IMMORTAL : 원격 메쏘드 호출에 기반한 결함허용 분산 미들웨어 시스템

(IMMORTAL : Fault Tolerant Distributed Middleware System based on Remote Method Invocation)

현 무 용 † 김 식 †† 김 명 준 ††† 야마키다 지로 ††††
(Mu Yong Hyun) (Shik Kim) (Myung Jun Kim) (Jiro Yamakita)

요 약 분산 시스템을 지원하기 위한 패러다임으로서 분산객체 기술이 각광받고 있다. DSOM, DCOM, CORBA, Java RMI 등으로 대표되는 분산 미들웨어 플랫폼들은 분산 어플리케이션의 개발을 용이하게 하지만, 어플리케이션들의 신뢰성 및 가용성을 증진시키기 위한 직접적인 지원은 미흡한 상태이다. 분산 객체 패러다임을 지원하기 위한 결함 허용 기술의 개발 작업은 상당히 복잡하며, 오류가 발생할 소지가 높기 때문에, 분산 객체의 신뢰성과 가용성을 지원하는 개발툴에 대한 요구가 급증하고 있는 실정이다. 본 논문에서는 RMI에 기반한 결함허용 분산 미들웨어 시스템인 IMMORTAL을 제안하고자 한다. 제안된 시스템은 신뢰성 있는 분산 컴퓨팅을 지원하기 위해 로그 기반 롤백 복구 메커니즘을 채택하였다. 일련의 실험을 통해 IMMORTAL 하에서 동작중인 실험용 어플리케이션들이 다양한 하드웨어 및 소프트웨어 결함에도 불구하고 지속적으로 동작함을 확인하였고, 제안된 시스템의 성능 및 비례확장성을 평가하였다.

키워드 : 분산 시스템, 분산 프로그래밍 환경, 결함 허용 시스템, CORBA, RMI

Abstract Distributed object technologies have become popular in developing distributed systems. Although such middleware platforms as DSOM, DCOM, CORBA and Java RMI ease the development of distributed applications, they do not directly improve the reliability and the availability of these applications. Because the task of developing fault-tolerance techniques for distributed object paradigms is often complicated and error-prone, there is a great need for a development toolkit that enhances the reliability and the availability of distributed objects. In this paper, we propose a fault-tolerant distributed middleware system based on RMI, called IMMORTAL. We use a log-based rollback-recovery mechanism for supporting reliable distributed computing. Through a series of experiments, we observe that benchmark applications on the IMMORTAL tolerate hardware and software failures and evaluate its performance and scalability.

Key words : Distributed Systems, Distributed Programming Environment, Fault-tolerant system, CORBA, RMI

1. 서 론

최근 응용 프로그램들이 보다 복잡해지고 분산 컴퓨팅을 지향함에 따라, 이를 지원하기 위한 성공적인 패러다임으로서 분산 객체 기술이 각광받고 있다. OMG의 CORBA[1]와 Java RMI[2]는 분산 객체 프레임워크(framework)를 지원하는 대표적인 미들웨어들이다.

CORBA는 OMG에 의해 제안된 분산 객체 컴퓨팅의 표준 규약이며, 원격 시스템에 존재하는 객체의 메쏘드를 마치 로컬 객체의 메쏘드 처럼 호출하게 하는 하부 구조를 제공한다. 또한, 서로 상이한 프로그래밍 언어나 플랫폼 하에서 개발된 객체들 사이의 상호운영성(interoperability)을 보장한다. Java RMI는 자바 플렛폼(platform) 상에서 클라이언트-서버 모델에 기반한 분산

† 정 회 원 : 대원과학대학 컴퓨터정보통신과 교수
myhyun@daewon.ac.kr
†† 비 회 원 : 세명대학교 전자정보통신학부 교수
shikm@telcom.semyung.ac.kr
††† 비 회 원 : 충북대학교 전기전자및컴퓨터공학부 교수
mjkim@cbucc.chungbuk.ac.kr
†††† 비 회 원 : 오카야마 현립대학교 통신공학과 교수
yamakita@c.oka-pu.ac.jp
논문접수 : 2002년 3월 7일
심사완료 : 2002년 5월 27일

어플리케이션의 개발을 지원한다. 기본 아이디어는 객체를 생성할 때 내부에 정의된 메쏘드를 다른 자바 가상 머신에서 호출할 수 있도록 허용하는 것으로서, 이러한 접근 방법은 자바 객체를 위한 원격 함수 호출(RPC) 메커니즘을 제공한다[3].

그러나, 이러한 분산 미들웨어 플랫폼(platform)들은 분산 응용프로그램의 개발을 용이하게 하고 개발된 소프트웨어 컴포넌트의 이식성(portability), 상호운영성(interoperability) 및 재사용성(reusability)을 향상시켜 주지만, 결함 허용 기능을 지원하지 않음으로서 신뢰성이 보장된 객체 기반 분산 응용프로그램의 설계 및 구현을 복잡하게 한다. 또한, 프로그램 개발자에 의한 결함 허용 응용프로그램의 설계 및 구현은 상당히 지루한 작업일 뿐만 아니라 오류 발생의 가능성을 내포하게 된다. 따라서, 분산 객체의 신뢰성 및 가용성을 지원하기 위한 개발툴(development toolkit)에 대한 요구가 급증하고 있으며, 현재 분산 객체 기반 응용 프로그램들의 신뢰성을 향상시키기 위한 다양한 노력들이 진행 중에 있다[4-12].

이 논문에서는 IMMORTAL 시스템을 제안하고 그 설계 및 구현 방안에 대해 논의하고자 한다. IMMORTAL은 결함 허용을 위해 투명한 롤백 복구(rollback recovery) 메커니즘을 채택한 분산 미들웨어 시스템이다. 이 논문에서 제안된 결함 허용 서비스는 RMI 메커니즘을 채택한 HORB[13]에 기반을 두고 있다. RMI 메커니즘은 자바 클래스 사이의 최적화 된 통신 프로토콜을 제공할 뿐만 아니라, 중·소규모 응용프로그램에 적합한 접근방식이다. 또한, 인터넷 기반 분산 어플리케이션 개발자들이 직면하고 있는 많은 문제점들에 대한 유용한 해결방안으로 인식되고 있다[14]. HORB는 경량화된 RMI 메커니즘에 기반한 분산 미들웨어이며, 사용이 간편하고 JavaSoft의 RMI 시스템보다 2~3배 빠른 성능을 자랑한다[13].

이 논문의 구성은 다음과 같다. 2장에서는 결함 허용 분산 미들웨어 시스템에 관련된 기존 연구들에 대해 논의하며, 3장에서는 제안된 시스템의 기본 구조, 주요 컴포넌트 및 동작 메커니즘에 대해 기술한다. 4장에서는 기존 시스템들과의 비교 분석을 통해 제안된 시스템의 성능 및 비례확장성을 평가하였으며, 5장에서 결론을 맺는다.

2. 관련 연구

분산 객체의 신뢰성을 향상시키기 위한 다양한 노력들은 크게 통합 방식(integration approach), 가로채기 방식(interception approach), 서비스 제공 방식(service approach)으로 분류될 수 있다[6].

통합 방식은 ORB를 현존하는 그룹 통신 서브 시스템과 결합하는 형태로서, Electra[4]와 Orbix+Isis[5]가 대표적인 시스템이다. 이 접근방식은 모든 그룹 관리 기능들을 하나의 비 표준 컴포넌트에 독립시켜 유지함으로써, 표준 규약과의 호환성 유지가 용이하다. 하지만, ORB의 기본구조와 언어 매핑(language mapping)에 대한 변경을 요구하므로, 개발된 서버 및 클라이언트 응용프로그램의 이식성을 떨어뜨리는 단점을 가진다.

