한 송수신의 부하를 차단하므로 전체 통신망의 통신부하를 조절하도록 하는 방법이다[5, 6, 7, 16, 18]. 즉 로칼서버가 처리 결과들을 수거하여 성공과 실패에 따른 결과를 글로벌서버에게 보냄으로써 전체통신망의 부하를 효율화 시킬 수가 있다. 또한 HGRPC 프로토콜은 멀티캐스트(multicast)를 이용하거나 점대점 통신(point-to-point communication)을 이용하여 하나이상의 서버에게 서비스를 요청을 할 수 있으며 전체적인 시스템의 신뢰성(reliability)과 효율성(facility)을 향상시킬 수가 있다. 본 연구에서는 이 기종(heterogeneous) 컴퓨터간의 데이타 교환을 위한 데이타의 코딩과 표현을 위하여 표준인 SUN의 XDR(External Data Representation) 을 사용하여 표현한다[9]. 또한 HGRPC의 설계에 대한 검증을 위하여 패트리 네트를 이용하여 그 정확성을 검증한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 관련 연구에 대하여 살펴보고, 3장에서는 효율적인 HGRPC의 설계새념을 살펴보고 4장에서는 효율적인 그룹 원격 프로시저 호출(HRPC)을 설계한다. 그리고, 5장에서 앞 장에서 제시한 개념과 설계 내용을 토대로 HGRPC의 프로토타입(prototype)을 정의하고, 6장에서는 패트리 네트를 이용하여 HGRPC모델을 분석하여 그 정확성을 검증하였고, 7장에서 향후 연구 계획과 결론을 맺는다.

2. 관련 연구

GRPC메카니즘은 간단히 정의 한다면 RPC에 그룹개념을 첨가한 것으로 이러한 처리를 효율적으로 수행하기 위하여 지금까지 연구되어진 방법들을 살펴본다면 다음과 같다.

1) 순환RPC[18]

송수신중에 장애가 발생하면 단원(troup)이라고 명명하는 하나의 일반적인 과정을 두어 이를 적극 활용함으로 효율적으로 장애가 발생하였을 경우 극복이 가능하다.

2) 세분 프로토콜(macro protocol)[17]

RPC의 모든 기능들을 세분프로토콜(macro protocol)화 하여 이러한 기능들의 관계를 조절하여 많은 프로토콜들이 수행되도록 설계된 것으로 설계에는 용이한 방법이나 각 프로토콜간의 관계를 정리하고 관리하여야 하는 결점이 있다.

3)독립실패(Independent failure) 극복 GRPC[11]

GRPC의 기능중에서 어느 한 쪽에 장애가 발생 할 경우 다른 지역에게는 그 사실이 알려지지 못함에 착안하여 이러한 독립실패(Independent failure)를 극복하고자 설계된 개념이다.

본 연구에서는 GRPC구성에서 갖는 통신량의 최소화하는데 연구의 목적이 있다. 따라서 이미 본 연구자에의하여 발표되어졌던 동적그룹지원관리시스템의 주요 결과[6, 7, 16]들을 이용하여 효율적으로 통신량을 조절할 수있는 HGRPC프로토콜을 설계하고자 하며 고아프로세스 처리에 대하여서는 발견하는대로 파괴하는 방법을 사용한다.

3. 효율적인 Group RPC(HGRPC) 프로토콜 설계 개념

3.1 동적그룹지원 관리 개념

본 연구에서 설계하는 HGRPC는 효율적인 동적그룹지원관리를 통하여 통신망에서의 통신부하를 효율적으로 조절하는데 있다. 이러한 동적 그룹지원 관리를 위하여서는 전체 바인더(Global Binder)는 전체 서버 그룹(GSG)들에 속하는 모든 서버들의 정보들을 관리하며 지역 바인더(Local Binder)는 각각의 서버 그룹에 속하는 서버들에 대한 정보들을 관리한다. GRPC 그룹 정보들을 제공하는 전체 바인더(Global Binder)에 장애가 발생하였을 경우에는 GRPC 시스템에 치명적일 수 있다. 이러한 장애의 극복을 위하여 전체 바인더(Global Binder)의 또 다른 복사본인 모니터 프로세스를 두개 한다. 만약 통신 중에 전체 바인더(Global Binder)에 장애가 발생하게 되면, 모니터 프로세스가 그런 상황을 파악하여 새로운 전체 바인더(Global Binder)을 위한 프로세스를 생성하고 자신이 가지고 있던 가장 최신의 정보를 새로 생성된 전체 바인더(Global Binder)의 현재 상태로 복사함으로써 아무런 손실없이 장애가 발생한 전체 바인더(Global Binder)를 복구할 수 있다[10].

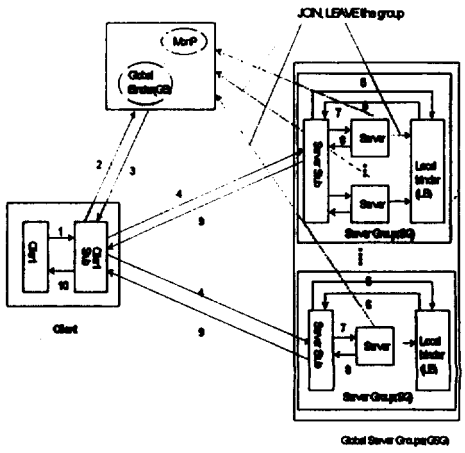
클라이언트 스터브에서 보낸 원격 서비스 요청이 일정한 시간이 지나도록 응답이 없을 경우에는 응답이 없는 서버에 대한 조사 요청을 하여 해당 서버의 등록 여부를 결정 관리한다.

