결함 허용을 제공하는 원격 프로시듀어 호출 기법

한 석 진† 구 용 완††

요 약

원격 프로시듀어 호출은 좀 더 효율적이며 신뢰성이 가미된 분산 프로그램을 프로그래머가 작성하기 쉽도록 하기 위하여 연구 되어 왔다. 본 연구에서는 하드웨어 결함에 대한 신뢰성 있는 결함 허용 원격 프로시듀어 호출 기법을 제시한다. 결함 허용은 chain이라 불리는 노드의 그룹으로 프로시듀어를 복제하여 제공되며, chain 내의 사본들은 선형적인 순서로 되어 있다. 프로시듀어에 대한 호출은 chain 내의 첫 번째 사본(primary copy)으로 보내지며, 나머지 사본에게는 내부적으로 전파된다. 결함이 발생한 경우, 결함이 발생하지 않은 chain 내의 첫 번째 사본이 caller에게 결과를 반환하게 된다. 특히, 제한적으로 ack message를 사용함으로써 중복된 call message와 result message의 처리를 피하였다. 이 기법은 기존의 원격 프로시듀어 호출에 비하여 효율적이고 신뢰성 있는 결함 허용을 제공하게 된다.

Remote Procedure Call Scheme to Support Fault-Tolerance

Suk Jin Han † Yong Wan Koo ††

ABSTRACT

RPC(Remote Procedure Call) has been studied for programmer to easily write distributed program of little bit higher efficiency and reliability. In this study, fault-tolerant remote procedure call for hardware failures is proposed. Fault-tolerance is supplied by replicated procedures with node group, so called chain, and copies along chains are linearly ordered. Calls for procedure are sent to primary copy along chains, and other copies are propagated internally. If failures happen, first copy in faultless chain returns the result to the caller. Especially, in this study processing of redundant call message and result message, while using limited ack message, are avoided. This method supplies efficient and reliable fault-tolerance compared with existing remote procedure call.

1. 서 론

디지탈 컴퓨터가 처음 개발되던 시절부터 시스템 신뢰성(reliability)은 주요 관심사가 되어 왔다. 초기의 진공관이나 릴레이 등과 같은 신뢰성이 없는 소자들로 만들어진 컴퓨터는 한번에 수분 이상 연속적으로 수행하는 것 조차 어려웠다. 그러나 반도체 소자가 개발되고 그 집적도가 높아 갈수록, 기본 단위 기능 당 신뢰도가 획기적으로 향상되고 있는 것이 사실이다. 그러나, 오늘날, 향상된 기능에 대한 요구에 부응하기 위한 컴퓨터 복잡도(complexity)의 증가가 거의 이러한 기본 단위 기능 당 신뢰도 증가 속도에 달하므로, 전체 시스템 자원에서의 신뢰성 문제는 여전히 문제점이 될 수 밖에 없다.

이러한 고 신뢰도에 대한 요구를 만족시키기 위한 것이 바로 결함 허용 시스템이다. 결함 허용 시스템이라 함은 그것을 구성하는 요소 중의 하나가 고장이 나더라도 계속 수행을 하도록 설계된 시스템을 의미한다. 결함 허용은 상대적인 말로서, 결함 허용성을 어느 정도 가진 시스템이라도 결함 허용 시스템으로 분류되지 않을 수도 있다. 예를 들어, 대부분의 시스템들은 디스크나 테잎에 있는 데이타를 읽을 때 에러가 날 경우, 보통 여러 번 재시도를 하도록 설계되어 있다. 이것도 하나의 결함 허용 기법을 사용하고 있지만, 이러한 시스템들을 모두 결함 허용 시스템이라 부르지는 않는다. 오늘날 결함 허용으로 분류되는 시스템은 특정한 표준 수준의 결함 허용성

† 정 회 원 : 수원대학교 대학원 전자계산학과
†† 정 회 원 : 수원대학교 전자계산학과 교수
논문접수 : 1994년 12월 6일, 심사완료 : 1995년 6월 22일

을 가지는 시스템을 말한다. 일반적으로 인정하는 표준은 "결함 허용 시스템은 모든 단일 결함을 허용하는 시스템"이라는 것이다[14].

특히, 본 논문에서는 원격 프로시듀어 호출 과정에서 관련된 프로시듀어를 소유하고 있는 노드의 결함으로 인한 문제를 해결하기 위해, 하드웨어 결함 허용 기법에서 흔히 사용하고 있는 복제 기법을 사용함으로써 새롭고 효율적인 결함 허용 원격 프로시듀어 호출 기법을 제시하고 있다.

한편, 분산 시스템 환경에서, 타 시스템의 자원을 공유하기 위한 방법으로써, 1) 분산 데이타 베이스 시스템에서의 자원 접근 기법을 이용한 자원의 조회, 획득 및 수정을 하는 방법, 2) 주로 운영체제 커널 수준에서의 기법을 적용하여 프로세스간의 통신을 통한 자원을 공유하는 메세지 전달 방법, 3) 프로그램 언어 수준에서 프로시듀어의 호출을 통해 타 시스템의 자원을 공유하는 원격 프로시듀어 호출(RPC : Remote Procedure Call) 방법이 있다.

분산 데이타 베이스 시스템은 중앙 집중식 데이타 베이스 시스템의 결함이라 할 수 있는 한 노드로의 집중적 과부하로 인한 위험 부담 및 비용을 감소시킬 목적으로 논리적(logical)으로는 같은 시스템에 속하지만 물리적으로는 컴퓨터 네트워크를 통해 노드별로 데이타를 분산시켜 놓은 형태를 의미한다.

메세지 전달을 통한 자원 공유 기법은 통상적으로 하나 이상의 프로세스에 의해서 접근되는 포트나 메일 박스를 통해서 정보교환이 이루어지게 되는 복수 개의 제어 링크를 갖는 것을 특징으로 한다[3].

또한, 일단 메세지를 상대편 주소에 전달해 주고 나면 그 다음의 수행 과정 이후의 단계는 고려하지 않아도 좋은 비동기적인 특징을 갖고 있으므로 병행성이 좋을 뿐만 아니라 상대적으로 단순하고 효율적이지만, 몇 가지 단점도 지니고 있다. 즉, 주로 운영 체제 수준의 저 수준 프리미티브를 사용함에 따른 사용자 수준에의 가시성을 제공하기 어렵다는 점과 기존 사용 중인 ALGOL 형태의 프로그램 언어들과의 변수의 형(type)과 그 범위에서 불일치성을 초래할 가능성이 높기 때문에 이를 위한 새로운 절차를 별도로 개발하여야 한다는 문제점을 안고 있다[19].

이에 반해, 원격 프로시듀어 호출 기법은 대개 순차적인 제어 구조의 특징을 갖고 있어서 분산 데이타 베이스 방식이나 메세지 전달 방식에 비해서 병렬성이 약하다는 단점은 있으나 프로시듀어 호출 형태로 사용되기 때문에 사용자에게 단순한 인터페이스를 제공한다는 장점을 지니고 있다. 따라서, 원격 프로시듀어 호출 기법이 분산 프로그램에서 폭 넓게 사용되고 있다. 특히, 원격 프로시듀어 호출은 분산 시스템을 구성하는데 있어서 가장 어려운 점(통신상의 복잡성, 병행성, 전송 오류, 노드 결함 등)을 사용자로부터 감추게 할 수 있는 도구로 제안되어 왔으며 프로그램 언어 수준에서 프로시듀어의 호출을 통해 타 시스템의 자원을 공유토록 하는 방법으로 사용되어 왔다.

