論文2002-39CI-2-1

HORB에 기반한 신뢰성 있는 분산 프로그래밍 환경의 설계 및 구현

(Design and Implementation of Reliable Distributed Programming Environment based on HORB)

玄 武 庸 *, 金 湜 **, 金 明 俊 ***

(Mu Yong Hyun, Shik Kim, and Myung Jun Kim)

요 약

DSOM, DCOM, CORBA, Java RMI 같은 객체 지향 분산 프로그래밍 환경을 이용한 분산 응용 프로그램 개발이 일반화되고 있다. 그러나, 이러한 분산 미들웨어들은 응용프로그램의 품질과 재사용성을 향상시켜 주지만, 결함 허용 기능을 지원하지 않음으로서 신뢰성이 보장된 객체 기반 분산 응용프로그램의 설계 및 구현을 복잡하게 한다. 본 논문에서는 RMI 메커니즘을 기반으로 한 결함 허용 분산 시스템 개발 환경인 에버그린(Evergreen)을 제안하고자 한다. 에버그린은 신뢰성 있는 분산 컴퓨팅을 지원하기 위해서 체크포인트와 롤백 복구(rollback recovery) 메커니즘을 이용하여 설계되었다. 일련의 실험을 통해 에버그린의 성능을 평가하였고 최적의 디자인 목표를 지원하기 위한 확장 가능성을 확인하였다.

Abstract

The use of Object-Oriented Distributed Programming(OODP) environment such as DCOM, DSOM, Java RMI, CORBA to implement distributed applications is becoming increasingly popular. However, absence of a fault-tolerance feature in these middleware platforms complicates the design and implementation of reliable distributed object-based applications, although they greatly enhance the quality and reusability of the distributed object-based applications. In this paper, we propose a fault-tolerant programming environment based on RMI, namely *Evergreen*, for the reliable distributed computing with checkpoints and rollback-recovery mechanism. Based on a series of experiments, we evaluate the performance of *Evergreen* and find its possibility of extension to fully support our optimal design goal.

Key words : Distributed Systems, Distributed Programming Environment, Software fault-tolerance, CORBA, RMI

* 正會員, 大元科學大學 컴퓨터 情報通信科
(Dept. of Information and communication, Daewon Science College)
** 正會員, 世明大學校 情報通信學科
(Dept. of Information and Communication Systems, Semyung Univ.)
*** 正會員, 忠北大學校 컴퓨터科學科
(Dept. of Computer Science Chungbuk National Univ.)
接受日字:2001年4月18日, 수정완료일:2001年10月25日

Ⅰ. 서 론

응용 프로그램들이 보다 세련되고 분산 컴퓨팅을 지향함에 따라, 그 설계 및 구현 작업이 상당히 복잡해지는 추세이다. 따라서, 분산 응용 프로그램의 개발을 지원하기 위한 방안으로서. OMG의 CORBA[1], Microsoft의 DCOM[2], IBM의 DSOM[3], Java RMI[4] 등 여러 가지 분산 객체 미들웨어들이 제안되고 있다. 그러나, 이러한 분산 미들웨어 플랫폼(platform)들은 분산 응용프

로그램의 개발을 용이하게 하고 그 품질 및 재사용성(reusability)을 향상시켜 주지만, 결함 허용 기능을 지원하지 않음으로서 신뢰성이 보장된 객체 기반 분산 응용프로그램의 설계 및 구현을 복잡하게 한다. 따라서, 사용자 편의의 결함 허용 시스템 개발툴에 대한 요구가 급증하고 있다.