3.2 효율적인 결과 처리

각각의 서버 그룹(SG)에서 반환되는 GRPC의 호출에 대한 결과를 클라이언트에서 처리하는 방법에 따라 신뢰성과 처리 시간이 달라진다. 즉, 호출된 서버들로부터 처리되는 결과를 많이 받을수록 GRPC 호출은 신뢰성이 높아질 것이고 반면에 처리 시간은 길어질 것이다. 높은 신뢰성을 요구하는 분산 시스템일 경우는 GRPC의 호출 결과를 많이 채택하도록 설계한다. 효율적인 결과 처리를 위하여 로칼서버는 해당 멤버들의 성공적인 처리와 실패를 파악하여 글로발서버에게 성공과 실패의 결과만을 송신함으로 글로벌서버에게 집중되는 통신부하를 효율화 시킬 수 있다.

4. 효율적인 Group RPC(HGRPC) 프로토콜 설계

HGRPC는 RPC에 그룹통신 개념을 첨가하여 효율적인 통신망 관리를 목적으로 한다. 본 연구에서의 HGRPC의 구성은 HGRPC를 호출(request)하는 클라이언트(client)와 호출에 대한 응답(response)을 하는 서버들(servers)로 구성된다. 각각의 서버들은 사용자의 요구에 의해 서버 그룹(Server Groups)들을 형성한다. 서버들에 대한 관리 정보(management information)들은 두가지 형태로 나누어져 관리가 된다. 각각의



(그림 1) 그룹 RPC의 논리적 구조
(Fig. 1) Config of GRPC

서버 그룹(Server Groups)들의 관리 정보들은 전체 바인더(Global Binder)에서 관리되고 각 서버 그룹에 속하는 각각의 서버들(Servers)의 관리 정보는 지역 바인더(Local Binder)에서 관리된다.(그림 1)

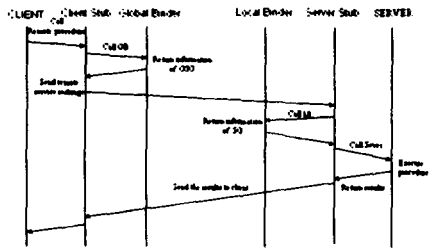
이 장에서는 HGRPC의 동작 메커니즘, HGRPC의 그룹 관리, 결과 처리, 순서 관계와 고아 프로세스의 처리 순서로 HGRPC를 설계한다.

4.1 HGRPC의 동작

일반적으로 HGRPC의 동작메커니즘은 다음과 같은 10단계의 과정을 거쳐 시행되며, 이 때의 메시지 흐름은 (그림 3)으로 나타난다.

1)클라이언트는 원격 프로시저 호출(Remote Procedure Call)을 한다.

2)클라이언트 스터브(client stub)는 호출될 GSG (Global Server Groups)에 대한 정보를 얻기 위해 XDR 데이타형으로 변환한 네트워크 메세지(network messages)를 전체 바인더(Global Binder)에게 전송한다.



(그림 2) GRPC 메커니즘의 메시지 흐름
(Fig. 2) A message flow in GRPC

3)전체 바인더(Global Binder)는 클라이언트 스터브가 요청한 XDR 데이타형의 정보를 다시 전체 바인더(Global Binder)의 데이타형으로 변환한다. 그리고 클라이언트 스터브의 요청이 적당한 가를 검사한다. 그리고나서 호출될 GSG의 멤버쉽 정보(membership infor-mation)를 전송할 메세지에 첨가하여 클라이언트 스터브에게 전송한다.

4)호출될 GSG의 멤버쉽 정보를 받은 클라이언트 스터브는 처리될 데이타뿐만 아니라 서버들에 관련된 호출 파라메터들을 전송될 메세지에 marshal하여 XDR로 변환한다. 그리고 호출된 서버그룹들의 각각의 서버 스터브에게 원격 서비스 요청 메세지를 전송한다.

5)원격 서비스 요청을 받은 각각의 서버 스터브는 XDR로 전송된 메세지를 각각의 서버 데이타형으로 변환한다. 그리고 호출 파라메터들을 해석하고 지역 바인더(Local Binder)에게 서버 그룹(Sserver Group)에 대한 정보를 요청한다.

6)지역 바인더(Local Binder)는 필요한 서버그룹들의 각각의 서버 정보를 서버 스터브에게 전송한다.

7)이러한 정보들을 가지고 서버 스터브는 클라이언트에서 요청한 프로시저를 각각의 서버에서 실행한다.

8)서버에서 실행된 프로시저의 결과는 반환되는 즉시 서버 스터브에게 전송된다.

9)서버 스터브는 실행된 결과를 marshal하여 다시 XDR로 변환하고 원격 서비스를 요청한 클라이언트 스터브에게 전송한다.

10)클라이언트 스터브는 반환된 실행 결과들을 모아서 다시 클라이언트 데이타형으로 변환하여GRPC 호출의 결과를 클라이언트에게 보낸다.

위의 동작과정을 거쳐서 정상적인 GRPC호출이 이루어진다.

4.2 HGRPC의 그룹 관리

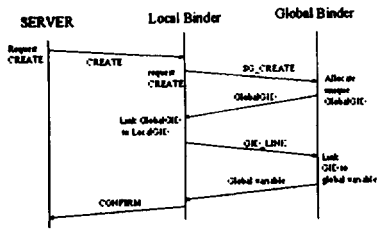
HGRPC의 그룹관리 프리미티브(primitives)는 서버 그룹들을 관리하기 위하여 CREATE, DESTROY, JOIN, 그리고 LEAVE 메시지를 사용한다.

1)CREATE

그룹을 생성하기 위하여 사용되어지며, 새로운 그룹을 만든 프로세스는 자연히 그 그룹의 멤버가 된다. 여기서 사용되어지는 GroupID는 글로벌 서버에게서 할당되어지며 그 이름은 전체 시스템 상에서 유일하다. 서버 그룹이 존재하지 않을 때, 새로운 HGRPC의 서버 그룹을 생성한다.