본 논문에서는 노드 결함에 대한 결함 허용 원격 프로시듀어 호출 기법을 제시한다. 노드 결함을 가지고 있는 경우, caller나 callee는 수행이 되지 않는다. caller에 결함이 발생한 경우, callee는 결과를 반환할 수 없게 되고 Orphan (고아) 프로시듀어가 된다. callee에 결함이 발생한 경우, caller는 결과를 무한정 기다리게 된다[11].

이에 따라, 본 논문에서는 chain 이라 불리는 결함이 발생하지 않은 노드의 그룹에 서비스를 수행하는 프로시듀어를 복제함으로 결함 허용 원격 프로시듀어 호출 기법이 제공된다. 또한, chain 내의 사본은 선형적으로 구성되어 있으며, 호출은 chain내의 첫번째 사본으로 보내지고 chain 내의 남은 모든 사본에게 전파된다. 그 호출을 수행할 수 있는 동일한 사본을 결함이 발생하지 않은 또 다른 노드에 복제를 함으로써 이루어진다.

복제된 사본들은 선형적인 chain으로 구성된다. i 번째 사본을 위해, i+1 번째 사본은 그것(i 번째 사본)의 백업을 구성한다. service request(서비스 요구)는 primary(chain내의 결함이 없는 첫번째 사본)에게 행해진다. primary callee는 그 자신의 백업(primary가 아닌 나머지 사본들)에게 수취한 call message를 전파함으로써, 모든 사본들이 호출된다. 호출의 결과는 pri-

mary callee에 의해 primary caller에게 반환된다. primary callee에 결함이 발생한다면 그것의 백업 사본이 primary callee의 역할을 맡게 된다.

중복된(nested) 원격 프로시듀어 호출에서 primary 사본만이 실제 호출을 만들어 내고, 또 다른 caller 사본들은 primary caller로부터 결과를 기다리기만 한다. primary caller에 의해 수취된 결과는 모든 다른 caller 사본들에게 전파된다. 만일 primary caller가 결과를 응답하는 과정에서 결함이 발생한다면, 그것의 backup 사본에게 결과가 전송된다.

기존의 결함 허용 원격 프로시듀어 호출 기법에서는 프로시듀어의 결함을 탐지하기 위해 ack message를 사용하고 있다. 이 ack message는 call message와 result message 또는 done message에 대한 확인의 의미로 사용하는 message이다. 하지만 이런 ack message를 사용함으로써 call message와 result message가 중복됨을 보이고 있다. 이에 따라, 상당한 메세지 오버 헤드가 발생하며 결함이 발생하지 않을 때도 ack message에 대한 오버 헤드가 발생하게 된다[11]. 또는 broadcast 방식을 사용하여 복제된 모든 사본에게 서비스를 요구하고, 모든 사본들로부터 결과를 기다리는 방식이 있지만, 이것 또한 많은 메세지 오버 헤드가 발생되는 기법으로 간주되고 있다[30].

본 연구에서는 전자의 경우를 개선하고 있다. 즉, ack message를 caller만이 전송하도록 제한함으로써 새롭고 좀 더 효율적인 결함 허용 원격 프로시듀어 호출 기법에 대해 제안한다.

2. 관련 연구

2.1 결함 허용

기존의 결함 허용 기법은 하드웨어 결함 허용 기법, 정보 결함 허용 기법, 소프트웨어 결함 허용 기법 등 시스템 구성 요소별로 구분할 수 있다. 모든 기법들이 근본적으로 여분의 자원(redundancy resource)을 사용하는 원칙을 두고 있다. 여러 가지 결함 허용 기법들이 제안 되어 왔지만 그 중 대표적인 것들만 간단히 살펴보면 다음과 같다[5, 8, 9, 12, 13, 16, 75].

2.1.1 하드웨어 결함 허용 기법

하드웨어로 구현한 것을 하드웨어 결함 허용 기법이라 하는데, 예로서 Triple Modular Redundancy(TMR), Duplication with Comparison, Primary-Standby Approach, Watchdog Timer, Modular Redundancy Approach등이 있다.

2.1.2 정보 결함 허용 기법

정보 결함 허용 기법은 시스템 입출력 또는 시스템 내의 모듈간의 정보 교환시 발생되는 노이즈(noise), 하드웨어 결함 또는 소프트웨어 결함 등에 의해 정보에 이상이 생기는 것을 감지하거나 방지하기 위해 여분의 정보를 추가하는 것으로써 대표적으로는 다음과 같다.

Parity code, m of n code, checksum, berger code, hamming error correcting code등이 있다.

2.1.3 소프트웨어 결함 허용 기법

소프트웨어 결함 허용 기법은 소프트웨어 모듈의 중복이나 재수행(rollback and retry), 또는 이 두 가지 방식의 혼용에 기초를 두고 있다.

대표적인 것은 check pointing, recovery block, conversation, distributed recovery block, N self-checking programming, N-version programming등이 있다.

2.2 분산 시스템에서의 원격 프로시듀어 호출

개방시스템간의 접속(OSI;Open System Interconnection) 참조 모델의 응용 계층은 응용 프로세스 사이의 통신을 제공함으로써 응용 프로세스에게 분산 처리 시스템에 존재하는 여러 가지 서비스를 사용할 수 있게 한다. 이러한 서비스 중 응용 프로세스가 원격 노드에 존재하는 프로시듀어를 사용함으로써 수행을 분산시킬 수 있게 하여 주는 응용 서비스를 원격 프로시듀어 호출이라 한다[4, 17, 29, 2]. 원격 프로시듀어 호출은 응용 프로세스 사이의 통신을 프로시듀어를 호출하는 형식으로 제공함으로써 여러 노드에서 병렬적으로 수행될 수 있는 분산 프로그램을 작

성할 수 있는 환경을 만들어 준다[18].

원격 프로시듀어 호출의 기본 모델은 프로그래머에게 원격 프로시듀어를 국부적 프로시듀어를 사용하는 것과 같은 방법으로 사용할 수 있게 함으로써 프로세스간의 통신을 제공한다. 따라서, 원격 프로시듀어 호출의 동작은 동작 방법이 정의되어 있는 원격의 프로시듀어에 필요한 인자(Argument)를 전송함으로써 원격 노드로 수행이 분산된다. 원격 노드에서 원격 프로시듀어에 의하여 수행된 결과 값은 원격 프로시듀어의 반환(Return)의 형태로 국부 프로시듀어에 전송된다. 이러한 형태는 프로그래머가 네트워크에 대한 아무런 고려를 하지 않아도 되므로 프로그래머의 관점에서는 매우 간단하고 편리하게 분산 프로그램을 작성할 수 있다.