가로채기 방식은 ORB에 의해 제기된 메시지들을 가로채어 객체 그룹의 모든 멤버들에게 전달하는 방식이다. Eternal 시스템[6]은 이 방식을 사용하여 CORBA 어플리케이션에게 투명한 결함 허용 기능을 제공한다. 이 접근 방식은 ORB 기본 구조에 대한 변경을 요구하지는 않지만, 클라이언트의 요청을 가로채기 위한 운영체제 종속적인(OS-specific) 메커니즘에 의존하고 있다.

서비스 방식은 결함 허용 기능을 위한 다양한 서비스 객체 및 인터페이스들을 제공하며, 대표적인 시스템으로 Filterfresh[9], JavaGroups[10], OGS[11] 등이 있다. 이 방식에 따르면, 추가되는 새로운 기능들이 ORB와는 독립적으로 존재하는 서비스 객체의 집합 형태로서 제공되므로, ORB 기본 구조에 대한 변경이 필요하지 않고 표준 규약과의 호환성 유지에도 용이하다. 그러나, 클라이언트 및 서버 객체가 결함 허용 기능을 가지기 위해서는 제공되는 서비스 객체 및 인터페이스들을 명시적으로 포함해야 하므로, 결함 허용 기능을 위한 투명성 보장이 저해된다.

위에서 언급된 연구들의 대부분은 신뢰성 있는 그룹 통신에 기반한 능동 복제(active replication) 메커니즘에 초점을 맞추고 있으며, 높은 등급의 가용성과 신뢰성이 요구되는 mission-critical 응용프로그램에 적합한 방식이다. 그러나, 분산 컴퓨팅이 일반화됨에 따라, non mission-critical 응용프로그램의 수가 증가추세에 있다. 예를 들면, 슈퍼컴퓨터에서 동작하는 병렬 응용프로그램들은 분산된 서버 클러스터 상에서 수행될 수 있다. 이러한 응용프로그램들에 있어서, 결함 허용 기능은 필수적이지만, 제한된 자원 및 수행 시간을 사용하여 결함 허용 연산에 수반되는 오버헤드를 줄이는 방향으로 제공되는 것이 바람직하다[15]. 또한, 응용 프로그램은 실행 환경이나 가용성 관련 요구에 따라 복제 방식을 선택할 필요가 있다. 따라서, 실행 시간에 응용 프로그램에 의해 사용될 복제 방식 및 결함 허용 서비스의 등급을 선택할 수 있도록 허용하는 유연한 메커니즘이 필요하다.

DOORS[8]와 FT_HORB[12]는 신뢰성 있는 분산 컴퓨팅을 위해 메시지 로깅에 기반한 롤백 복구 메커니즘을 채택한 결함 허용 분산 미들웨어 시스템이다. DOORS는 서버 객체의 복제, 결함 탐지 및 복구 기능을 기존의 ORB 상에서 동작하는 서비스 객체 형태로 제공함으로써 CORBA 객체의 결함 허용 동작을 지원하고 있다. 그러나, DOORS가 채택한 복제 관리자(Replica Manager)는 모든 서버 객체의 복제관리를 총괄하는 중앙 집중식으로 설계되어 있어서, 빈번한 메시지 로깅 요구가 발생하는 경우 통신상의 병목 현상을 초래한다. 또한, 서비스 객체들을 위한 IDL 인터페이스들을 제공하고 있어서, 서비스 객체가 신뢰성 있는 분산 컴퓨팅을 지원하기 위해서는 반드시 이 인터페이스들을 상속해야만 한다. 이 제약은 언어의 객체모델과 IDL의 객체모델은 서로 상이하다는 점을 고려할 때, 프로그래머에게는 부담이 될 수 있다.

FT_HORB는 RMI에 기반한 결함허용 분산 프로그래밍 개발 환경이다. FT_HORB는 결함 허용 분산 컴퓨팅을 위해 체크포인트와 롤백 복구 메커니즘을 채택하여 자바언어를 확장한 형태로 설계되었으며, 최소한의 제약으로 자바 프로그램의 신뢰성 있는 분산 컴퓨팅 동작을 지원하고 있다. 그러나, FT_HORB는 몇 가지 중요한 결점을 내포하고 있다. 첫째, FT_HORB의 주요 컴포넌트 중의 하나인 Log Manager는 자체 결함이 발생할 경우에 대한 고려가 결여되어 있고, 중앙 집중식으로 설계되어 있어서 심각한 병목현상을 초래한다. 둘째, 일반적으로 분산 시스템에서의 결함모델은 크게 통신 결함(communication faults)과 프로세스 결함(process faults)으로 분류할 수 있다[16]. 그러나, FT_HORB의 결함 탐지 및 복구 메커니즘은 프로세스 결함에 대한 처리에만 집중되어 있어서, 통신 결함에 기인한 서버 객체의 결함 복구는 불가능하다.

본 논문에서는 메시지 로깅 및 롤백 복구 메커니즘에 기반하여 투명한 결함 허용 연산을 지원하는 IMMORTAL 시스템을 제안하고자 한다. 제안된 시스템은 이미 언급된 FT_HORB의 문제점을 개선한 형태로 설계되고 구현되었다. 로그 매니저의 결함 및 병목현상을 해결하기 위해 전송자 기반 메시지 로깅(Sender Based Message Logging) 메커니즘[17]을 도입하였고, Fault Watcher 컴포넌트를 추가하여 통신결함에 대한 탐지 및 복구가 가능하도록 하였다. 제안된 시스템의 주요 설계 목표는 아래와 같이 요약될 수 있다.

- 일관성(consistency) 있는 객체 복제 메커니즘 지원을 통한 자바 프로그램의 투명한 결함허용 분산 실행
- 자바 문법의 추가적인 변경 불필요
- IMMORTAL 지원 하에 개발된 응용 프로그램의 상호 이식성 보장
- IMMORTAL 시스템의 상호 이식성 보장

3. 시스템의 기본 구조

이 장에서는 IMMORTAL 시스템의 기본 구조 및 주요 컴포넌트에 대해 알아보고, 서버 객체의 신뢰성과 가용성을 보장하기 위한 기본 메커니즘에 대해 기술하기로 한다.

3.1 주요 컴포넌트

분산 객체의 신뢰성을 지원하는 표준 규약 제정에 대한 많은 요구들이 제기되고 있는 실정이다. CORBA의 경우 서버 객체의 결함 허용을 지원하기 위한 표준 규약에 대한 기본 골격이 완성되었고[18], Java RMI의 경우는 표준안을 위한 활발한 논의들이 진행 중에 있다[19].

결함 허용 관련 표준 규약에 따르면, 결함 허용 모드는 크게 수동(*Passive*)과 능동(*Active*)의 두 가지 타입으로 분류된다. 수동 타입은 서버 객체에 대한 두 가지 이상의 복사본으로 구성되어 있고, 그 중에 하나를 주 객체(primary object)라 칭하며, 클라이언트의 서비스 요청에 응답하는 역할을 담당한다. 주 객체를 제외한 나머지 객체들은 cold standby나 warm standby 모드로 대기하게 된다. 능통 타입은 서버 객체에 대한 두 개 이상의 복사본으로 구성되며, 각각은 신뢰성 있는 그룹통신[6]의 지원 하에 클라이언트의 요청들을 처리한다. 현재, 제안된 IMMORTAL 시스템은 수동 타입의 결함허용 서비스만 지원한다.