처음 등록하고자 하는 프로세스가 로칼서버에게 create메세지를 보내면 로칼서버가 글로벌 서버에게 ID생성을 요청한다. 글로벌 서버는 새로운 ID를 할당하고 새로운 그룹테이블을 생성하고 GroupID요청

한 로컬서버를 그룹테이블에 저장한 후 모든 로컬서버에게 새로운 그룹의 생성을 GroupID와 함께 알린다. 메세지를 수신한 로칼서버는 새로운 그룹테이블과 멤버테이블을 생성한다. 글로벌서버에게서 할당받은 GroupID와 멤버 테이블의 첫주소를 그룹테이블에 저장한다. create요청을 한 프로세스를 로칼서버내에 존재하는 멤버 테이블에 저장한다. 로컬서버는 GroupID의 링크를 글로벌서버에게 요청하고 글로벌서버는 이를 시행한후 그 결과를 로칼서버에게 전송함으로 새로운 프로세스를 생성하게 된다(그림 4).



(그림 3) 새로운 서버 그룹의 생성
(Fig. 3) Creation a new server group

2) DESTROY

서버 그룹을 탈퇴할 때 사용되며, 서버 그룹의 마지막 멤버이면 그 그룹은 파괴는데 과정은 create와 동일하다.

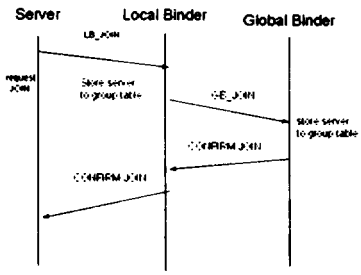
3) JOIN

서버 그룹에 참여하기 위해서는 다음의 두 단계로 구성되어 진다. 먼저 지역 바인더(LB)에게 JOIN 메시지를 보내면 지역 바인더(LB)가 전체 바인더(GB)에게 JOIN 메시지를 보낸다. (그림 5)는 JOIN 프리미티브를 나타낸다.

4) LEAVE

서버그룹을 탈퇴할 경우에 사용되며 과정은 JOIN 과정의 역순으로 동작이 이루어진다.

위와 같은 방법으로 그룹 관리 프리미티브가 동작하므로 그룹의 무결성(integrity of group view)이 유



(그림 4) JOIN 프리미티브의 동작
(Fig. 4) Join primitive

지된다.

HGRPC 그룹 정보들을 제공하는 전체 바인더(Global Binder)에 장애가 발생하였을 경우에는 HGRPC 시스템에 치명적일 수 있다. 따라서 장애의 극복을 위하여 전체 바인더(Global Binder)를 감지하는 전체 바인더(Global Binder)의 또 다른 복사본인 모니터 프로세스를 두게 여 전체 바인더의 장애 발생시 보다 유연하게 대처 할 수 있다. 전체 바인더(Global Binder)는 전체 서버그룹(GSG)의 내용이 변할 때, 변화된 상태를 모니터 프로세스에게 보내게 된다. 이를 받은 모니터 프로세스는 가장 최신의 정보를 유지한다.

만약 통신 중에 전체 바인더(Global Binder)에 장애가 발생하게 되면, 모니터 프로세스가 그런 상황을 파악하여 새로운 전체 바인더(Global Binder)을 위한 프로세스를 생성하고 자신이 가지고 있던 가장 최신의 정보를 새로 생성된 전체 바인더(Global Binder)의 현재 상태로 복사함으로써 아무런 손실없이 장애가 발생한 전체 바인더(Global Binder)를 복구할 수 있다[10].

클라이언트 스터브에서 보낸 원격 서비스 요청이 일정한 시간이 지나도록 응답이 없을 경우에는 응답이 없는 서버에 대한 조사 요청을 전체 바인더(Global Binder)에게 한다. 그러면 전체 바인더(Global Binder)는 의심나는 서버에게 원격 서비스를 요청하게 된다. 일정한 시간동안 응답이 있으면 전체 바인더(Global Binder)는 응답을 다시 클라이언트 스터브에게 보내게 되고 일정한 시간동안 응답이 없을 경우에는 전체 바인더(Global Binder)는 서버가 동작하지 않는 것으로 간주한다. 그리고, 서버 그룹에서 응답이 없는 서버를 제외시킨다. 만약 응답이 없는 서버가 장애로부터 복구가 이루어지면 다시 참여 메세지를 통하여 서

버 그룹에 참여할 수 있다.

4.3 GRPC의 결과 처리

각각의 서버 그룹(SG)에서 반환되는 GRPC의 호출에 대한 결과를 클라이언트에서 처리하는 방법에 따라 신뢰성과 처리 시간이 달라진다. 즉, 호출된 서버들로부터 처리되는 결과를 많이 받을수록 GRPC 호출은 신뢰성이 높아질 것이고 반면에 처리 시간은 길어질 것이다. 높은 신뢰성을 요구하는 분산 시스템일 경우는 GRPC의 호출 결과를 많이 채택하도록 설계한다. 처리 시간을 줄여서 분산 시스템의 전체 처리 속도를 높이고 싶을 경우는 제일 먼저 서버에서 처리되어 결과로 반환되는 값을 채택하도록 설계하면 된다.

본 논문에서는 GRPC의 결과 처리를 사용자로 하여금 선택할 수 있도록 설계한다. 즉, 파라메터의 값으로 결과 처리를 하도록 한다. 예를 들면, 파라메터가 1이면 가장 먼저 도착하는 결과값을 채택하겠다는 것을 말하며 처리 시간을 최소화 한다.

4.4 순서 관계(ordering)

GRPC 시스템에서는 메시지 전달방식에 의해서 서로 통신을 할 수 있으므로 일관성있는 작업을 위해서 메시지의 순서화된 수신이 필수적이다. 그래서, 본 논문에서는 다음과 같이 순서관계를 유지하여 메시지의 일관성을 유지한다.