현재, 분산 객체 기반 응용프로그램의 신뢰성을 향상시키기 위한 노력들은 크게 시스템 접근 방법(System approach)과 객체 접근 방법(Object approach)으로 구분될 수 있다. 시스템 접근방법의 대표적인 예로서 Electra 시스템[5]이 있고, Arjuna[6], Phoinix[7], DOORS[8]는 객체 접근방법을 대표한다. Electra는 C++ 언어를 기반으로 한 신뢰성 있는 분산 시스템 구축을 지향하며 이를 위한 추상화 도구들의 집합을 제공하는 객체 지향 툴킷이다. 이 툴킷은 네트웍에 연결된 다른 노드에서 동작하며 다른 Electra C++ 객체와의 동기적 통신이 이루어지는 C++ 객체의 생성을 가능하게 한다. 그러나, ISIS[9], Horus[10], Chorus[11]와 같은 런 타임 시스템 상에서 동작하는 신뢰성 있는 그룹 통신 서비스의 지원이 필수적이다.

Phoinix와 DOORS는 OMA(Object Management Architecture)와 호환 가능한 결함 허용 분산 응용프로그램 개발 환경이다. Phoinix와 DOORS의 결함 허용 서비스는 크게 재시작(restart, level 1), 롤백 복구(rollback-recovery, level 2)의 두 수준으로 분류되며 결함 허용 능력은 서비스 수준이 증가할수록 향상된다. Electra와 Arjuna에 비해 Phoinix와 DOORS가 가지는 중요한 장점은 CORBA 표준 규약을 따르고 있어서, 최소한의 변경만으로도 대부분의 CORBA 호환 제품들로의 이식이 가능하다는 것이다. 그러나, 서비스 객체들을 위한 IDL 인터페이스들을 제공하며, 서비스 객체가 신뢰성 있는 분산 컴퓨팅을 지원하기 위해서는 반드시 이 인터페이스들을 상속해야 한다. 이 제약은 언어의 객체 모델과 IDL의 객체 모델은 서로 상이하다는 점을 고려할 때, 프로그래머에게는 부담이 될 수 있다.

Arjuna는 결함 허용 분산 응용프로그램 개발을 위한 도구들의 집합을 제공하는 객체지향 프로그래밍 시스템이다. Arjuna는 대부분의 객체 지향 언어에서 나타나는 여러 가지 특징들을 포함하고 있으며 기존 운영체제 시스템에서 일반적으로 나타나는 제한된 시스템 능력들만 요구한다. 단위 동작들(atomic action)은 C++ 객체의 인스턴스들로 구성된 지역 및 원격 객체들의 동작 순서들을 제어하며, 원격 객체 상의 오퍼레이션들은 원격 프로시쥬어 호출(RPC) 메커니즘을 통해서 호출된다. 그러나, Arjuna는 표준 규약과의 호환성이 결여되어 있다는 중요한 결점을 가진다.

분산 시스템을 개발하기 위한 접근 방안으로서 분산 컴퓨팅을 위한 새로운 언어를 설계하는 방법과 기존의 언어에 분산 프로그래밍을 위한 라이브러리를 추가하는 방법이 있으며, 이 두 가지 접근 방안 중 어느 쪽이 더 효율적인가에 관해서는 논란의 여지가 많다.[5] 멀티 프로세싱의 증가 및 프로세스 가격의 하락으로 인해, 멀티 쓰레딩을 위한 언어적 지원이 광범위하게 요구되고 있다. 오늘날 자바는 네트웍 컴퓨팅을 위한 가장 중요한 기술 중의 하나로 간주되고 있고, 객체 지향 언어이며 C++ 언어와 비교해 볼 때 비교적 단순한 문법구조를 가진다. 자바는 네트웍 컴퓨팅을 고려하여 설계된 언어지만, 그 자체로는 분산 컴퓨팅 및 결함 허용 기능을 지원하지 않으며 단순히 멀티 쓰레드 방식으로 동작하는 단일 프로세스 언어이다.