IMMORTAL 시스템은 운영 환경상의 몇 가지 가정하에 설계되었다. 첫째, 제안된 시스템은 fail-stop 모델 [20]을 따른다고 가정하였다. 이는 모든 서버 객체들은 결함 발생시에, 적절하지 못한 메시지의 추가 발생 없이 즉시 종료됨을 의미한다. 둘째, 대기중인 객체들은 결함이 발생한 후에 주 서버 객체와의 일관성 유지를 위해 체크포인트 복구 메커니즘을 사용한다. 이는 대기중인 객체에 대한 모든 메쏘드 호출은 *piecewise determinism*[20]이 보장되어야 함을 의미한다. 한 객체가 *piecewise deterministic* 하다는 것은, 자신의 상태는 호출되어지는 일련의 오퍼레이션들의 기능에 의해서만 변경 가능하고, 특정 상태에서 동일한 오퍼레이션을 반복시켰을 때, 그 결과로 도달되는 새로운 상태는 항상 동일하다는 것을 의미한다. 이는 관련연구 분야의 일반적인 가정에 속한다[12]. 셋째, 클라이언트의 원격 함수 호출은 내재되지(nested) 않는다고 가정한다. 즉, 호출되는 서버 객체는 다른 서버 객체를 호

출할 수 없다. 이는 클라이언트의 결함에 기인한 서버 객체 결함은 발생하지 않음을 의미하므로, 클라이언트의 결함 탐지는 불필요하다.

그림 1은 제안된 시스템의 결함 허용 구조는 HORB ORB에 기반한 소프트웨어 계층으로 설계되었음을 보여주고 있다. 결함 허용 스텁(stub) 생성기는 서버 객체를 정의하는 소스 프로그램을 입력으로 받아 결함 허용 프락시(proxy)와 결함 허용 스켈톤(skeleton) 스텁 루틴을 자동 생성한다. 프로그램의 실행 도중, 클라이언트 객체와 주 서버 객체 사이의 바인딩이 이루어지면, 클라이언트 객체와 주 객체는 자신의 결함 허용 스텁 루틴을 통해 결함 탐지 및 복구 초기화를 담당하는 Fault Watcher에 등록된다.

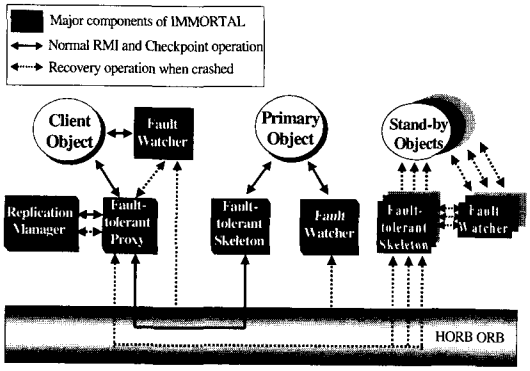


그림 1 IMMORTAL 시스템의 기본구조

그림 2는 Fault Watcher의 인터페이스 정보를 나타내고 있다. Fault Watcher는 결함 탐지에 관련된 기본 서비스들을 제공하며, 클라이언트 측에서 동작하는 모듈과 서버 측에서 동작하는 모듈로 구성된다. 서버 측에서 동작하는 Fault Watcher는 주 객체의 등록(registration) 및 결함 발생 여부를 감시하기 위한 함수들을 제공한다. Fault Watcher 실행 초기에 **Initialization()**이 호출되며, 다양한 환경 설정 작업들을 수행하게 된다. 주 객체를 Fault Watcher에 등록하는 작업은 주 객체의 결함 허용 스켈톤 루틴에서 **registerObject()**를 호출함으로서 이루어지며, 일단 주 객체가 등록되면, Fault Watcher는 자신의 **isAlive()** 루틴을 실행하여 주기적으로 주 객체에게 ping 메시지를 전송함으로써 주 객체의 결함여부를 감시하게 된다. 주 객체에 대한 ping 동작이 실패하면, 서버 객체에 결함이 발생한 것으로 간주되며, **callback NotifyFailure()**를 호출하여 클라이언트 측에서 실행 중인 Fault Watcher를 통해 결함 허용 프락시에게 통지된다.

```
interface FaultWatcher {
  public void Initialization();
  public void registerObject(Object obj);
  public boolean isAlive();
  public void callbackNotifyFailure();
  ...
}
```

그림 2 Fault Watcher의 인터페이스 정보

클라이언트 측에서 동작하는 Fault Watcher는 서버 측 Fault Watcher와 동일한 함수들을 제공한다. 그러나, 클라이언트의 동작을 감시하는 대신에, 서버 측 Fault Watcher에게 주기적으로 ping message를 전송하여 서버 측의 호스트 결함, 서버 측 Fault Watcher 결함 및 통신상의 링크 결함을 탐지하게 된다. 제한된 시간 내에 서버 측 Fault Watcher로부터 응답이 도착하지 않을 경우, 결함이 발생한 것으로 간주, 적절한 결함 복구 동작을 실행하게 된다. 결함 복구에 관련된 동작들은 서버 객체에 결함이 발생한 경우와 동일하다.

복제 관리자(Replication Manager)는 전송자 기반 메시지 로깅 메커니즘에 기반하여 설계되고 구현되었다. 기본 아이디어는 클라이언트와 주 서버 객체사이에 교환되는 모든 메시지를 유지하기 위한 저장소를 전송 측 임시 메모리 공간에 유지하는 것이다. Cold standby 모드에서 복제 관리자는 마지막 체크포인트 이후의 RMI 메시지 및 체크포인트가 발생한 시점에서의 주 서버 객체 상태를 유지한다. Warm standby 모드에서는 마지막 체크포인트 이후의 RMI 메시지들만 저장한다.

3.2 복제 관리

객체 지향 프레임워크(framework)에서의 결함 허용 서비스는 객체 복제를 통해서 제공되는 것이 일반적이다. 객체 복제의 목적은 서비스 객체에 대한 동일한 복사본을 추가로 제공함으로써 결함이 발생한 뒤에도 그 서비스를 계속 진행할 수 있도록 하는 것이다. 따라서, 복제된 객체간의 일관성 유지 및 결정적인(deterministic) 동작이 중요하다. 이 절에서는 객체 복제를 통한 결함 허용 서비스를 제공하기 위해서 본 논문에서 채택하고 있는 설계 개념 및 기본 메커니즘에 대해 기술하기로 한다.

3.2.1 복제 방식

일관성 있는 객체 복사본 유지를 위해 필요한 메커니즘은 복제 방식에 따라 다양하다. 결함 허용 관련 표준안에 따르면, 복제 방식은 크게 능동 복제와 수동 복제로 분류될 수 있다. 능동 복제 방식 하에서는, 대기 중인 모든 복사본들이 클라이언트의 원격 함수 호출(remote

method invocation)에 응답하게 된다. 수동 복제 방식 하에서는, 복사본들 중의 하나를 주 객체로 정하고, 모든 클라이언트 요청에 대한 처리를 전담하며, 나머지 복사본들은 대기상태에 위치하게 된다. 주 객체에 결함이 발생하면, 대기중인 복사본 중의 하나가 새로운 주 객체로 선정되며, 적절한 복구 과정을 거쳐서 결함이 발생한 주 객체의 동작을 대신하게 된다. 3.1절에 기술된 바와 같이, IMMORTAL 시스템은 수동복제 방식 중심으로 설계되고 구현되었으며, 이 복제방식은 대기중인 복사본들이 주 객체와 동기화되는 정도에 따라 cold standby 모드와 warm standby 모드로 세분될 수 있다.