원격 프로시듀어 호출은 하나의 요구에 대한 응답 시간인 호출 시간(Latency time)이 짧은 것을 기본 성격으로 한다. 따라서, 요구-응답 응용(request-response application)에 적합하다. 연결의 설정은 호출 시간을 짧게 하는데 많은 부담을 주므로 원격 프로시듀어 호출은 트랜스포트와 네트워크 계층이 연결 지향 서비스(Connection Oriented Service) 보다는 비연결 지향 서비스(Connectionless Oriented Service)를 제공하는 상황에서 구현되어지는 경우가 많다[17, 4]. 따라서 원격 프로시듀어 호출은 많은 양의 데이타를 한꺼번에 보내는데 사용하기에는 부적합하다[21, 7].

2.2.1 원격 프로시듀어 호출의 구성 요소

원격 프로시듀어 호출은 네트워크상의 다른 프로세스와 통신하기 위하여 프로시듀어 호출의 형태를 사용하므로 RPC가 구현되어지는 프로그램 언어의 성격에 따라 실제적인 구조가 달라질 수 있다. 따라서, 프로그램 언어는 중요한 요소 중 하나이다.

하나의 프로세스가 원격 프로시듀어를 사용하기 위하여는 호출하기 위한 프로시듀어의 네트워크상의 위치와 그 프로시듀어의 인터페이스의 정의를 알아야 한다. 따라서, 이러한 정보를 가지고 있는 네트워크상의 데이타 베이스가 필요한데, 이것을 RPC 디렉토리 서버(RDS : RPC Directory Server)라 하며, RDS로부터 원하는 정보를 얻어와 실제로 사용할 수 있도록 결합하는 과정을 바인딩(binding)이라 한다[20, 6]. 특히, RDS는 네트워크상에 잘 알려진 주소를 가지고 있다.

원격 프로시듀어 호출에서 또 하나의 중요한 요소는 원격 프로시듀어를 고객에게 제공하는 RPC 서버이다[20, 6]. RPC 서버는 하나 이상의 원격 프로시듀어를 자신의 루틴 안에 포함하고 원격 프로시듀어의 인자와 결과에 대한 정보와 원격 프로시듀어의 장소에 대한 정보를 네트워크상의 RDS에 등록함으로써 원격 프로시듀어를 사용할 수 있는 방법을 제공한다. RPC 서버는 고객 프로세스의 요구에 의하여 자신이 가지고 있는 프로시듀어 중 하나를 수행하여 그 수행의 결과 값을 반환하는 역할을 한다.

프로그래머는 네트워크에 대한 세부적인 사항을 고려하지 않아야 하므로 프로그래머로부터 네트워크에 대한 자세한 정보를 은폐하기 위하여 사용되는 코드를 스터브(Stub)라 한다. 고객 스터브는 원격 프로시듀어를 요구하는 고객에게는 실제 원격 프로시듀어처럼 보이며 원격 프로시듀어의 스터브는 원격 프로시듀어에 대한 서비스를 제공하는 RPC 서버에게는 고객처럼 보이게 된다. 스터브는 네트워크 프리미티브를 이용하여 네트워크에 대한 동작을 하는 루틴으로 이루어져 있으며 상위 계층으로부터 제공 받은 인자를 통신에 적합 형태로 바꾸기 위하여 정렬화/역정렬화(marshall/unmarshall) 네트워크 프리미티브를 사용한다[20, 6].

2.3 기존의 결함 허용 원격 프로시듀어 호출

위에 제시한 여러 가지 결함 허용 기법 중, 현재는 주로 하드웨어 결함 허용 기법이 실제 많이 적용되고 있다. 특히, 본 논문에서 주안점을 두고 있는 원격 프로시듀어 호출에서의 결함 허용 기법에서는 위에 제시된 기법 중 primary-standby approach와 modular redundancy approach가 주로 적용되고 있다.

이는 원격 프로시듀어 호출에서 결함 허용 기법을 제공하기 위해 프로시듀어를 복제하는 방법을 많이 사용하기 때문이다[11].

특히 기존의 결함 허용 원격 프로시듀어 호출 기법에서는 위에 제시한 primary-standby approach와 modular redundancy approach의 장점이 결합된 기법이 사용되기는 하지만 프로시듀어의 결함 상태를 확인하기 위한 ack message의 사용으로 인해 실제 결함이 발생하든 결함이 발생하지 않든 간에 상당한 오버 헤드가 발생하는 단점을 보이고 있다[11]. (그림 2)는 기존의 결함 허용 원격 프로시듀어 호출 기법에서의 동작 수행 과정을 나타내고 있다.

2.4 본 연구에서 제안된 기법

본 논문에서 제시한 원격 프로시듀어 호출 메카니즘은 앞서 설명한 Primary Standby 기법과 Modular Redundancy 기법의 조합이다. 즉, 본 논문은 참고문헌 [11]과 비교 될 수 있으며, 그 연구에서 발생되는 불필요한 오버헤드를 줄이는 데 역점을 두었기 때문에, 보다 더 신뢰성있고 빠른 결함 허용 원격 프로시듀어 호출 메카니즘을 제안하고 있다.

모든 복제된 모듈들은 각기 능동적이며 병행적으로 call을 수행하며(modular redundancy approach), caller는 모든 복제된 요소들 중 primary로 선언된 요소에게만 호출을 한다(primary-standby approach). primary는 호출 메세지를 받고 나머지 사본들에게 그 호출 메세지를 전파한다. 따라서, 호출 메세지를 모든 복제된 사본들에게 전달할 필요가 없으며, 그 복제된 사본들이 능동적으로 수행을 하기 때문에 이들의 상태를 체크하는 작업 또한 필요 없다.

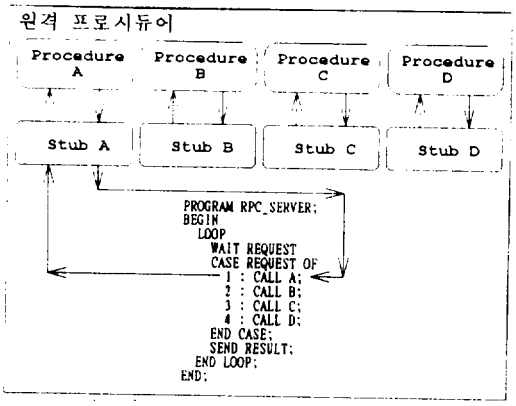
3. 시스템 모델

본 논문에서 제시된 결함 허용 원격 프로시듀어 호출 메카니즘은 modular redundancy approach와 primary-standby approach의 유용한 장점만을 결합한 형태이다. 즉, 모든 복제된 모듈들은 각기 능동적이며 병행적으로 call을 수행하며, caller는 모든 복제된 요소들 중 primary로 선언된 요소에게만 호출을 한다. primary는 호출 메세지를 받고 나머지 사본들에게 그 호출 메세지를 전파한다. 따라서, 호출 메세지가 모든 복제된 사본들에게 전달될 필요가 없으며, 그 복제된 사본들이 능동적으로 수행을 하기 때문에 이들의 상태를 체크하는 작업 또한 필요 없다.