전체 바인더(Global Binder)는 호출된 GSG의 멤버쉽 정보를을 클라이언트 스터브에게 반환할 때, 전체 바인더(Global Binder)는 GRPC호출에 대한 유일한 작업 식별자(unique transaction identifier)를 할당한다. 그러면 클라이언트 스터브는 할당된 작업 식별자를 이용하여 원격 서비스 요청 메세지를 만든다. 여기서 작업 식별자는 논리적 타임스탬프를 말한다. 이 요청 메세지가 호출된 GSG의 서버 스터브들에 도착하면 각각의 서버 스터브는 작업 식별자를 이용하여 스케쥴을 하고 서버 프로시저를 실행한다. 호출된 서버 프로시저의 스케쥴은 수신된 원격 서비스 메세지의 작업 식별자에 대한 순차적 순서에 의해서 이루어진다. 순서 관계를 정하는 방법은 다음과 같다. 먼저 기간이 지난 원격 서비스 요청 메세지는 버려지고 메세지들은 큐(queue)에 저장된다. 원격 서비스 요청 메세지는 한번 실행된다(exactly once). 이렇게 함으로써 호출된 GSG의 순서 관계를 유지한다.

4.5 Orphan process 처리

서버에서 클라이언트로부터 요청한 서비스를 처리하고 있을 때, 만일 클라이언트에서 장애가 발생한다면 고아 프로세스(Orphan process)가 발생하게 된다. 고아 프로세스는 컴퓨터의 자원을 낭비할 뿐 아니라, 클라이언트로부터의 새로운 서비스 요청을 방해하게 된다. 예를 든다면, 클라이언트에서 서버에게 서비스 요청을 하고나서 클라이언트에서 장애가 발생하여 장애를 복구한 다음, 다시 서버에게 서비스 요청을 보낸다면 이미 서버에서는 이전에 요청한 서비스가 처리되고 있다.

본 논문에서는 생성된 고아 프로세스를 파괴함으로써 고아 프로세스를 관리한다. 즉, 고아 프로세스를 발견하는 즉시 고아 프로세스를 파괴시킨다. 고아 프로세스의 검출은 두 가지 방법으로 구현될 수 있다. 하나는 장애로부터 복구된 클라이언트로부터 새로운 메시지가 생성될 때, 이전의 클라이언트는 없다는 것을 메시지에 포함시킴으로써 서버에서 고아 프로세스를 검출할 수 있다. 또 다른 방법은 클라이언트를 주기적으로 검사함으로써 서 고아 프로세스를 검출할 수 있다.

5. HGRPC의 프로토타입(Prototype) 정의

이 장에서는 4장에서의 설계 개념을 활용하여 객체지향적인 접근을 위하여 C++를 이용하여 HGRPC의 프로토타입을 정의한다.

5.1 XDR layer

5.1.1 XDR 스트림(streams)

XDR 스트림의 중요한 두가지 형:

XDRIn:불연속적인 데이터를 입력하는 입력 스트림.

XDROut:연속적인 데이터를 출력하는 출력 스트림.

5.1.2 XDR 기본 데이터형

XDR 프로토콜은 이기종의 데이터형을 어떻게 일반적인 데이터형으로 표현할 것인가를 정의한다. 그래서 기본적인 데이터형의 정의가 필요하다. XDR

기본 데이터형은 C언어의 데이터형과 유사하며 서로 호환 가능하다. 실제로 XDR은 더미 클래스(dummy class)이며 GRPC 클래스의 부모 클래스이다.

XDR::Boolean boolean형.

XDR::Integer 32비트의 signed integer.

XDR::UnsignedInteger unsigned integer.

XDR::Float 32비트의 floating point.

XDR::DoubleFloat 64비트의 floating point.

XDR::HyperInteger 64비트의 long signed integer.

XDR::UnsignedHyperInteger 64비트의 long unsigned integer.

XDR::Char 표준 문자형

XDR::Byte 8비트의 byte 표준형

XDR::Opaque Byte와 같지만, 생(raw) 데이터를 가르킴. 크기는 8비트이지만, 중요한 것은 opaque배열에는 어떤 데이터형으로도 선언이 가능.

XDROpaqueBlock 다양한 크기의 Opaque배열을 정의하는 클래스

XDRString 다양한 크기의 C string을 정의하는 클래스. 스트링은 널 문자(\0)로 끝난다.

5.2 GRPC layer

GRPC layer는 전송할 데이터를 변환하는 XDR layer를 이용하여 GRPC call을 위한 필요한 메커니즘을 제공한다.

5.2.1 GRPC 클래스와 데이터형(types)

GRPC라는 클래스를 정의하고 XDR 클래스로부터 모든 데이터형들을 상속 받는다.

GRPC::ID XDR::UnsignedInteger형이고 GRPC 프로토콜내에서 식별자(identifier)로 사용된다.

GRPC::ProgramID "프로그램ID"를 식별하기위한 상수

5.2.2 GRPCCall 클래스

클라이언트에 의해서 요청되는 서비스를 나타내는 객체들(objects)의 기본 클래스이다. 이 클래스의 호출에 의해서 데이터를 주고 받는다.

XDR::Boolean XDRQuery(XDROut) 클라이언트에서 호출 입력 인수들(parameters)을 연속(serial)되게 하는 XDR 필터(filter)이다. 입력 인수들을 변환하기 위해서는 각각 인수의 XDR 필터를 차례대로 호출한다.

XDR::Boolean XDRReply(XDRIn) 클라이언트에서 호출 출력 인수들을 불연속(unserial)되게 하는 XDR 필터이다. 출력 인수들을 변환하기 위해서는 각각 인수의 XDR 필터를 차례대로 호출한다.

XDR::Boolean XDRQuery(XDRIn) 서비스를 위한 호출 입력 인수들을 불연속되게 하는 XDR필터이다. 입력 인수들을 변환하기 위해서는 각각 인수의 XDR 필터를 차례대로 호출한다. 일반적으로 대칭 필터와 동일하다.

XDR::Boolean XDRReply(XDROut) 서비스를 위한 호출 출력 인수들을 연속되게 하는 XDR필터이다. 출력 인수들을 변환하기 위해서는 각각 인수의 XDR필터를 차례대로 호출한다. 일반적으로 대칭 필터와 동일하다.

5.2.3 GRPCProcedure 클래스

GRPCProcedure 객체는 GRPC 프로시저 자신과 그외에 필요한 것들이 서술한다. 이것의 주요 정보는 프로시저를 가르키는 코드넘버(code number)이다.