Warm standby 모드에서 클라이언트는 주 객체와 통신을 하며, 체크포인트가 요구될 때마다 독립적인 통신 연결을 통해서 대기중인 warm standby들의 내부상태를 갱신한다. 체크포인트 이 후의 모든 RMI 메시지들은 복제 관리자에 저장된다. Cold standby에서는 주 객체의 결함이 발생했을 경우 복제 관리자에 저장된 RMI 메시지들과 주 객체의 내부 상태를 이용하여 복구절차를 수행한다. 주 객체에 결함이 발생하면, warm standby 혹은 *cold standby* 모드로 대기중인 복사본 중의 하나가 자동적으로 주 객체의 역할을 대신하게 된다. 일반적인 형태의 모든 하드웨어 및 소프트웨어 결함은 위에서 설명한 두 가지 결함 허용 모드로 처리 가능하다.

3.2.2 기본 메커니즘

IMMORTAL의 설계 철학은 객체 복제와 관련된 복잡한 쟁점들에 대한 투명성(transparency) 보장이다. 수동 타입 결함 허용 서비스의 두 가지 기본 모드는 IMMORTAL내의 결함 허용 프락시와 스켈톤 스텁에 의해 지원된다.

그림 3는 클라이언트가 서버 객체의 원격 함수를 호출할 때 일어나는 기본 동작을 보여주고 있다. 클라이언트 측 결함 허용 프락시는 초기화 과정에서 복제 관리자의 인스턴스를 생성하여 동일한 주소 공간에 유지한 뒤, 서버 측에 연결(bind)요청을 하게 된다. 클라이언트 측으로부터 연결요청을 접수한 서버 측은 주 서버 객체의 인스턴스 생성 등이 포함된 서버 객체 초기화 과정을 실행하게 된다. 그림이 예시하는 바와 같이, 결함 허용 프락시는 RMI 메시지를 주 서버 객체와 복제 관리자에게 병행적으로 전송하므로, 복제 관리자는 복제되는 객체의 일관성 보장을 위해서 전달받은 메시지를 정렬하고, 중복된 메시지를 처리할 수 있는 메커니즘을 제공해야 한다. 복제 관리자 측으로 전달된 RMI 메시지는 복제 관리자가 관리하는 저장소에 저장되며, 이러한 처리과정들은 서버 객체와 복제 관리자의 동기적(synchronous)인 관리를 통해 이루어진다. 이 후의 원격 함수 호출도 동일한 절차에 의해 진행이 된다.

그림 4, 5는 cold standby 와 warm standby 모드에서 체크포인트가 호출되었을 경우의 동작을 나타낸다. 그림내의 도식들은 두 가지 연속적인 동작, 즉, 체크포인트 관련 동작과 체크포인트가 성공적으로 수행된 이후의 일반적인 동작을 보여주고 있으며, *cold standby* 와 warm standby 모드간의 차이점은 이 두 동작으로 예시된다.

Cold standby 모드에서는, 체크포인트 요청이 발생할 때마다 클라이언트는 주 서버 객체와 통신하여 서버 객체의 내부 상태를 복제 관리자로 전송한다. 복제 관리자는 전송된 서버 객체의 상태를 이용, 유지되고 있는 서버 상태를 갱신한 뒤 저장중인 모든 RMI 메시지들을 삭제한다. Warm standby 모드에서는, 체크 포인트 요청이 발생하면 주 서버 객체는 자신의 내부 상태를 클라이언트로 전송하며, 클라이언트는 전송 받은 서버 객체 상태를 대기중인 나머지 서버 객체들에게 비동기적인

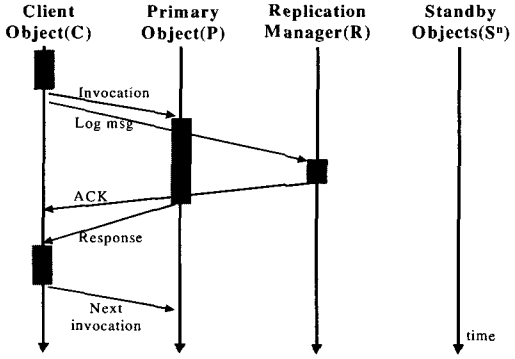


그림 3 일반적인 원격 함수 호출 동작

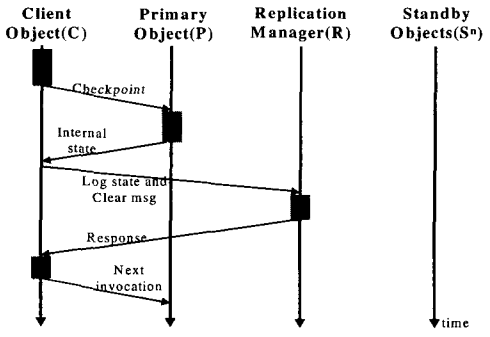


그림 4 Cold standby 모드에서 클라이언트 객체에 체크포인트가 요청되었을 경우

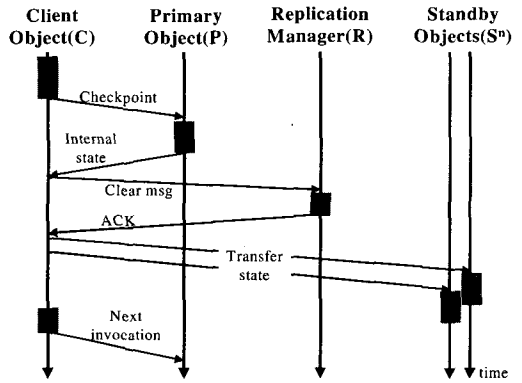


그림 5 Warm standby 모드에서 클라이언트 객체에 체크포인트가 요청되었을 경우

(asynchronous) 방법으로 전달한 뒤, 복제 관리자에 저장되어 있는 모든 메시지들을 삭제한다. 두 모드의 차이점을 요약하면 다음과 같다.

- Cold standby 모드에서는 주 서버 객체의 내부 상태가 복제 관리자에 의해 유지되지만, warm standby 모드에서는 대기 중인 나머지 서버 객체들로 전송된다.
- Cold standby 모드에서는 클라이언트와 주 서버 객체 사이에 동기적인 연결 설정이 이루어지는 반면, warm standby 모드에서 주 서버 객체 상태를 대기중인 나머지 서버 객체로 전송하는 과정은 실행상의 오버헤드를 고려하여 HORB에 의해 지원되는 비동기적인 메커니즘으로 진행된다.

3.3 결함 탐지 및 복구 관리

결함 탐지 및 복구는 결함이 발생한 서버 객체로 향하는 클라이언트의 모든 요청들을 가로채어 동일한 서비스를 제공하기 위해 대기중인 다른 서버 객체로 전달함으로서 원격 서버 객체의 결함을 숨기는 일련의 과정들을 의미한다. 이를 위해 설계된 모델은 프로세서의 결함뿐만 아니라 클라이언트에 의한 원격 서버 객체의 접근을 불가능하게 하는 통신 결함도 처리 가능해야 한다. 결함 탐지 및 복구에 대한 투명성을 유지하기 위해서, 제안된 시스템의 결함 탐지 및 복구 메커니즘은 클라이언트 혹은 서버 측 응용 프로그램의 어떠한 수정도 불필요한 형태로 설계되었다. 이 절에서는, 제안된 시스템이 네트워크 상에서 발생하는 다양한 형태의 하드웨어/소프트웨어 결함들을 탐지하고 복구하기 위해 채택한 기본 메커니즘에 대해 설명하기로 한다.