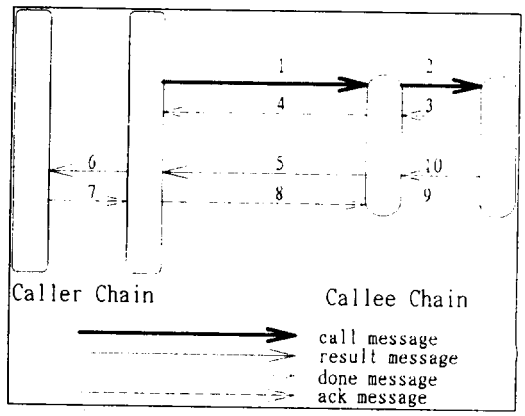
더우기, 프로시듀어의 결함을 탐지하기 위해 사용하는 ack message를 제한적으로 사용함으로써, 기존의 결함 허용 원격 프로시듀어 호출 기법에 비해 ack message에 의한 오버 헤드를 감소할 수 있게 되었다. 또한, 그로 인해 발생하는 call message와 result message의 중복 현상을 방지할 수 있게 된다.

3.1 본 연구에서 사용된 용어의 정의

본 연구에서 제시한 시스템 모델의 용어는 다음과 같이 정의된다.



(그림 1) RPC 서버의 논리적 구조
(Fig.1) Logical structure of RPC server



(그림 2) 기존의 결함 허용 RPC 기법
(Fig. 2) Existing Fault-tolerance RPC mechanism

사본 : 서로 다른 다수의 노드에 상주하는 프로시듀어의 사본으로서, 프로시듀어의 각 사본은 서로 동일하다.

primary 사본 : chain 내의 결함이 없는 첫번째 사본, chain들 사이의 통신을 담당한다.

secondary 사본 : chain 내의 primary 사본이 아닌 다른 모든 사본들로서. primary가 실패 했을 때, secondary 사본은 primary가 되어 그것의 기능을 수행한다.

chain : 복제된 사본의 그룹으로 구성된다.

caller chain : call을 만들어 내는 chain

callee chain : call을 service 하는 chain.

call message : RPC call을 만들어 낸다.

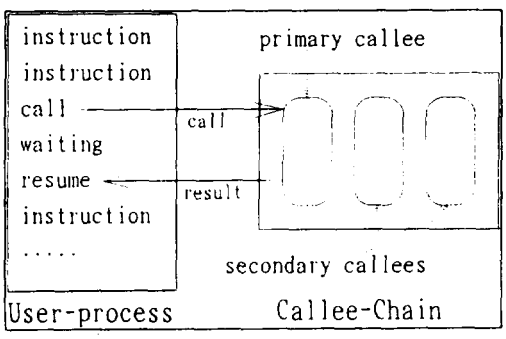
result message : RPC call의 반환 값을 포함한다.

ack message : caller 사본이 result message를 받은 후에 수취의 확인을 알리기 위한 메세지이다.

3.2 시스템 모델의 정의

RPC을 만들어 내는 사용자 프로세스는 복제되지 않으며, 단 하나의 사본만을 갖는 특별한 경우의 caller chain이다. chain 내에서, 사본들은 primary의 역할을 수행하는 우선 순위에 따라 순서화 된다.

(그림 3)은 callee-chain과 사용자-프로세스와의 상호 작용을 보여 주고 있다. 특히, (그림 3)의 callee chain은 두개의 복제된 사본을 가지고 있다. 사용자-프로세스는 수행 도중 callee-chain의 primary callee에게 서비스를 요청한다.



(그림 3) 본 연구의 시스템 모델
(Fig. 3) System model for this study

primary callee는 이 call message를 그것의 첫번째 secondary callee에게 전파한 후 서비스를 수행하고, 첫번째 secondary callee는 두번째 secondary callee에게 call message를 전파한 후 마찬가지로 서비스를 수행한다. 이는 모든 복제된 사본이 능동적이며 병행적으로 수행할 수 있기 때문에 가능하다. 두번째 secondary callee는 call message를 마지막으로 받게 된다.

4. 결함 허용 원격 프로시듀어 호출 메카니즘의 설계

본 연구에서 제시된 기법에서, primary callee는 call message를 수취한 후 그것의 사본에게 그 call message를 전파하므로 callee chain 내의 모든 사본들은 병행적이며 능동적으로 call을 수행한다. 또한, primary caller가 result message를 수취한 후 그것의 사본에게 그 result message를 전파하기 때문에, caller chain 내의 모든 사본들은 동일한 결과의 사본을 수취한다. 그러나, caller는 하나의 call만을 만들어 내며, callee는 단 하나의 결과만을 caller chain에게 보낸다. 여기서, chain들 끼리의 모든 통신은 각각의 primary에 의해 수행된다. secondary 사본은 모든 superior 사본들에 결함이 발생했을 때만 primary의 역할을 수행한다.

특히, 결함이 발생하는 시기와 어느 지점에서 결함이 발생했는 지에 따라 결함 허용 기법을 제공하기 위한 세부적인 메카니즘이 요구되기 때문에, 이에 대한 메카니즘을 제시한다.

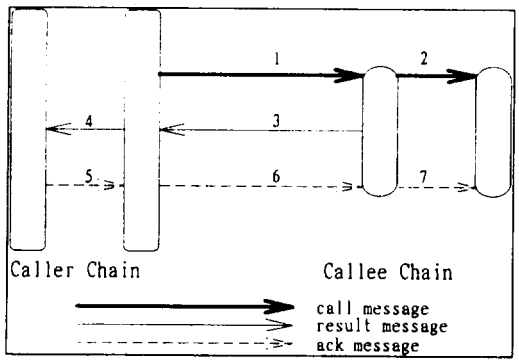
4.1 결함이 없는 경우

call을 수취하는 과정에서, primary callee는 immediate subordinate에게 동일한 call을 전송한다. 즉시, 그 사본은 immediate subordinate에게 그 call을 전송한다. 반면에, 서비스 요구를 받은 primary callee는 Local call을 수행하여 그 결과를 primary caller에게 보낸다.

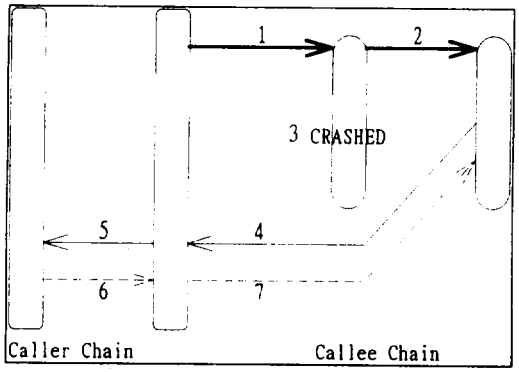
caller chain에서 primary는 RPC call을 전송한 뒤, 결과를 기다린다. secondary caller들도 결과를 기다리지만, 그들은 어떤 call message들을 만들어 내지는 않는다. 각각의 secondary

caller는 immediate superior로부터 결과를 수취하는 것이다. 또한 primary caller는 result message를 전파한 후, secondary caller들로부터 ack message를 기다리며, 그 메세지를 받은 후, primary callee에게 전송한다.