본 논문에서 제안된 결함 허용 서비스는 RMI 메커니즘에 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.[12]

HORB[13]는 경량화된 RMI 메커니즘에 기반한 분산 미들웨어이며, 사용이 간편하고 JavaSoft의 RMI 시스템보다 두 세배 빠른 성능을 자랑한다[14]. 이런 관점에서, 본 논문에서는 신뢰성 있는 분산 컴퓨팅을 지원하기 위한 방안으로서, 체크포인트와 롤백 복구(rollback recovery) 메커니즘을 이용하여 기존의 HORB를 확장한 에버그린을 제안하였다. 또한, 에버그린의 주요 컴포넌트를 구현한 뒤 일련의 실험들을 통해서 그 성능과 비례확장성(scalability)을 평가하였다.

II. 제안된 에버그린

1. 시스템 구조

현재 결함 허용 RMI와 관련된 표준화 노력들은 활발하게 진행되지 않고 있는 실정이다. 따라서, 본 논문에서 제안된 에버그린은 결함 허용 CORBA[15]에 관련된 RFP(Request for Proposals)를 참조하여 설계되었

다. 결함 허용 CORBA의 RFP에 따르면, 결함 허용 모드는 Passive와 Active 두 가지 타입으로 크게 분류된다. Passive 타입은 서버 객체에 대한 두 개 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 standby 나 warm standby 모드로 대기하게 된다. Active 타입은 서버 객체에 대한 두 개 이상의 복사본으로 구성되며, 각각은 신뢰성 있는 그룹 통신[16]의 지원 하에 클라이언트의 요청들을 처리한다.

그러나, 응용프로그램 영역에 따라 요구되는 결함 복구 서비스의 QoS(Quality of Service)도 달라질 수 있으며, QoS는 도메인 객체의 QoS 제약을 만족하도록 변경 가능해야 한다. 현재 구현된 에버그린은 Passive 타입의 결함 허용 모드만을 지원하고 있으며, Passive 타입은 cold standby와 warm standby 모드로 세분된다.

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 RM(Replica Manager)에 저장된다. Cold standby에서는 주 객체의 결함이 발생했을 경우 RM에 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.

에버그린은 운영 환경 상에서의 몇 가지 가정 하에서 설계되었다. 첫째, 객체의 결함은 fail-stop 모델[5]을 따른다고 가정한다. 이는 객체는 결함 발생 즉시 바로 종료한다는 것을 의미한다. 일반적으로 많은 시스템들에서 분산 환경 하의 사이트 자체의 결함은 비교적 드물게 발생하며 서로 독립적이다. Standby 객체들은 결함이 발생한 후에 주 서버 객체와의 일관성 유지를 위해 체크포인트 복구 메커니즘을 사용한다. 이 사실은 standby 객체에 행해지는 모든 메쏘드 호출은 piece-wise deterministic[7]이라는 가정을 내포하고 있으며, 이는 결함 허용 시스템 분야에서 일반적인 가정이다. 또한, 클라이언트의 호출은 내재되지 않는다고 가정한다. 이 가정은 호출된 서버 객체는 다른 서버 객체를 호출하지 않으며, 클라이언트의 결함으로 인해 발생되는 서버 객체의 결함은 존재하지 않는다는 사실을 의미하므로 클라이언트 자신의 결함 감지는 불필요하다.

그림 1에서와 같이 에버그린의 결함 허용 구조는 HORB에서 제공되는 HORB ORB의 소프트웨어 계층 위에서 설계되었다. 결함 허용 스텁(stub) 생성기는 서버 객체를 정의하는 소스 프로그램을 입력으로 받아 결함 허용 프락시(proxy)와 스켈톤(skeleton) 스텁 루틴을 자동 생성한다. 프로그램의 실행 도중, 클라이언트 객체와 주 서버 객체 사이의 바인딩이 이루어지면, 클라이언트 측의 결함 허용 프락시는 주 서버 객체에 대한 모니터링 작업을 시작하며, 주 서버 객체에 대한 결함 탐지 및 복구 작업을 주관하게 된다.