3.3.1 주 객체의 결함 탐지 및 복구

결함 탐지 메커니즘은 Fault Watcher로부터 제공되는 기본 서비스에 의존한다. 그림 6은 주 서버 객체에 대한 결함 탐지 및 복구 절차를 설명하고 있다. 서버 측에서 동작 중인 Fault Watcher는 주기적인 폴링(polling)을 통해서 주 서버 객체의 결함을 감시한다. 주 서버 객체의 결함을 탐지한 Fault Watcher는 클라이언트 측에서 동작중인 Fault Watcher를 경유하여 클라이언트로 예외신호를 전송한다. 이 예외신호는 클라이언트로 직접 전달되기 전 결함 허용 프락시에 의해 처리되며, 결함 허용 프락시는 대기중인 서버 객체 중의 하나를 활성화한 뒤, 결함 허용 모드에 따른 적절한 복구 작업을 수행하게 한다. Cold standby 모드에서는, 대체(alternative) 주 객체는 복제 관리자에 유지중인 서버 객체의 상태를 이용, 자신의 상태를 갱신한 뒤 저장된 RMI 메시지들을 재 수행하여 결함이 발생한 지점까지의 서버 객체 상태를 복원하게 된다. 그러나, warm standby 모드에서의 대체 주 객체는 마지막 체크포인트 이후의 RMI 메시지에 대한 재 수행만 요구된다.

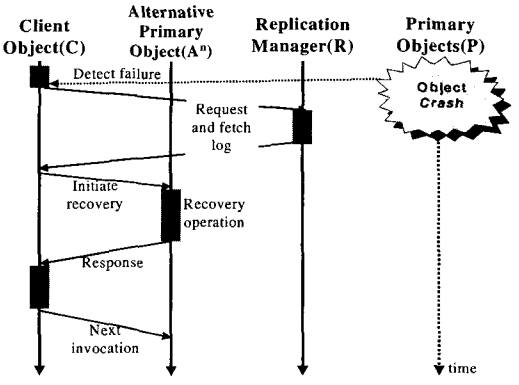


그림 6 주 객체에 결함이 발생했을 경우의 복구 동작

3.3.2 Fault Watcher의 결함 탐지 및 복구

그림 7은 서버 측에서 동작하는 Fault Watcher의 결함이나 통신 결함에 대한 탐지 및 복구 메커니즘을 보여주고 있다. 클라이언트 측에서 동작하는 Fault Watcher가 이를 위한 모든 처리과정을 담당한다. 클라이언트 측 Fault Watcher는 결함 탐지를 위해 서버 측 Fault Watcher로 주기적인 메시지를 전송한다. 제한된 시간 내에 응답 메시지가 도착하지 않을 경우, 결함이 발생한 것으로 간주, 클라이언트에게 적절한 복구 작업

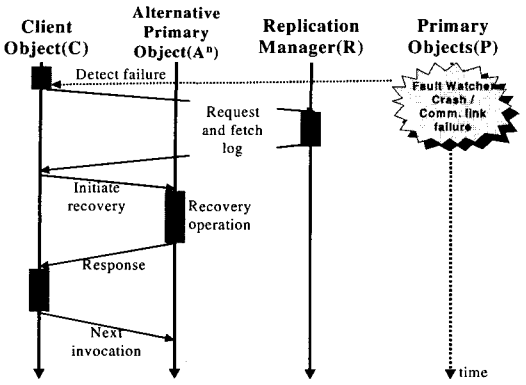


그림 7 *Fault Watcher* 결함 혹은 통신 결함이 발생했을 경우의 복구 동작

을 실행하도록 통지한다. 이 후의 결함 복구 처리는 주 객체의 결함 복구 과정과 동일하게 진행된다.

4. 실험 및 고찰

이 장에서는 일련의 실험들을 통해 IMMORTAL 시스템 하에서 생성된 응용 프로그램들이 다양한 하드웨어 및 소프트웨어의 결함에도 불구하고 지속적으로 동작하는지 여부를 검증하고, 제안된 IMMORTAL 시스템의 성능 분석 및 최적의 디자인 목표를 지원하기 위한 비례확장성(scalability)에 대해 논의하기로 한다.

4.1 실험 환경 및 모델

자바 언어(JDK 1.3)와 RMI 메커니즘을 기반으로 한 IMMORTAL은 결함 허용 분산 시스템을 위한 프로그래밍 환경으로서 설계되었고, 100Mbps 이더넷 네트워크에 연결된 SunUltra10(UltraSparc-IIi, 440 Mhz) 워크스테이션 상에서 구현되었다. IMMORTAL 환경 하에서 실험을 위해 구현된 응용 프로그램들이 여러 가지 형태의 하드웨어 및 소프트웨어 결함에도 불구하고 지속적으로 동작함을 확인하였다.

분산 응용프로그램의 전체 수행시간 중 통신이 차지하는 비율은 일부분에 불과하며 체크포인트와 결함 복구 동작은 아주 드물게 발생한다. 보다 실제적인 환경에서 메시지 저장의 오버헤드를 분석하기 위해, 본 논문에서는 ***n*-queens** 문제(n-queens problem), **tsp** 문제(traveling salesman problem), **gauss** 소거법(gaussian elimination)의 세 가지 응용프로그램을 IMMORTAL 환경 하에서 작성한 뒤 그 실행시간을 측정하였다.

이 세 가지 프로그램들을 실험 모델로 채택한 이유는 서로 상이한 통신 량과 통신 패턴 때문이다. ***n*-queens** 프로그램의 경우, 클라이언트 객체와 원격 서버 객체 사이의 통신은 프로그램 실행 시작 시점과 종료 시점에서만 발생하며, 원격 서버 객체 사이에는 아무런 통신도 일어나지 않는다. ***n*-queens** 프로그램의 전체 통신 량은 문제 크기(problem size)에 관계없이 항상 일정하다.

tsp 프로그램에서는, 도시와 도시간의 경로로 구성된 지도(map)가 원격 서버 객체들에게 전송되며, 부분 문제(subproblem) 해결을 담당한 원격 서버 객체들은 문제 해결을 위한 탐색 도중 새로운 부분 문제를 요청하고 그 결과를 보고하기 위하여 클라이언트 객체와 통신한다. 부분 문제의 개수는 지도상의 도시 개수에 의존하므로, 전체 통신량은 n개의 도시에 대해서 $O(n)$이다. 문제 해결을 위해 사용된 **branch-and-bound** 알고리즘 특성상 실행시간은 입력된 지도에 의존한다. ***n*-queens** 프로그램과 마찬가지로 원격 서버 객체들 사이의 통신은 발생하지 않는다.

gauss 프로그램은 제시된 3가지 실험 모델 중 가장 많은 통신을 수행한다. 행렬의 열들은 프로그램의 시작 시점에서 클라이언트 객체로부터 원격 서버 객체들에게 전송되며, 종료시점에서 클라이언트 객체로 다시 수거된다. 실행의 각 단계에서 피봇(pivot)열은 클라이언트에 의해 결정되며, 결정된 피봇열은 모든 원격 서버 객체로 전송된다. 전체 통신량은 n x n 행렬에 대해서 $O(n^2)$이다.