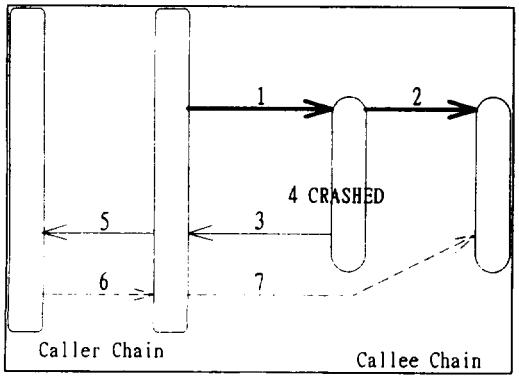
primary callee는 primary caller로부터 ack message를 받고 call이 완성되었다는 것을 알리기 위해 immediate subordinate에게 ack message를 보낸다. 이 메시지는 모든 다른 secondary callee들에게 전파된다. 이러한 동작 과정



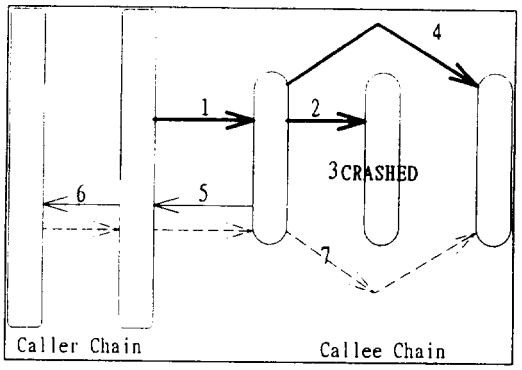
(그림 4) 결함이 없는 경우의 동작 과정
(Fig. 4) Operation process in case of Non-Faults



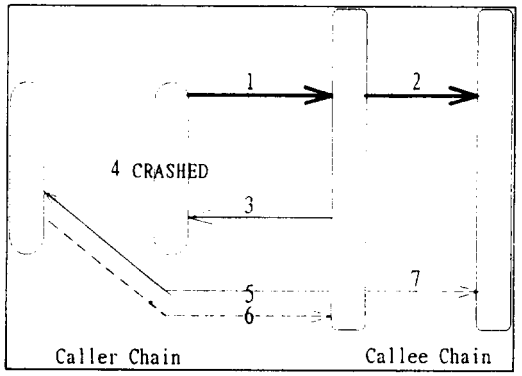
(그림 5) 결함이 발생한 경우 1
(Fig. 5) Case in which fault happens (1)



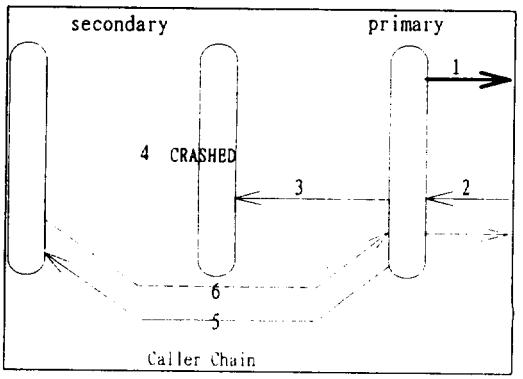
(그림 6) 결함이 발생한 경우 2
(Fig. 6) Case in which fault happens (2)



(그림 7) 결함이 발생한 경우 3
(Fig. 7) Case in wich fault happens (3)



(그림 8) 결함이 발생한 경우 4
(Fig. 8) Case in which fault happens (4)



(그림 9) 결함이 발생한 경우 5
(Fig. 9) Case in which fault happens (5)

을 (그림 4)에 나타내고 있다.

4.2 결함이 있는 경우

RPC call을 primary callee에게 보낸 후, caller는 result message를 기다린다. result message를 primary callee가 전송하기 전에, primary callee에 결함이 발생했다는 것을 primary callee의 백업이 발견한다면, primary callee의 백업은 primary의 역할을 수행하며 result message를 primary caller에게 전송한 후, ack message를 받는다. 이러한 동작 과정을 (그림 5)에 나타내고 있다.

만일 primary callee가 결과를 전송한 다음 secondary에게 done message를 전송하기 전에 primary callee에 결함이 발생했다면, 일정 기간 동안 ack message를 받지 못한 secondary callee는 primary callee를 check하여 결함을 탐지한 후 새로운 primary callee로서의 역할을 한다. primary caller는 ack message를 주기 전에 primary callee에 결함이 발생했음을 탐지하고 새로운 primary callee에게 ack message를 전송한다. 새로운 primary callee는 primary caller로부터 ack message를 받는다. 이러한 동작 과정을 (그림 6)에 나타내고 있다.

call message를 수취하는 과정에서, secondary callee 사본에 결함이 발생했다면, 결함을 발견하는 과정에서 그것의 immediate superior는 결함이 발생된 사본의 immediate subordinate에게 동일한 call message를 전송한다. 즉, 이러한 과정은 결함이 발생된 immediate subordinate가 그것의 immediate superior를 check하여 결함이 발생된 사본의 superior에게 call message를 요구하여 이루어진다. 이러한 동작 과정을 (그림 7)에 나타내고 있다.

primary caller가 그것의 호출 결과를 수취하고 그것을 전파하기 전에 결함이 발생한 경우이다.

primary callee는 result message를 primary callee에게 전송하고, ack message를 기다린다. secondary caller는 일정 기간이 지난 후 result message를 받지 못했다면, secondary caller는 primary caller를 check하여 결함을 탐지한 후 새로운 primary caller가 되어 primary callee로부터 result message를 기다린다. 일정 기간 동안 primary callee가 result message를 보낸 후 일정 기간 동안 ack message를 수취하지 못하면, primary callee는 새로운 primary caller가 되어 있는 secondary caller의 주소로 result message를 다시 보낸다. 이러한 동작 과정을 (그림 8)에 나타내고 있다.

caller 사본은 subordinate에게 result message를 전파하는 과정에서 하나의 secondary에 결함이 발생한 경우이다. 만일 일정 기간 secondary caller로부터 ack message를 받지 못했다면, 결함이 발생된 사본의 subordinate에게 result message를 보내며 ack message를 기다린다. 일정 기간 동안 제2 caller 사본이 제1 caller 사본으로부터 result message를 받지 못했다면, 제2 caller 사본은 제1 caller 사본의 역할을 하며 다시 result message를 기다린다. 이러한 동작 과정을 (그림 9)에 나타내고 있다.