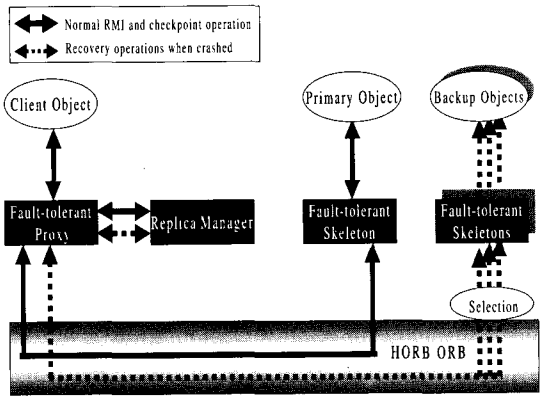

그림 1. 에버그린 시스템의 구조
Fig. 1. Evergreen Architecture.

결함 탐지 메커니즘은 HORB ORB의 기본적인 서비스에 의존한다. 클라이언트와의 바인딩이 유지되는 한, 서버 객체의 결함은 HORB ORB에 의해 탐지 가능하다. 클라이언트는 서버 객체에 대해 원격 메쏘드 호출을 실행한 후, HORB ORB로부터 예외신호(exceptional signal)를 받게 되면 서버 객체의 결함으로 간주한다. 이 예외신호는 클라이언트로 직접 전달되기 전 결함 허용 프락시에 의해 처리 가능하며, 결함 허용 프락시는 HORB ORB를 경유하여 대기중인 서버 객체 중의 하나를 활성화한 뒤, 클라이언트와의 바인딩 및 결함 허용 모드에 따른 적절한 복구 작업을 수행한다.

그림 2는 RM 객체의 Interface 정보를 나타내고 있다. RM은 클라이언트와 주 서버 객체 사이에 전달되는 모든 메시지들을 보관/관리하는 역할을 담당한다. Standby 객체의 위치 및 개수를 포함하는 환경 설정

정보는 미리 제공된다고 가정한다. 결함 허용 Proxy의 초기화 루틴에서 RM에 대한 인스턴스를 생성하게 되고, 그 과정에서 Initialization()이 호출되며, standby 객체의 위치 및 개수를 RM에 등록한다. 결함 허용 Proxy 루틴은 원격 메쏘드의 수행이 성공되는 시점에서 saveMSG()을 호출하여 RMI 메시지를 저장하며, 적절한 시점에서 checkpoint()를 호출하여 서버객체의 상태를 RM에 저장한다.

Cold standby 모드에서 RM은 마지막 체크포인트 이후의 RMI 메시지 및 체크포인트가 발생한 시점에서의 주 서버 객체 상태를 유지한다. Warm standby 모드에서는 마지막 체크포인트 이후의 RMI 메시지들만 저장한다. 주 서버객체에 대한 결함이 탐지되면, 결함 허용 Proxy 루틴은 getAltanativeObj()를 호출하여 대기중인 서버객체들 중에서 새로운 주 객체를 선정한다. 주 객체의 선정에는 FIFO 알고리즘이 사용되었으며, 주 객체로 선정된 객체는 대기 객체 목록에서 자동적으로 삭제된다.

```
interface ReplicaManager {
 public void Initialization();
 public void checkpoint(String id,
                        Object state);
 public Object recoverState(String id);
 public void saveMSG(String id,
                     invoMSG invo);
 public Vector retrieveMSG(String id);
 public String getAltanativeObj(String id);
 public boolean isCheckpointed(String id);
 public void clearAuditTrail(String id);
 ...
}
```

그림 2. Replica Manager의 인터페이스 정보
Fig. 2. Interface of Replica Manager

2. 동작 메커니즘

결함 허용 시스템의 기본 설계 철학은 결함 탐지 및 복구에 대한 투명성(transparency) 보장이다. 이를 위하여 에버그린 시스템은 Passive 타입 결함 허용 서비스를 채택하였고, 메시지 로깅 및 롤백 복구 메커니즘을 이용하여 서버 객체의 상태저장 및 결함이 발생할 경우의 복구 과정을 수행하게 된다. Passive 타입 결함 허용 서비스의 두 가지 기본 모드는 에버그린 내의 결함 허용 프락시와 스켈톤 스텁에 의해 지원된다.