제시된 세 가지 응용 프로그램들은 cold standby 와 warm standby 모드에서 고정된 집합의 문제 해결을 위해 사용되었다. 문제 해결은 메시지 로깅 및 체크포인트 메커니즘을 사용한 경우와 사용하지 않은 경우로 나뉘어 반복 시도되었다. **tsp** 프로그램을 위한 지도와 **gauss** 프로그램을 위한 행렬은, 각각, 정수형(int, 4 bytes) 및 배정도 실수형(double, 8 bytes)의 2차원 배열로서 정의하였으며, 임의적으로 생성되어 반복되는 실험을 위해 따로 저장되었다. 각 프로그램에서 제시된 문제는 적절한 파티션(partition) 단계를 거쳐, 네트웍에 연결된 8개의 독립적인 노드에서 대기중인 서버 객체들에게 분배되었다. 메시지 로깅 메커니즘을 사용한 실험의 경우, 클라이언트와 서버 객체간에 전달되는 모든 메시지들이 저장되었다.

4.2 결과 분석

표 1은 cold standby 모드에서 메시지 로깅의 오버헤드를 보여주고 있다. 체크 포인트의 오버헤드는 그 발생 빈도수에 크게 의존적이므로, 이 실험에서는 제외되었다. 표의 각 엔트리는 실험에 관련된 응용 프로그램 및 문제 크기를 의미한다. 각 프로그램의 해를 찾기 위해

표 1 IMMORTAL 환경하의 Cold standby mode에서의 메시지 로깅 오버헤드 (msec)

Program	Size	Message Logging		Overhead	
		with	without	time	percent(%)
n-queens	6	12.1	10.4	1.7	16.3
	7	23.1	21.5	1.6	7.4
	8	32.7	31	1.7	5.5
tsp	6	55.4	50.9	4.5	8.8
	7	873.2	824.5	48.7	5.9
	8	53627.6	51987	1640.6	3.2
gauss	30	668.9	623.5	45.4	7.3
	40	1146.7	1089.4	57.3	5.3
	50	2069.3	2001.4	67.9	3.4

요구되는 실행시간을, 메시지 로깅을 사용한 경우와 그렇지 않은 경우로 각각 나누어서 제시하였다. 표에 의하면, 각각의 응용 프로그램은 주어진 문제 크기에서 약 16 퍼센트에서 3 퍼센트 정도의 메시지 로깅 오버헤드를 보여주고 있다. 각 실험 모델에서, 문제 크기가 증가할수록 전체 실행시간에서 메시지 로깅이 차지하는 비율이 줄어듬을 알 수 있는데, 이는 문제 크기가 증가하면 메시지 전송 사이의 계산량이 함께 증가하므로 상대적으로 메시지 로깅의 오버헤드는 감소한다는 사실에 기인한다.

그림 8은 **gauss** 프로그램을 대상으로 체크 포인트 빈도 수를 다양하게 변경시키면서 실험한 결과를 보여주고 있다. X축은 체크포인트의 회수를, Y축은 실행시간을 의미한다. 실험은 cold standby 모드에서 문제크기를 8로 고정하여 실시되었다. 실험에서, 클라이언트는 주 서버 객체에 대해 80회의 원격 호출을 실행한다. 실험 결과는 체크포인트의 회수가 증가할수록 그 오버헤드도 비례적으로 증가하나, 포화현상(saturation)은 발생하지 않음을 보여주고 있다. 또한, 한 번의 체크 포인트

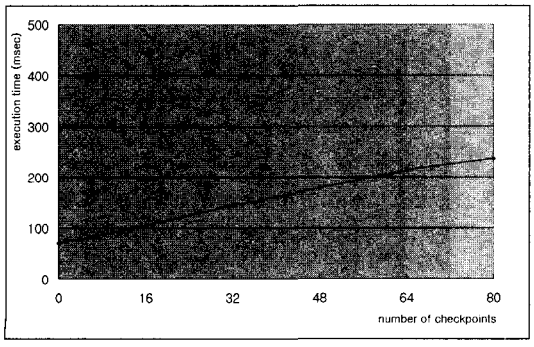


그림 8 체크포인트 변화에 따른 실행시간

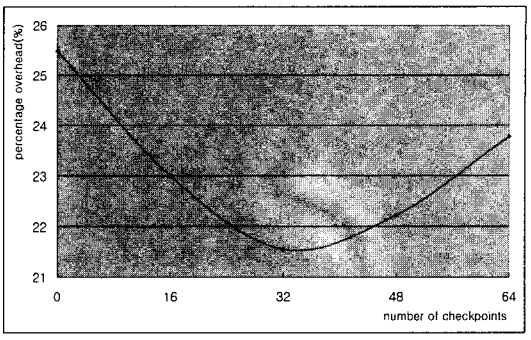


그림 9 체크포인트 변화에 따른 결함복구 오버헤드

가 전체 실행시간에서 차지하는 오버헤드는 평균 3 퍼센트(2.1 msec)로서 미세함을 알 수 있다. 따라서, 체크포인트 빈도수 증가로 인한 결함 복구 시간의 단축을 고려한다면, 메시지 로깅과 체크 포인트 메커니즘을 바탕으로 한 IMMORTAL은 결함 허용 분산 시스템의 개발 환경으로서 적절한 선택이 될 수 있음을 알 수 있다.

일반적으로 체크포인트의 횟수가 증가하면, 결함복구 비용은 줄어들지만 빈번한 체크포인트 실행에 기인한 전체 프로그램의 실행시간 증가를 유발한다. 따라서, 이러한 상관관계를 고려한 최적의 체크포인트 횟수를 선택하는 전략이 필요하다. 그림 9는 **gauss** 프로그램에 결함이 발생했을 경우, 체크포인트 회수에 따른 결함 복구 오버헤드를 측정한 결과이다. 결함복구 오버헤드란 프로그램의 전체 실행시간에서 결함복구에 소요되는 시간이 차지하는 비율을 퍼센트로 표현한 수치이다. X축은 체크포인트 회수를, Y축은 결함복구 오버헤드를 의미한다. 실험은 cold standby 모드에서 프로그램의 문제 크기를 8로 고정하여 실시되었으며, 결함발생 빈도를 일정하게 유지하였다. 실험 결과는 체크포인트 횟수가 32일 때 최적의 성능을 보여주고 있다. 체크포인트 횟수가 많을수록 결함복구 비용은 줄어들지만 전체 프로그램의 실행시간 증가를 유발하므로, 실험결과는 체크포인트 횟수와 결함복구 효율이 비례하지 않음을 나타내고 있다. 최적의 체크포인트 횟수는 응용 프로그램과 문제 크기에 의존적이므로 이를 고려한 적절한 선택이 필요하다.