위의 제안된 결함 허용 원격 프로시듀어 호출 메카니즘을 알고리즘으로 표현하면 다음과 같다. (알고리즘1, 2)

```
1)   ROCEDURE s_name(argument)
2)   TRANSMIT of service
3)   WHILE(not result_arrived)
4)   check_time_out
5)   IF(time_out_over)
6)     IF(status_primary_callee == fault)
7)       TRANSMIT of service
8)   END
9)   RECEIVE of result
10)  IF(!last_secondary)
11)  propagate to subordinate the result
12)  WHILE(not arrived_ack_message) ;
13)   check_time_out
14)   IF(time_out_over)
15)     WHILE(status_immediate_subordinate == fault)
16)       POP from stack
17)       IF(status = wait)
18)           TRANSMIT of result
19)     END
20)  END
21)  ELSE
22)  TRANSMIT of ack
23)  RETURN ;
```

(알고리즘 1) 결함 허용 RPC 알고리즘 (Client side)
(Algorithm 1) Fault Tolerance RPC Algorithm(Client side)

```
1)   PROCEDURE s_procedure(argument)
2)   REGISTRATION of server
3)   LOOP BEGIN
4)   WHILE(not arrived_call_message) ;
5)   RECEIVE of service
6)   handling duplicates
7)   PUSH to stack
8)   propagate to subordinate the service
9)   local call
10)  TRANSMIT of result
11)  WHILE(not arrived_ack_message)
12)   check_time_out
13)   IF(time_out_over)
14)       WHILE(status_immediate_superior == fault)
15)           POP from stack
16)           IF(status_immediate_superior == progress)
17)               return ;
18)           IF(status_immediate_superior == wait)
19)               TRANSMIT of result
20)       END
21)  END
22)  END LOOP
```

(알고리즘 2) 결함 허용 RPC 알고리즘 (Server side)
(Algorithm 2) Fault Tolerance RPC Algorithm (Server side)

5. 구 현

본 연구에서 제안된 결함 허용 기법은 Sun RPC 메커니즘 상에서 구현되었으며, Sun RPC는 동적 데이타와 중복된 call를 허용하고 있다.

5.1 Chain의 등록

RPC call 과정에서 caller chain은 callee chain에 대한 정보를 요구하며, callee chain 또한 caller chain의 정보를 요구한다. 이것은 요구를 받아들일 준비가 되어 있을 때, chain을 등록하도록 요구함으로써 이루어진다. 이렇게 chain 내의 모든 사본들은 동일한 방법으로 등록된다.

등록하는 과정에서 요구된 정보는 네트워크 상의 모든 노드로 분산된다. 이것은 어떤 화일에 새로운 chain에 대한 정보를 추가한 후 그것을 분산 시켜서 이루어진다. 그 화일은 요구를 받아들일 준비가 되어 있는 모든 chain에 대한 정보를 포함하고 있다. 각각의 chain은 프로그램-ID에 의해서 표현된다.

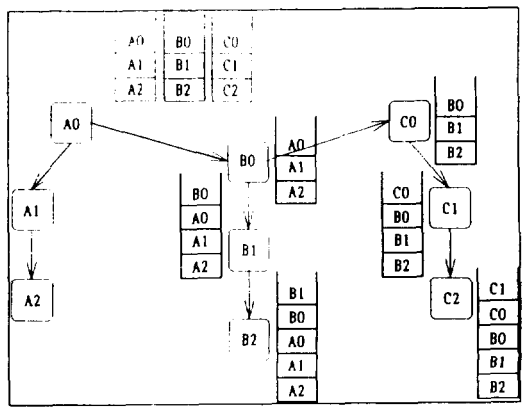
5.2 결함의 탐지

secondary callee는 그것의 immediate superior에 결함이 발생했다는 것을 확인할 필요가 있다. caller는 primary callee에 결함이 발생했다는 것을 확인한 경우에만 또 다른 callee에게 메시지를 전송할 것이다.

본 연구에서는 원격 프로세스의 상태를 스스로 질의(query)하는 RPC 메커니즘을 이용하였다. 따라서, 원격 노드에 있는 또 다른 프로세스의 상태에 관심을 갖고 있는 프로세스는 그 노드에게 query를 전송한다. 만일 질의에 대한 응답이 지정된 시간 안에 확인되지 않으면, 그 프로세스에 결함이 발생했다고 가정한다.

서로 다른 chain에 있는 caller 사본의 상태를 질의하기 위해선 프로그램-ID가 필요하다. caller의 프로그램-ID는 call message의 부분으로 callee에게 전달된다. 그러나, 동일한 chain 내의 사본의 상태를 조사하기 위해선, 추가적인 어떤 정보도 요구되지 않는다.

secondary callee 사본은 caller의 프로그램-ID와 그것의 모든 superior를 알 필요가 있다. 이러한 정보는 스택에 저장된다. 각각의 사본은 그것의 subordinate에게 call message를 전송하기 전에 스택에 자신에 대한 정보를 위치시킨다. 즉 스택의 top은 immediate superior에 대한 정보를 포함하며 뒤따르는 아래 단계는 더 높은 레벨에 대한 정보를 포함한다. caller에 대한 정보는 스택의 bottom에 위치된다.



(그림 10) Chain의 등록과 결함의 탐지
(Fig. 10) Registration of chain and dection of fault

immediate superior에 결함 발생이 탐지될 때 스택은 pop이 되며, 새로운 immediate superior의 정보는 다시 스택의 top이 된다. 스택의 높이가 '1'일 때, 사본이 chain의 primary임을 뜻한다. 위의 일련의 과정을 (그림 10)로 나타내었다.

위 그림에서 프로시듀어 B0에 결함이 발생했을 때, 프로시듀어 B1이 B0의 결함 상태를 탐지하여 자신이(B1) 소유한 스택의 top의 정보(B0)를 pop 시킨다. 그 후 B1은 그때 스택의 top 정보(A0)에게 result message를 보낼 수 있다.

마찬가지로 B1의 결함 상태를 B2가 탐지할 수 있지만, 위 그림에서처럼 2개의 사본이 있는 경우 마지막의 사본(B2)에 결함을 탐지하는 프로시듀어는 제공되지 않는다. 하지만 B2에 결함이 발생했다 하더라도 B0나 B1이 정상적으로 서비스를 수행한다면 정상적인 호출이 완료 될 수 있다. 따라서, 본 논문에서 제안한 결함 허용 원격 프로시듀어 호출 기법에서는 하나의 chain 내에 사본이 N개 있을 경우, N-1 개의 결함을 허용하지만 N개의 결함은 허용하지 않는다.

5.3 중복된 메세지의 처리

본 연구에서는, RPC call이 만들어 질 때, 그 call에 대한 전체적인 네트워크 시스템에 대한 유일한 순서 번호(sequence number)가 생성되도록 하였다. 그것은 등록된 프로그램의 유일한 ID(identified)와 카운트를 사용하여 만들어 진다.

각기 등록된 RPC chain은, 프로그램 번호(program number)와 버전 번호(version number)로 구성되어 있는, 유일한 프로그램-ID에 의해 인식된다. 즉, 프로그램-ID는 서로 다른 RPC chain을 구별하기 위해 사용된다. 카운트는 프로그램이 RPC call을 만들어 내고 증가하는 각각의 시간이다. 동일한 RPC chain에 의해 만들어지는 서로 다른 call은 이 카운트에 의해 구별된다.

등록된 서비스는 모든 caller 프로그램으로부터 call을 수취했을 경우, 모든 caller 프로그램의 순서 번호를 저장한다. 각 caller chain의 가장 큰 순서 번호 즉, 가장 큰 카운트 값만이 유지된다. 그 정보는 연결 리스트(linked-list)에 저장된다. 메시지가 수취 되었을 때, 순서 번호는 연결 리스트 내의 순서 번호와 비교된다. 이때 새로운 프로그램-ID가 수취된 경우에는 그 순서 번호를 연결 리스트에 삽입하며 그렇지 않은 경우에는 카운트 값이 비교된다. 연결 리스트에 있는 순서 번호의 카운트 값이 현재 수취된 순서 번호의 카운트 값보다 크거나 같은 경우, 그 메세지는 중복된 것이다. 만일 그 메세지가 중복되지 않은 경우라면 연결 리스트의 카운트 값이 현재 수취된 순서 번호의 카운트 값으로 대치된다.