1) 메시지 로깅 및 체크포인트

제안된 시스템에서는 메시지 로깅 및 체크포인트 메커니즘을 이용하여 클라이언트와 서버 사이에 교환된 모든 메시지 및 실행중인 서버객체의 상태를 저장하며, 저장된 정보들은 결함 발생시 복구 처리에 이용된다. 그림 3, 4, 5는 결함허용 스텁과 RM 상에서 구현된 메시지 로깅 및 체크포인트 메커니즘을 보여주고 있는 상태전이도(state diagram)이다.

그림 3은 클라이언트가 서버 객체의 원격 메쏘드를 호출할 때 일어나는 기본 동작 및 서버 측의 상태변화를 예시하고 있다. 클라이언트 측 결함 허용 Proxy는 초기화 과정에서 RM의 인스턴스를 생성하여 동일한 주소 공간에 유지한 뒤, 서버 측에 연결요청을 하게 된다. 클라이언트 측으로부터 연결(bind) 요청을 접수한 서버 측은 주 서버 객체의 인스턴스 생성등이 포함된 서버 객체 초기화(Server object initialization)를 실행하게 된다. 일단, 첫 번째 원격 메쏘드 호출이 성공적으로 수행되면, 서버 객체의 상태 변경이 이루어진다. 이때, 주 서버 객체의 결함 허용 프락시는 결함 복구 처리를 위해 현재의 RMI 메시지를 RM에 저장하며, 이러한 처리과정들은 서버 객체와 RM 사이의 동기적(synchronous)인 동작하에 이루어진다. 이 후의 원격 메쏘드 호출도 동일한 절차에 의해 진행이 된다.

그림 4, 5는 cold standby와 warm standby 모드에서 체크포인트가 호출되었을 경우의 동작 및 서버 측의 상태변화를 보여주고 있으며, 그림 3을 기반으로 체크

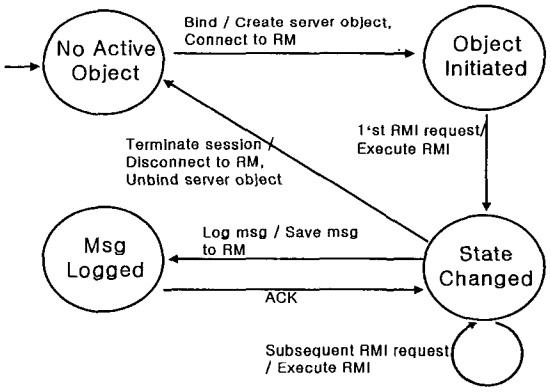

그림 3. 일반적인 원격 메쏘드 호출을 위한 상태전이도
Fig. 3. State diagram of normal operation for remote method invocation.

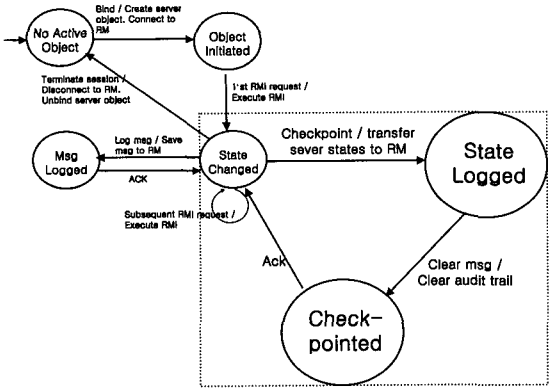

그림 4. Cold standby 모드에서 체크포인트가 호출되었을 경우의 상태전이도

Fig. 4. State diagram for checkpoint issued on client object in cold standby mode.