GRPCProcedure::Creator GRPCCall::Create()를 포인터하도록 정의한다.

GRPCProcedure::Persistence 바인딩의 지속성을 정의하기 위한 enum형이고 다음과 같은 값을 가진다.

- *UseBinding* 바인딩은 활성, 호출 후에도 계속 활성화 됨.
- *OpenBinding* 바인딩은 클라이언트에 의해서 자동적으로 활성화되고 호출 후에도 계속 활성화 됨.
- *CloseBinding* 바인딩은 활성화, 그런, 호출 후에 비활성화.
- *TransientBinding* 바인딩은 클라이언트에 의해서 자동적으로 활성화되고 호출 후에는 비활성화.

GRPCProcedure(id, version, creator, persistence, reply) 이 생성자는 주어진 프로시저의 코드수(id), 버전(version)을 가지고 프로시저 객체를 초기화한다. 그리고, 호출 객체를 생성시키기 위해 주어진 "Creator"를 이용하며 지속성(persistence)를 지원하고 reply는 응답이 필요한지 아닌지를 표시한다.

5.2.4 GRPCVersion 클래스

GRPCVersion객체는 GRPC의 버전(version) 자신

과 그외에 관련된 것들을 서술한다. 이것의 주요 정보는 프로시저의 버전을 가르키는 코드넘버(code number)이다.

GRPCVersion(id, program) 이 생성자는 주어진 버전 코드 수를 가지고 주어진 프로그램(program)을 이용하여 버전 객체를 초기화 한다.

5.2.5 GRPCProgram 클래스

GRPCProgram 객체는 GRPC program 자신과 그외에 관련된 것들을 서술한다. 이것의 주요 정보는 프로그램을 가르키는 코드넘버(code number)이다.

GRPCVersion 클래스가 "versions"을 관리하고 GRPCProgram 클래스는 "program"을 관리함으로써 서로 비슷하다.

GRPCProgram(id, interface) 이 생성자는 주어진 프로그램의 코드넘버와 주어진 인터페이스(interface)를 이용하여 프로그램 객체를 초기화 한다.

5.2.6 GRPCGroupName 클래스

GRPCGroupName 객체는 서버그룹 자신과 그와 관련된 것들을 서술한다. 이것의 주요 정보는 서버그룹을 가르키는 코드넘버(code number)이다.

GRPCGroupName(id, GroupName) 이 생성자는 주어진 그룹 코드넘버와 주어진 그룹 이름을 이용하여 GRPCGroupName 객체를 초기화 한다.

6. Petri nets(PN)와 HGRPC의 검증

6.1 PN의 일반적 정의[11, 12, 13]

PN은 6가지로 구성된 튜플이다:PN=(P, T, E, S, Mo), 여기서 P={p_1, p_2, ..., p_n}, |T|≠0, P는 플레이스의 유한집합, T={t_1, t_2, ..., t_m}, |T|≠0, T는 트랜지션의 유한집합, P∩T=ϕ, E:P×T→N, E는 입력함수, S:T×P→N, S는 출력함수 (N:양의 정수 집합), M_o∈M={M|M:P→N}, M_o는 초기토큰상태.

N=(P, T, E, S, M_0)과 N'=(P', T', E', S', M_0')은 두 개의 PN이라고 하자. 만일 P⊇P', T⊇T', E'=E∩(P'×T'), S'=S∩(T'×P'), 라면 N'는 N의 부분집합(subnets)이라 하고 N⊇N'으로 표시한다. 이 때 만일 t∈•p (t∈p•)이고 t∈T'이면, 플레이스 p∈p'는 N'의 입력문(Input door:ID) (반대로, 출입문 Output door :OD)이라 한다.

마킹 m_1에 대하여 임의의 트랜지션 계열이 점화하여 마킹 m_2를 얻게되면, m_2는 m_1으로부터 도달가능하다. 마킹 m을 갖는 N=(P, T, E, S, Mo)의 도달 집합 R(N, m)은 m으로부터 도달한 모든 마킹의 집합을 의미한다. 초기마킹 m_o를 갖는 N=(P, T, E, S, Mo)은 ∀m∈R(N, m), p∈P에 대하여 m(p)≤1이면 안전하다고 한다. 또한 ∀m∈R(N, m_o), p∈P에 대하여 m(p)≤k인 정수 k가 존재한다면 보존하다고 한다.

6.2 HGRPC프로토콜의 검증

HGRPC의 구성은 앞 절에서 설명하였듯이 일반적인 동작의 흐름, 동적그룹지원관리 그리고 결과처리를 모델링하여 검증한다.

1)GRPC의 동작

HGRPC의 동작에는 클라이언트, 서버, 서버스토브와 로칼서버, 로칼서버스토브를 기준으로 나타낼 수가 있다.

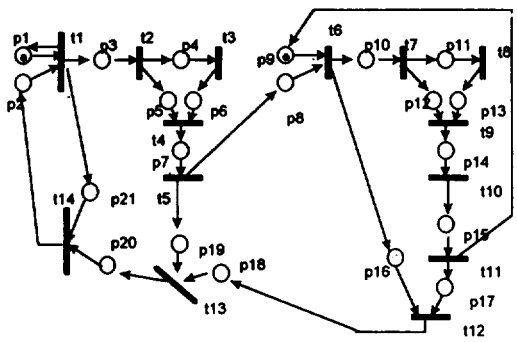
HGRPC의 동작 메카니즘은 총14개의 트랜지션과 21개의 플레이스로 구성되는데, 트랜지션과 플레이스의 집합은 〈표 1, 2〉와 같다.