6. 성능 평가

본 장에서는 본 논문에서 제시한 결함 허용 원격 프로시듀어 호출 메카니즘을 평가하기 위한 시뮬레이션 모델에 대한 가정 및 환경을 기술하고, 기존의 결함 허용 원격 프로시듀어 호출 메카니즘과의 비교를 통하여 본 논문에서 제시한 메카니즘의 효율성을 보이고 있다.

6.1 큐잉 모델

본 논문에서 채택한 분산 시스템 모델의 형태는 N*(M/M/1) 시스템이다. 큐잉 모델을 선택한 이유는 비록 모델의 성격이나 규정(specification)에 대해서는 정확한 정보를 제공할 수 없다는 단점이 있지만, 모델의 시간적인 변화 추이를 제공하는 데에는 상대적으로 Petri-net 등의 타 모델 기법에 비해 효율적이기 때문이다. 특히 큐잉 모델은 어느 자원을 모델링 하는데 있어 확률 분포를 이용한 수학적 접근이 용이하므로 컴퓨터 시스템 및 컴퓨터 네트워크 등의 성능 평가에 많이 이용되고 있다[22].

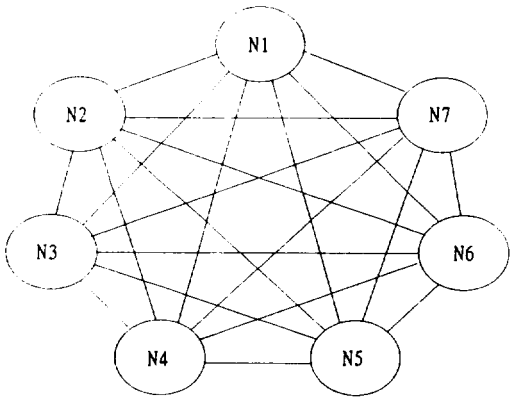
6.2 시뮬레이션 환경

본 논문에서 제시한 결함 허용 원격 프로시듀어 호출 메카니즘을 시뮬레이션 하기 위한 분산 시스템 모델은 노드가 7개로 구성된 7*(M/M/1) 큐잉 모델이라고 가정하였다. 또한, SLAM II 3.0이라는 시뮬레이션 툴을 이용하여 IBM PC 486 상에서 모의 실험 하였다[23, 24].

SLAM(Simulation Language for Alternative Modeling)이란 과정 지향적 접근 방법과 사건 지향적 접근 방법을 모두 포괄할 뿐만 아니라, 연속형 시뮬레이션까지도 포괄한다. 특히, 이산형 과정 지향적 접근 방법은 네트워크 모델에서 구현되므로 본 논문에서의 시뮬레이션 환경인 큐잉 모델에 대한 시뮬레이션으로 적합한 성질을 지니고 있다.

6.2.1 네트워크 위상

본 논문에서 시뮬레이션을 위한 네트워크 위상은 (Fig. 11)과 같다. 그림에서는 노드 7개가 있으며, 이들 모든 노드는 의사결정을 위한 똑같은 알고리즘 및 자료 구조를 가지고 있고, 제어(Control)는 완전 분산되어 있고, Service는 어느 노드에서나 모두 수행 가능하다고 가정한다. 또한, 각 노드의 처리기의 성능은 같다고 가정하고, 각 노드간의 통신상의 전송 에러도 없다고 가정하며, 통신 전용 프로세스가 존재한다고 가정한다.



(그림 11) 분산 환경의 네트워크 위상
(Fig. 11) Network Topology of distributed system

6.3 성능 평가를 위한 RPC 메카니즘

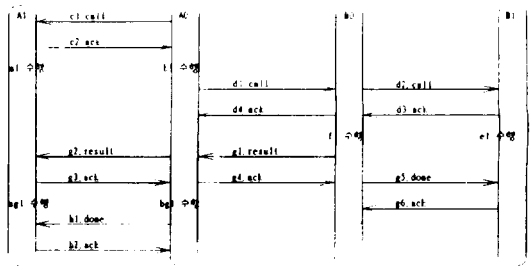
성능 평가를 위한 시나리오의 작성을 위해서 앞에서 제시했던 것보다 더욱 자세한 메카니즘이 (그림 12)와 (그림 13)에 제시되고 있다. 또한, 성능 평가를 위해 작성된 총 12개의 SLAM II 소스 프로그램 중에서 1개의 사본이 복제된 경우와 결함이 발생하는 경우 각각에 대해서 기존 연구의 메카니즘과 제안된 연구의 메카니즘의 경우에 적용된 2개의 프로그램이 부록에서 제시되고 있다.

성능 평가는 임의의 개체가 primary caller에서 발생하며, primary callee에게 call message를 전송할 때부터의 시간을 측정하여, callee 측에서의 done message 또는 ack message가 성공적으로 수행될 때까지의 시간을 측정하였다.

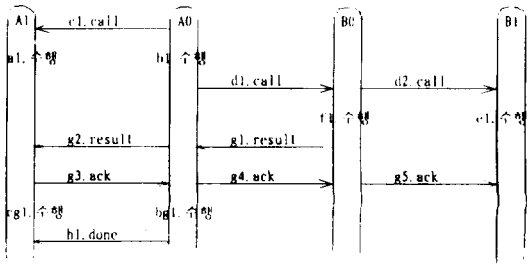
결함이 발생된 경우의 성능 평가에서는 원격 프로시듀어 호출과 관련된 프로시듀어 중 임의의 프로시듀어에 대하여 하나의 호출 과정 중에서의 임의의 시간에 결함이 발생하도록 하였다. 결함이 발생하는 시점과 발생하는 지점은 각각의 호출에 대하여 서로 다르기 때문에 확률적으로 타당한 결과를 얻을 수 있다.

사실, 성능 평가에 있어서 중요한 요인이 될 수 있는 부분은 Time Out 값을 설정하는 과정이다. 성능 평가의 척도인 호출 완료시간은 다른 프로시듀어의 상태를 체크하기 전에 ack message를 기다리는 시간인 Time Out에 크게 의존하기 때문이다. 이러한 사실은 실제 성능평가 과정에서 나타나는 현상이었다.

관련된 연구에서는 이를 사용자가 설정하도록



(그림 12) 기존의 연구에서의 메카니즘
(Fig. 12) Mechanism for the existing study



(그림 13) 본 논문에서 제안된 메카니즘
(Fig. 13) The mechanism proposed in this paper

지정하고 있다[11, 30]. 본 논문에서는 이를 규정할 필요가 있었기 때문에, 가장 객관적인 현상을 적용하기로 하였다. 즉, 기존 연구에서의 메카니즘에서는 call message의 전송 시간과 ack message의 전송 시간을, 본 논문에서 제안된 메카니즘에서는 call message의 전송 시간과 callee의 서비스 수행 시간과 result message의 전송 시간을 각각 관련지어 Time Out 값을 설정(기존의 메카니즘 : 4ms, 제안된 메카니즘 : 7ms)하였다.