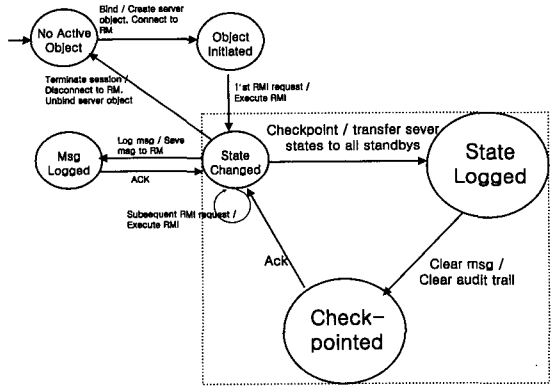

그림 5. Worm standby 모드에서 체크포인트가 호출되었을 경우의 상태전이도

Fig. 5. State diagram for checkpoint issued on client object in worm standby mode.

포인트와 관련되는 동작을 중심으로 나타내었다. 그림 내의 도식들이 보여주듯이, 두 가지 모드에서의 체크포인트 동작들은 서로 상이한 절차를 통해 진행된다. Cold standby 모드에서는, 체크포인트 요청이 발생할 때마다 클라이언트는 주 서버 객체와 통신하여 서버 객체의 내부 상태를 RM으로 전송한다. RM은 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 (asynchronous) 방법으로 전달한 뒤, RM에 저장되어 있는 모든 메시지들을 삭제한다. 두 모드의 차이점을 요약하면 다음과 같다. 첫째, Cold standby 모드에서는 주 서버 객체의 내부 상태가 RM에 의해 유지되지만, warm standby 모드에서는 대기 중인 나머지 서버 객체들로 전송된다. 둘째, 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.

2) 결함 탐지 및 롤백 복구

결함 탐지 메커니즘은 HORB ORB의 기본적인 서비스에 의존하며, 클라이언트는 서버 객체에 대해 원격 메쏘드 호출을 실행한 후, HORB ORB로부터 예외신호(exceptional signal)를 받게 되면 서버 객체의 결함으로 간주한다. 그림 6, 7은 cold standby와 warm standby 모드에서 서버 객체에 대한 결함 탐지 및 롤백 복구 메커니즘을 보여주고 있으며, 그림 3을 기반으로 롤백 복구와 관련되는 동작을 중심으로 나타내었다.

주 서버 객체에 대한 원격 호출 이후 HORB ORB로부터 예외신호가 도착하면 클라이언트는 주 서버 객체에 결함이 발생했음을 감지하게 된다. 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 RM에 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.

Ⅲ. 성능 평가 및 결과 분석

이 장에서는 일련의 실험들을 통해 제안된 에버그린의 성능을 분석하고 최적의 디자인 목표를 지원하기 위한 에버그린의 비례확장성(scalability)에 대해 논의하기로 한다.

자바 언어(JDK 1.2)와 RMI 메커니즘을 기반으로 한

에버그린은 결함 허용 분산 시스템을 위한 프로그래밍 환경으로서 설계되었고, 100Mbps 이더넷 네트워크에 연결된 Sun Ultra10(UltraSparc-IIi, 440 Mhz) 워크스테이션 상에서 구현되었다. 에버그린 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 에버그린 환경하에서 작성한 뒤 그 실행시간을 측정하였다.

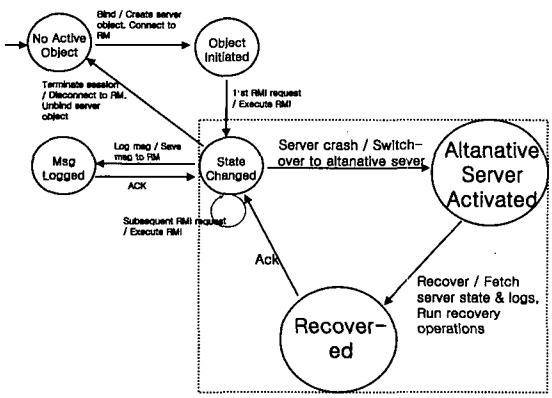

그림 6. Cold standby 모드에서 서버 객체에 결함이 발생했을 경우의 상태전이도

Fig. 6. State diagram for recovery operation when fault is detected in cold standby mode.