본 모델링은 기존에 발표되어진 매크로 플레이스 축조변형[12, 14, 15, 20, 21]을 이용한다면 한 개의 매크로 플레이스로 구성되어 질수있다(그림 7). (여기서 이중원과 이중막대는 각각 마크로플레이스와 마크로트랜지션을 의미한다) 따라서 토큰이 일정한 값을 유지하고, 모든 트랜지션이 도달가능하며 안전하고, 보존적인 성격을 만족하고 있다고 하겠다. 따라서 HGRPC의 동작메카니즘의 과정의 흐름은 타당하다고 하겠다. 또한 동작메카니즘 모델은 단일 스케쥴로 구성되어있어 도달성을 쉽게 증명 할 수있으며 다시 원래의 상태로 돌아감을 스케쥴을 통하여도 쉽게 이해 할 수있다.

S:t1t2t3t4t5t6t7t8t9t10t11t12t13t14

2)HGRPC의 그룹관리

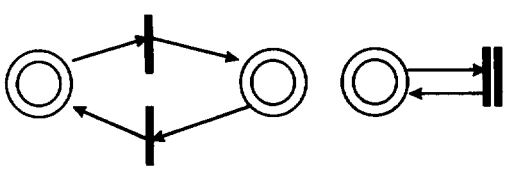
동적그룹지원을 위한 그룹관리는 Create/destroy와 Join/leave를 각각 부페트리 네트의 모델화하여 각각의 부페트리 네트 모델들이 도달가능하고 보존적이



(그림 5) HGRPC 동작매커니즘의 패트리 네트
(Fig. 5) Petri nets model of the function mechanism of HGRPC

〈표 1〉 트랜지션 집합
〈Table 1〉 A set of transitions of (Fig. 5)

트랜지션	내 용
t1	프로세스호출
t2	글로벌바인더에게 전송
t3	글로벌바인더에서 점검
t4	서버스토브에 전송
t5	로칼서버에게 전송
t6	서버데이타형으로 변환
t7	로칼서버에게 요청
t8	로칼서버스토브에게 회신
t9	프로시저를 시행
t10	서버스토브에 전송
t11	XDR로 변환
t12	글로벌서버에게 전송
t13	글로벌서버데이타형으로 변환
t14	결과를 서버에게 전송



(a)축조된 HGRPC모형
(b)최종적으로 축조변형된 HGRPC

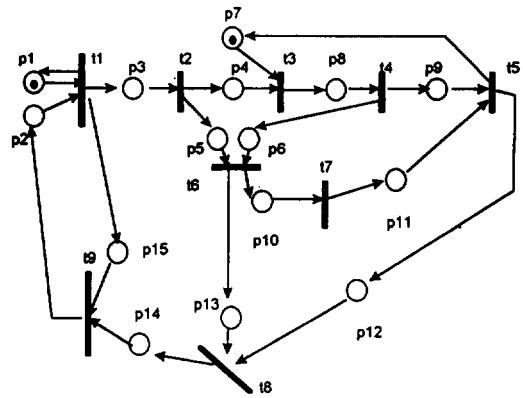
(그림 6) 축조변형된 HGRPC 동작매커니즘모델
(Fig. 6) Reduced petri nets of HGRPC mechanism

〈표 2〉 플레이스의 집합
〈Table 2〉 A set of places of (Fig. 5)

플레이스	내 용
p1	초기
p2	호출의뢰
p3	네트워크메세지
p4	GB데이타형으로 변환
p5	대기시간
p6	멤버쉽정보
p7	VDR변환
p8	호출요청메세지
p9	초기치
p10	로칼서버데이타형 변환
p11	정보요청
p12	대기
p13	로칼서버정보
p14	로칼서버스토브에 전송
p15	프로시저시행
p16	대기
p17	서버스토브에 전송
p18	글로벌스터브에 전송
p19	대기
p20	글로벌서버데이타형으로 변환
p21	대기

며 안전하다면 전체의 모델도 그와같은 성질을 유지한다는 패트리 네트의 일반적인 성질들을 이용한다.

(그림 8)은 Creat/destroy의 처리 모델이며 (그림 9)는 Join/leave의 과정을 모델링하였다. 또한 이 두개의 모델링을 종합한 것이 (그림 10)으로 그룹관리를 나타낸다.



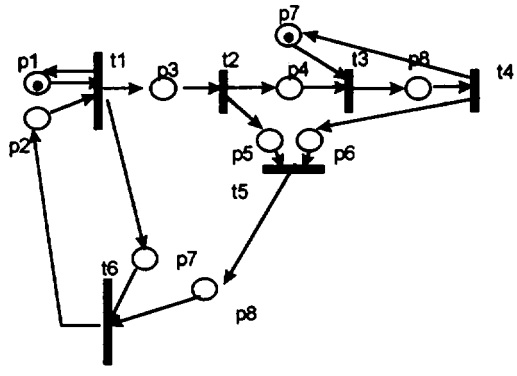
(그림 7) Creat/Destroy 메세지 처리모델
(Fig. 7) Process model of Creat/Destroy

〈표 3〉 (그림 8)의 트랜지션 집합
〈Table 3〉 A set of transitions of (Fig. 8)

트랜지션	내 용
t1	Crea/destory 메세지 요청
t2	글로벌서버에게 GroupID여청
t3	새로운 GroupId할당
t4	그룹테이블에 저장
t5	GroupID를 링크
t6	새로운 그룹테이블 생성(로칼서버)
t7	글러벌서버에게 GroupID링크요청
t8	링크시행우 로칼서버에게 전송
t9	로컬바인더정리

〈표 4〉 (그림 8)의 플레이스 집합
〈Table 4〉 A set of places of (Fig. 8)

플레이스	내 용
p1	대기
p2	메세지 발생
p3	로칼서버에게 전송
p4	GroupID요청 메세지
p5	대기
p6	GroupID정보
p7	대기
p8	GroupID할당
p9	해당그룹의 인덱스 저장
p10	로칼서버의 그룹테이블에 저장
p11	GroupID의 링크 요청
p12	링크요청 결과
p13	대기
p14	로칼서버에 저장
p15	대기