(그림 12)와 (그림 13)은 각각 기존의 결함 허용 원격 프로시듀어 호출 기법과 본 연구에서 제안된 결함 허용 원격 프로시듀어 호출 기법에서의 메세지 전송 과정을 성능 평가를 위해 세부적으로 작성한 그림이다. 특히 위 그림에서는 한 개의 사본이 복제 되어 있음을 알 수 있다. 여기서, 결함을 탐지하기 위해 스택을 사용하는 기법과 chain을 등록하기 위해 chain의 정보를 네트워크의 모든 노드로 분산시키는 과정은 기존의 기법과 유사하므로 성능 평가에서는 고려치 않았다.

6.4 성능 평가 결과 및 분석

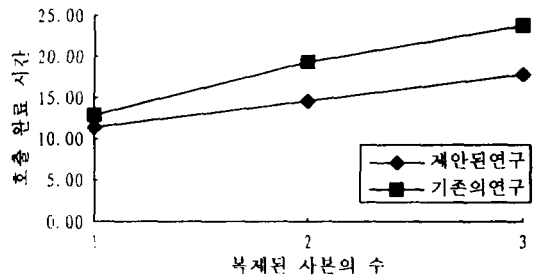
복제된 사본의 수를 1개, 2개, 3개로 서로 다르게 하여, 제안된 연구의 결함 허용 원격 프로시듀어 호출 메카니즘과 기존 연구의 결함 허용 원격 프로시듀어 호출 메카니즘 각각에서의, 모두 정상인 경우와 임의의 하나에 결함이 발생한 경우로 나누어 성능 평가를 하였다. 또한, 총 12개의 소스 프로그램을 각각 100회 수행하여 정확한 수치를 얻은 것이며 이런 과정을 통해 〈표 1〉과 같은 결과를 얻었다.

〈표 1〉로 보아서, 제안된 연구의 기법은 기존

〈표 1〉 성능 평가 결과표
(Table 1) Result table of performance evaluation (ms)

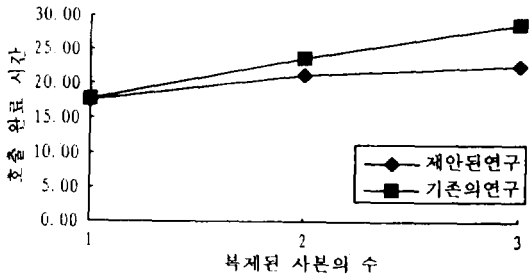
	제안된 연구의 메카니즘		기존의 연구의 메카니즘	
사본의 수	모두정상	결함발생	모두정상	결함발생
1	11.4	17.5	12.7	17.7
2	14.6	21.2	19.3	23.7
3	17.8	22.6	23.7	28.8

모두 정상인 경우



(그림 14) 모두 정상적인 경우
(Fig. 14) All normal cases

임의의 하나에 결함이 발생한 경우



(그림 15) 임의의 하나의 프로시듀어에 결함이 발생한 경우
(Fig. 15) Case in which fault happens in a arbitrary procedure

연구의 기법에서보다 모두 정상인 경우와 결함이 발생한 경우에서 모두 빠른 호출 완료 시간을 보이고 있음을 알 수 있다.

(그림 14)와 (그림 15)는 위의 결과를 그래프로 나타내 주고 있다.

앞의 (그림 14)와 (그림 15)를 보면 알 수 있듯이, 모두 정상인 경우와 임의의 하나의 프로시듀어에 결함이 발생한 경우에는 호출 완료 시간의 차이가 별로 없다. 또한, 기존 연구의 기법과 제안된 연구의 기법을 비교해 보면 복제된 사본의 수가 많을 수록 호출 완료 시간이 더 빠른 것으로 나타났다. 비록 본 논문에서는 복제된 사본의 수를 3개까지로 한정하였지만 그 이상의 사본이 복제되었을 경우, 기존의 연구 기법과 현저한 성능 차이를 예측할 수 있겠다.

7. 결 론

본 연구에서는 하드웨어 결함에 대한 결함 허

용 원격 프로시듀어 호출 기법을 제시하였다. 특히, 제한적으로 ack message를 사용함으로써 중복된 call message와 result message의 처리를 피하였으므로, 기존의 결함을 허용하는 원격 프로시듀어 호출에 비하여 효율적이고 신뢰성 있는 결함 허용을 제공하게 된다.

결함으로 발생된 Orphan을 효율적으로 처리하는 기법과 결함이 발생된 사본을 회복시킬 때, chain을 재배열시키는 효율적인 방법을 연구 중이다. 또한, 결함이 하나의 호출에 대해 중복되어 여러 번 발생할 경우에는 본 알고리즘을 적용하기 어려우며, 이에 대한 방안이 연구 중이다.

참 고 문 헌

[1] A.L.Ananda, B.H.Tay and E.K.Koh, "ASTRA-An Asynchronous Remote Procedure Call Facility", IEEE, 1991.

[2] A.D.Birrell and B.J.Nelson, "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems 2, 1 (Feb.1984), 39-59.

[3] Lan Cunningham, "Message-Handling Systems and Protocols", Proceedings of the IEEE Vol. 71, Dec. 1983, pp. 1425-1430.

[4] Coulouris and Doullimore, "Distributed systems : Concepts and Design", Addison Wesley Publishing, 1990.

[5] R.Emmerson and M,McGowan, "Fault Tolerance Achieved in VLSI", IEEE Micro, Vol. 4, No. 6, pp. 34-43, 1984.

[6] P.B.Gibbons, "A stub generator for Multilanguage RPC in Heterogeneous Environments", IEEE Trans. on Software Engineering, pp. 76-87, Jan. 1987.

[7] D.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.

[8] H.H.Huang, "Fault-Tolerant design of a modern receiving system", Proceedings of FTCS-10, pp. 375-378, 1980.

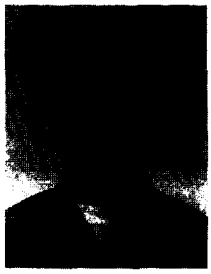
[9] B.W.Johnson, "Design and Analysis of Fault Tolerant", Digital Systems, Addison Wesley, 1989.

[10] M.F.Kaashoek and A.S.Tanenbaum, "Group Communication in the Amoeba Distributed Operating System", In proceedings of the Eleventh International Conference on Distributed Computer Systems, Pages 222-230. IEEE Computer Society, May 1991.

[11] Kiam S. Yap and Pankaj Jalote, and Satish Tripathi, "Fault tolerant remote procedure call", In the eighth International Conference on Distributed Computing Systems, pages 48-54, IEEE Computer Society, 1988.

구 용 완

중앙대학교 공과대학 전자계산학과 졸업 (학사)
중앙대학교 대학원 전자계산학과 졸업 (석사)
중앙대학교 대학원 전자계산학과 졸업 (박사)
현재 수원대학교 전자계산학과 교수겸, 전자계산 소장
관심분야 : Operating System을 근간으로 Distributed System과 System Software



한 석 진

수원대학교 전자계산학과 졸업 (학사)
수원대학교 대학원 전자계산학과 재학
현재 전자계산학과 연구 조교
관심분야 : Operating System, Computer Network, Distributed System 등