그림 10은 **gauss** 프로그램을 warm standby 모드에서 standby 개수를 변화시키면서 실험한 결과이다. x축은 standby 개수를, y축은 실행시간을 의미하며, 실험을 위해 문제크기는 8, 체크포인트의 횟수는 32로 고정되었다. 그림 10에 의하면, 실행시간은 standby 개수에 비례하여 증가하고 있으나, 포화(saturation) 현상은 발생하지 않음을 알 수 있다. warm standby 모드가 제공하는

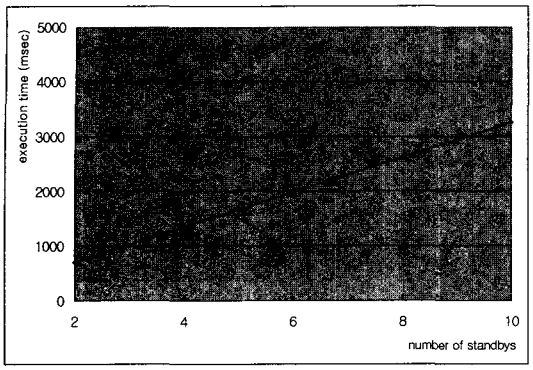


그림 10 Standby 개수 변화에 따른 실행시간

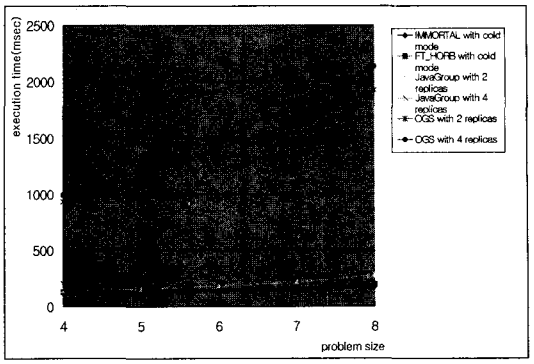


그림 11 신뢰성 있는 그룹 통신을 지원하는 시스템과의 성능비교(***n*-queens**의 경우)

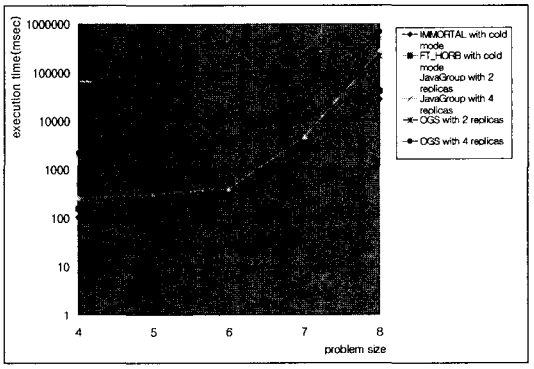


그림 12 신뢰성 있는 그룹 통신을 지원하는 시스템과의 성능비교(**tsp**의 경우)

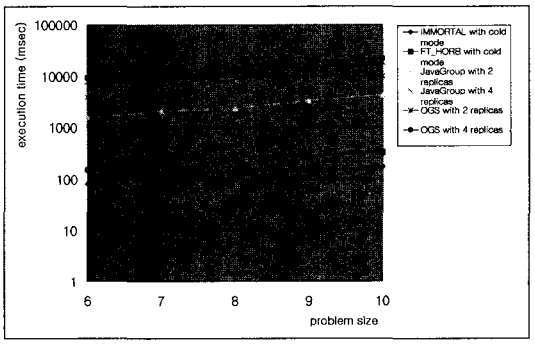


그림 13 신뢰성 있는 그룹 통신을 지원하는 시스템과의 성능비교(**gauss**의 경우)

빠른 결함 복구시간을 고려할 때, 실험결과는 warm standby 모드의 적절한 성능을 보여주고 있다. 응용프로그램이 warm standby 모드에서 적절한 성능을 발휘하기 위해서는, 분산 시스템에서의 파티션 개수를 고려하여 warm standby의 개수를 신중하게 결정해야 한다.

제안된 시스템이 non mission-critical 어플리케이션에 적합한 개발 환경임을 검증하기 위하여, 여러 가지 테스트 프로그램을 신뢰성 있는 그룹 통신을 지원하는 시스템 환경에서 각각 구현한 후 상호간의 성능을 비교하는 실험을 실시하였다. 그림 11, 12, 13은 ***n*-queens**, **tsp**, **gauss** 프로그램을 IMMORTAL, FT_HORB, Java Groups(version 1.0), OGS (verson 0.8b) 하에서 각각 구현 한 뒤 그 실행 시간을 측정한 결과이다, X 축은 문제크기를, Y축은 실행시간을 의미한다. IMMORTAL, FT_HORB 하에서의 성능측정은 cold standby mode에서 체크포인트 횟수를 32로 고정하여 실시되었다. OGS의 경우에는 Visibroker for Java(version 4.5) 지원하에 클라이언트 객체가 GroupAccess 객체에 접속하도록 하였으며, Untyped 서버로 동작하도록 서버 객체를 수정하였다. JavaGroups와 OGS의 경우 그룹에 소속된 구성원(replica)의 개수를 2개, 4개로 구분하여 실험하였다.

실험에 의하면, IMMORTAL 환경에서 구현된 ***n*-queens**, **tsp**, **gauss** 프로그램은 JavaGroup, OGS 환경에서 구현된 프로그램들보다 확연한 성능향상을 보여주었다. 또한, FT_HORB 환경에서 구현된 프로그램과 비교하여, 각각, 약 11%, 50%, 90%의 성능향상을 보여주고 있다. 이는 IMMORTAL에서 채택한 송신자 기반 메시지 로깅 메커니즘이 복제 관리자의 병목현상을 개선한 결과이며, 프로그램의 전체 통신량이 $O(c)$, $O(n)$, $O(n^2)$ 순으로 증가할수록 IMMORTAL 시스템이 채택한 메시지 로깅 매커니즘이 보다 향상된 성능을 발휘함을 입증하고 있다. 그림 11, 12, 13의 실험결과는 JavaGroups와 OGS에서 지원하고 있는 신뢰성 있는 그룹 통신이 유발하는 성능저하가 제안된 시스템에 비하여

매우 크며, 그룹에 소속된 구성원의 개수가 늘어날수록 그 오버헤드도 비례적으로 증가함을 보여주고 있다. 이는 비교적 낮은 등급의 신뢰성과 가용성을 요구하는 응용 프로그램의 경우에는 적절하지 못한 메커니즘임을 의미한다. 실험 결과는, 본 논문에서 제안된 IMMORTAL이 경량화된 결함 허용 메커니즘을 제공하고 있음을 증명하고 있다.

5. 결 론

응용 프로그램들이 보다 세련되고 분산 컴퓨팅을 지향함에 따라, 신뢰성 있는 분산 응용프로그램을 개발하기 위한 결함 허용 시스템 개발툴에 대한 요구가 급증하고 있다. 본 논문에서는 RMI를 기반으로 한 결함 허용 분산 미들웨어 시스템인 IMMORTAL을 제안하였다. IMMORTAL은 신뢰성 있는 분산 컴퓨팅을 지원하기 위해서 체크포인트와 롤백 복구 메커니즘을 이용하여 기존의 자바 언어를 확장한 형태로 설계되었으며, 결함 허용 스텁 생성, 복제 관리, 결함 탐지 및 복구를 위해서 자동 스텁 루틴 생성기, 복제 관리자, Fault Watcher 모듈이 추가되었다. 제안된 IMMORTAL은 JDK(version 1.3)와 HORB(version 2.0)을 기반으로 구현되었고, 현재, cold standby 와 warm standby의 두 가지 결함 허용 모드를 성공적으로 지원하고 있으며, IMMORTAL 환경 하에서 구현된 응용 프로그램들이 여러 가지 형태의 하드웨어 및 소프트웨어 결함에도 불구하고 지속적으로 동작함을 확인하였다.