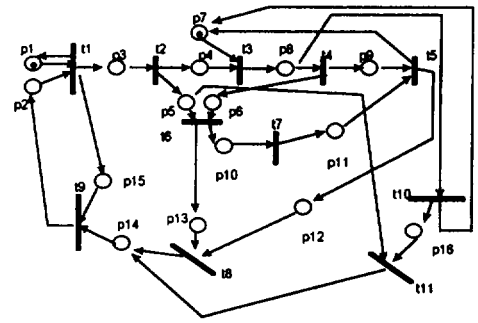
(그림 8) Join/Leave 메세지 처리 모델
(Fig. 8) Process model of Join/Leave

〈표 5〉 (그림 9)의 트랜지션 집합
〈Table 5〉 A set of transitions of (Fig. 9)

트랜지션	내 용
t1	메세지 전송
t2	글로벌서버에게 등록 사실통고
t3	그룹테이블정리
t4	그룹테이블 인덱스통고
t5	저장
t6	인덱스 할당

〈표 6〉 (그림 9)의 플레이스 집합
〈Table 6〉 A set of places of (Fig. 9)

플레이스	내 용
p1	대기
p2	메세지 발생
p3	저장
p4	글로벌서버에게 통고
p5	대기
p6	그룹테이블 인덱스통고
p7	대기
p8	그룹테이블 저장
p9	인덱스 할당
p10	대기



(그림 9) HGRPC의 그룹관리 모델링
(Fig. 9) Group management model of HGRPC

이 모델의 분석은 도달성을 중심으로 분석하기 위하여 스케줄을 작성하여 보면 다음과 같다.

S1 : t1t2t3t4t6t7t5t8t9 ; Create/Deatroy의 스케줄

S2 : t1t2t3t4tt5t6 ; Join/Leave의 스케줄

S3 : t1t2t3t4t6t7t5t8t9 ; Create/destroy의 스케줄 (그림 10)

S4 : t1t2t3t10t11t9 ; Join/Leave의 스케줄(그림 10)

여기에서도 모든 트랜지션이 점화가능하며 또한 모든 플레이스가 $\forall|p|=1$이므로 안전적이고 보전적임을 알 수가 있다.

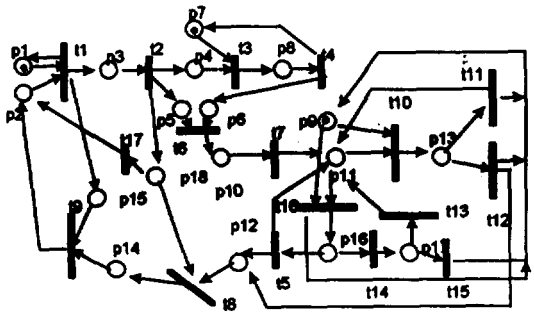
3)결과처리 분석

결과처리에 있어서 제일 중요한 것은 로칼 서버에서 모든 호출의 결과를 종합한 후에 글로벌서버에로 송신을 한다는 점이다. 따라서 이러한 결과처리를 패트리 네트으로 나타내면 (그림 11)과 같다.

결과처리모델에 있어서는 실패처리와 성공처리 그리고 타임아웃 처리로 나누어 스케쥴을 분석 할 수가 있다.

S1 : t1t2t17 ;타임아웃
S21 : t1t2t3t4t5t7t10t11 ;실패메세지처리
S22 : t1t2t3t4t5t7t10t12t18 ;최종실패메세지처리
S31 : t1t2t3t4t5t7t12t5 ;처음성공메세지
S32 : t1t2t3t4t5t7t16t15 ;마지막성공메세지
S33 : t1t2t3t4t5t7t16t13 ;두번째이상성공메세지

여기서 살펴 본다면 처초의 성공 메세지를 글로벌 서버에게 전송하여 결과를 처리한다고 하더라도 로칼 서버는 모든 멤버로부터 그 결과들을 전부 수신하여야한다. 또한 실패인 경우 모든 결과를 받아 전부 실패일 경우 실패로 간주하므로, 만일 고칼 서버가 이러한 내용을 모두 글로벌서버에게 전송한다면 통신망의 부하는 크게 늘 것이다. 또한 이러한 메세지들은 모두 불필요한 것이므로 로칼서버에서 관리하여 최종 결과만을 글로벌 서버에게 전송하므로 통신망을 효율적으로 운영관리 할 수가 있다. 이러한 스



(그림 10) 결과처리 모델
(Fig. 10) Process model of result handling

〈표 7〉 (그림 11)의 트랜지션집합
〈Table 7〉 A set of transitions of (Fig. 11)

트랜지션	내 용
t1	호출 발생
t2	XDR변형 및 전송
t3	호출메세지수신
t4	XDR변형
t5	첫번째성공메세지
t6	멤버에게 송신
t7	결과 수신
t8	글로벌서버에서 수신 및 변형
t9	결과처리
t10	실패
t11	실패메세지도착
t12	최종실패메세지도착
t13	최종이전실패메세지
t14	2번째 이후 성공결과메세지
t15	최종성공결과메세지
t16	성공메세지수신
t17	타임아웃

〈표 8〉 (그림 11)의 플레이스 집합
〈Table 8〉 A set of place of (Fig. 11)

플레이스	내 용
p1	대기
p2	호출메세지
p3	XDR변형
p4	로칼서버에게 전송
p5	대기
p6	호출결과수신
p7	대기
p8	멤버에게 호출요청
p9	대기
p10	호출결과 수신
p11	결과 검사
p12	성공벅 결과
p13	실패 메세지
p14	서버에서 결과 처리
p15	대기
p16	결과 순서(성공) 검사
p17	최종결과 순서 검사
p18	대기

케쥴을 통하여 결과처리의 모델도 도달성과 안전성 그리고 보전적임을 알 수가 있다.

7. 토의 및 결론

본 연구에서는 통신망에 연결된 많은 노드들이 새로운 하나의 그룹을 이루어서 효율적이고 신뢰성 있는 서비스를 제공하는 그룹 통신 방법을 SUN RPC 시스템에 적용시켜서 그룹 원격 프로시저 호출(Group Remote Procedure Call)을 설계하였다. 또한 그 모델링과 분석을 Petri nets를 이용하여 동작메카니즘, 동적그룹지원관리와 결과처리로 구분하여 각각 검증하였다.

HGRPC는 분산 시스템을 구성하기 위해 많이 사용되고 있는 RPC 프로토콜의 장점인 신뢰성(reliability)과 효율성(facility)을 그대로 유지하고 있기 때문에 HGRPC도 마찬가지로 단순성과 쉬운 접근 방식을 가지고 있다. 특히 본 연구에서 제시된 GRPC는 글로벌 바인더와 멤버관리를 통하여 로칼그룹에서의 조절과 글로벌그룹에서의 조절관리 능력을 각 서버들이 보유한 관계로 로칼단위로의 통신을 유도하므로 글로벌 서버에게 로드를 걸리게하지 않는 장점이 있다. 그러나 이러한 연구의 설계에는 메모리의 확장이 요구되는 단점이 있다.

앞으로의 주요 연구과제로서는 HGRPC의 기능 향상을 위해 많은 연구가 진행되어야 하는데, 구체적으로 다음과 같은 연구가 진행되고 있다.

- SUN RPC 시스템의 RPCGEN을 기반으로 한 HGRPC 시스템의 GRPCGEN 컴파일러의 개발
- 기존의 RPC와 구현된 GRPC 프로토콜 비교 실험 및 성능분석평가.

참 고 문 헌

[1] A.D Birrel and B.J. Nelson, "Implementing Remote Procedure Call", ACM Trans. On Computer System, vol. 2,no1, pp. 39-59, Feb. 1984.

[2] John Bloomer, Power Programming with RPC, O'Reilly & Associates, 1992.

[3] G. Coulouris, etc., Distributed Systems-Concepts and Design, Addison-Wesley, 1994.

[4] A.S. Tanenbaum, Modern Operation Systems, Prentice-Hall, 1992.

[5] H. Garcia-Molina and A. Spauster, "Message Ordering in Multicast Environment", Proc. Of 9th Int'l Conf. On Distributed Computing Systems, 1989.

[6] 이광휘, 조현주, 김한수, "분산네트워크 관리 시스템 지원을 위한 멀티캐스트 프로토콜의 설계 및 구현", 창원대학교, 기초과학논문집, vol. 6, 1994.

[7] Kwang-Hui Lee, "A Group Communication Protocol for Distribute Network Management System", Proc. Of IEEE SICON/ICIE '95, Singapore, 1995.

[8] RFC1831 RPC: Remote Procedure Call Protocol Specification Version 2

[9] RFC1832 XDR: External Data Representation Standard

[10] K.Birman, R.Cooper, O.Hagsand, and H.Herzog, "Object-oriented Reliable Distributed Programming", Proceedings Second International Workshop on Object Orientation in Operating Systems, pp. 180-188, 1992.

[11] G. W. Brams, "Reseaux de Petri theorie et Pratique", Masson, Paris, 1982.

[12] G. Berthelot, "Transformations de reseaux de Petri", TSI, 4(1), pp. 91-101, 1985.

[13] J.L.Peterson, "Petri Net Theory and the Modeling of System", Prentice-Hall, NJ, 1981.

[14] J.K.LEE, "세미조인을 기반으로한 페트리넷의 형식적 정의", 정보처리학회논문지, 1(2), pp. 202-214, 1994.

[15] J.K.LEE, "일반 페트리넷의 관계적 축조," 기초과학연구소 논문집 제 4호, pp. 141-156, 1993.

[16] LEE J.kun, LEE K.Hui, "Modeling of the Multicast Transport Protocols using Petri Nets", Proc. Of IEEE SICON/ICIE '95, Singapore, pp. 106-110, 1995.

[17] M.A.Hiltunen, R.D.Schlichting, "Constructuring a Configurable Group RPC Service", Proc. Of 15th Distributed Computer System, Vancouver, Canada, 1995, pp. 288-295.

[18] C.S.LEE, K.H.LEE, J.K.LEE, "A Group RPC Protocol for Distributed System", Proc. In ICICS

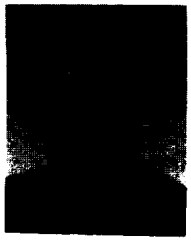
'97, Singapore, pp. 805-809, 1997.

[19] J.K.LEE, C.S.LEE, K.H.LEE, "A Group RPC Protocol for Distributed System Using Petri nets", Proc. In IEEE-SMC '97, Orlando, USA, pp. 4382-4387, 1997.

[20] K.H.Lee, J.Favrel, "Hierarchical reduction method for Analysis and decomposition pf Petri nets", IEEE tr. On System, man and cyb., SMC-15(2), pp. 272-280, 1985.

[21] C.Andre et alt., "Synthese et Realisation des Systemes Logiques a Evolution Simultanee", RADIO, 10, pp. 67-86, 1976.

[22] Xingwei Wang, Hong Zhao and Jiakeng Zhu, "GRPC:A Communication Cooperation Mechanism in Distributed Systems", Operating Systems Review, 27(3), Jul 1993.



이 종 근

1974년 숭실대학교 전산과 졸업
1978년 고려대학교 경영대학원 경영학(경영학석사)
1986년 숭실대학교 대학원 전산과(공학석사)
1992년 몽페리에대학교 전산과(박사과정수료)
1987년~1992년 LSI/USTL 연구원
1993년~1995년 창원대학교 전산소장
1983년~현재 창원대학교 전산과 교수
관심분야:Petri Nets 이론과 변형, 성능분석, FMS 스케쥴링 분석

이 광 휘

1983년 고려대학교 전자공학과(공학사)
1985년 고려대학교 대학원 전자공학과(공학석사)
1989년 고려대학교 대학원 전자공학과(공학박사)
1988년~현재 창원대학교 전자계산학과 교수
관심분야:network management, 분산시스템

이 창 석

1996년 창원대학교 전자계산학과 졸업
1997년 창원대학교 대학원 전자계산학과 석사과정
관심분야:컴퓨터 네트워크, 분산시스템