일련의 실험을 통해서 IMMORTAL의 성능을 검증하였고, 최적의 디자인 목표를 지원하기 위한 비례 확장성을 제시하였다. 보다 실제적인 환경에서 메시지 로깅의 오버헤드를 분석하기 위해 ***n*-queens**, **tsp**, **gauss**의 3가지 실험 모델을 채택하여 그 실행시간을 측정한 결과, 대부분의 문제 크기에서 적절한 수준의 메시지 로깅 오버헤드를 보여주었다. 체크포인트 메커니즘 관련 실험에서는 메시지 로깅과 체크포인트 메커니즘이 정상적으로 동작함을 알 수 있었고, 체크포인트 횟수와 결함복구 오버헤드는 서로 반비례하지 않음을 보여 주었다. 응용프로그램이 warm standby 모드에서 적절한 성능을 발휘하기 위해서는, 프로그래머는 분산 시스템에서의 파티션 개수를 고려하여 warm standby의 개수를 신중하게 결정해야 함을 확인하였다. JavaGroups, OGS 등 신뢰성 있는 그룹 통신을 지원하는 시스템들과의 비교 실험에서 제안된 시스템은 비교적 낮은 수준의 오버헤드만으로 결함 허용 연산을 성공적으로 지원하였고, 낮은 등급의 신뢰성과 가용성을 요구하는 응용 프로그램에 적합한 시스템임을 확인하였다.

제안된 시스템은 client 측에 결함이 발생할 경우에는 결함 허용 기능을 제대로 수행하지 못한다는 결점을 가진다. 현재, 제안된 시스템은 client측 결함은 발생하지 않는다는 가정 하에 설계되고 구현되었으나, 이 가정을 보완하기 위한 추가 연구가 요구된다. 또한, 결함복구 효율을 고려한 최적의 체크포인트를 선택하기 위해서, 결함복구 오버헤드를 체크포인트 오버헤드와 롤백 오버헤드 등으로 세분화하여 평가하는 심도있는 연구가 필요하다.

참 고 문 헌

[1] J. Siegel, *CORBA Fundamentals and Programming*, John Wiley & Sons, 1996.

[2] Sun Microsystems, JDK 1.3 Documentation, <URL: http://www.javasoft.com/>

[3] M. Hyun, S. Kim, and S. Lee, *"Design and Implementation of Translation System between RMI to CORBA," Journal of IEEK*, Vol.36, No.2, 1999.

[4] S. Maffeis, *"Run-Time Support for Object-Oriented Distributed Programming,"* Ph.D Thesis, University of Zurich, Zurich, 1995.

[5] Isis Distributed Systems Inc. and Iona Technologies Limited. *Orbix+Isis Programmer's Guide*, 1995.

[6] P. Narasimhan, *"Transparent Fault Tolerance for CORBA,"* Ph.D thesis, University of California, Santa Barbara, CA, 1999.

[7] S. K. Shrivastava, G.N.Dixon and G. D. Parrington, *"An overview of the Arjuna distributed programming system," IEEE Software*, January, 1991.

[8] P. Chung, Y. Huang, S. Yajnik, D. Liang, and J. Shih, *"DOORS : Providing fault tolerance to CORBA objects," in poster session of Middleware'98*, September. 1998.

[9] A. Baratloo, P.E. Chung, Y. Huang, S. Rangarajan, and S. Yajnik., *"Filterfresh : Hot replication of Java RMI server objects," Proceedings of the Fourth USENIX Conference on Object-Oriented Technologies and Systems*, pp65-78, Santa Fe, NM, April, 1998.

[10] Mark Hayden, *The Ensemble System*. Technical Report 98-1662, Cornell University, January, 1998.

[11] P. Felber, R. Guerraoui, and A. Schiper., *"The implementation of a CORBA object group service," Theory and Practice of Object Systems*, Vol.4, No.2, pp.93-105, 1998.

[12] S. Kim, M. Hyun and J. Yamakita, *"FT_HORB :*

A Fault-Tolerant Distributed Programming Environment based on RMI," IEICE Trans. on Information and Systems, Vol.E85-D, No.3, pp.510-517, 2002

[13] HIRANO, H., Yasu, Y. and Igarashi, H., *"Performance Evaluation of Popular Distributed Object Technologies for Java," ACM Workshop on High-Performance Network Computing for Java*, 1998.

[14] Sanjav P. A and Renato Q., *"Performance Evaluation of Java RMI : A Distributed Object Architecture for Internet Based Applications," Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp565-569, August, 2000.

[15] S. Rao and H. Vin, *"Egida: An Extensible Toolkit For Low-overhead Fault-Tolerance," In Proceedings of the 29th Fault-tolerant Computing Symposium*, Madison, Wisconsin, pp.48-55, June, 1999.

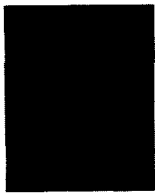
[16] P. Jalote, Fault Tolerance in Distributed Systems, pp51-52, Prentice-Hall, 1994.

[17] Jian Xu, R.H.B. Netzer, M. Mackey, *"Sender-based message logging for reducing rollback propagation," Proceedings of Seventh IEEE Symposium on Parallel and Distributed Processing*, pp602-609, 1995

[18] Object Management Group, Fault Tolerant CORBA Specification, OMG Technical Committee Document (ptc/2000-03-04), March, 2000.

[19] Sun Microsystems, Inc. Java Specification Request. JSR000117 : J2EE APIs for Continuous Availability, 2001.

[20] D. Liang, S.C. Chou and S.M. Yuan, *"A Fault-Tolerant Object Service in OMG's Object Management Architecture," Information and Software Technology*, Vol.39, pp.965-973, 1998.



현 무 용

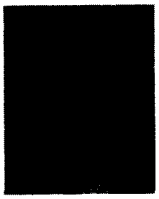
1993년 2월 경북대학교 컴퓨터 공학과 학사 졸업. 1995년 2월 경북대학교 대학원 컴퓨터 공학과 석사 졸업. 2002년 3월 충북대학교 대학원 전자계산학과 박사수료. 1995년 3월 ~ 1997년 2월 대원전문대학 전자계산과 전임강사. 1997년 3월 ~ 현재 대원과학대학 컴퓨터 정보통신과 조교수. 관심분야는 컴퓨터 네트워크, 분산 처리, 결함허용 시스템, Bluetooth technology

김 식

1979년 2월 경북대학교 전자공학과(전산전공) 학사 졸업. 1979년 ~ 1988년 국방과학연구소 선임연구원. 1990년 Texas A&M 전산학과 석사 졸업. 2002년 일본 오카야마 현립대학교 통신공학과 박사 졸업. 1993년 ~ 현재 세명대학교 전산정보통신학부 교수. 관심분야는 컴퓨터 네트워크, 분산 처리, 결함허용 시스템, Bluetooth technology

김 명 준

1979년 8월 서강대학교 수학과 학사 졸업. 1984년 6월 플로리다 공대 전자계산과 석사 졸업. 1992년 8월 텍사스 A&M 전자계산과 박사 졸업. 1993년 9월 ~ 현재 충북대학교 전기 전자 및 컴퓨터 공학부 부교수. 관심분야는 실시간 시스템, 운영체제, 분산운영체제



야마키다 지로

1969년 교토 공과대학 전자공학과 학사 졸업. 1971년 동대학원 석사졸업. 1979년 오사카 현립대학 전자공학과 박사 졸업. 1979년 ~ 현재 오카야마 현립대학교 통신공학과 교수. 관심분야는 Scattering of waves, Analysis of waveguide