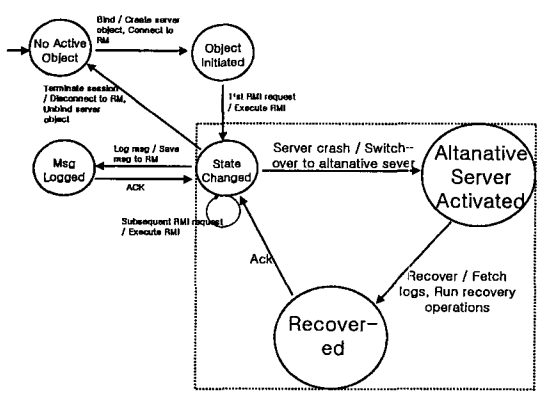

그림 7. Worm standby 모드에서 서버 객체에 결함이 발생했을 경우의 상태전이도

Fig. 7. State diagram for recovery operation when fault is detected in worm standby mode.

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, 통신 복잡도는 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. 통신 복잡도는 n x n 행렬에 대해서 O(n2)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 임의적으로 생성되었으며, 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.

표 1은 cold standby 모드에서 메시지 로깅의 오버헤

드를 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 요구되는 실행시간을, 메시지 로깅을 사용한 경우와 그렇지 않은 경우로 각각 나누어서 제시하였다.

표 1. 에버그린 환경하의 Cold standby mode 에서의 메시지 로깅 오버헤드 (msec)

Table 1. Message logging overhead in cold standby mode with Evergreen (msec).

Program	Size	Message Logging		Overhead	
		with	without	time	percent(%)
n-queens	6	11.65	11.55	0.1	0.9
	8	19.95	19.8	0.15	0.8
	10	201.5	201.4	0.1	0.05
tsp	4	81.7	79	2.7	3.4
	6	117.25	114	3.25	2.9
	8	29498.94	29290.4	208.54	0.7
Gauss	20	365	340.28	24.72	7.3
	40	1393.55	1343.3	50.25	3.7
	60	3275	3210.35	64.65	2.01

표 1에 의하면, n-queens와 tsp 프로그램은 주어진 문제 크기에서 약 3 퍼센트에서 1 퍼센트 미만의 메시지 로깅 오버헤드를 보여주고 있다. 세 가지 실험 모델 중 가장 많은 통신을 수행하는 Guass 프로그램은 약간 높은 약 7 퍼센트에서 2 퍼센트의 메시지 로깅 오버헤드를 나타내고 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 기인하고 있다.

그림 8은 Gauss 프로그램에 결함이 발생했을 경우, 체크포인트 회수에 따른 결함 복구 오버헤드를 측정한 결과이다. X축은 체크포인트 회수를, Y축은 전체실행시간에서 결함복구에 소요되는 시간이 차지하는 비율을 나타내는 결함복구 오버헤드를 의미한다. 실험은 cold standby 모드에서 프로그램의 문제 크기를 20으로 고정하여 실시되었다. 실험 결과는 체크포인트 횟수가 6일 경우에 최적의 성능을 보여주고 있으며, 체크포인트 횟수와 결함복구 오버헤드는 서로 반비례하지 않음을 나타내고 있다. 최적의 체크포인트 횟수는 응용 프로그램과 문제 크기에 의존적이므로 이를 고려한 적절한 선택이 필요하다.

결함 허용 CORBA의 RFP에 따르면, Passive 결함 허용 모드는 cold standby 와 warm standby로 구성되며, 제안된 에버그린에서는 두 모드를 성공적으로 지원하고 있다. 그림 9는 tsp 프로그램을 warm standby 모드에서 standby 개수를 변화시키면서 실험한 결과이다. x축은 standby 개수를, y축은 실행시간을 의미하며, 실험을 위해 문제크기는 6, 체크포인트의 횟수는 2로 고정되었다. 그림 9에 의하면, standby 개수가 증가함에 따라 평균 13.7% 정도의 오버헤드가 발생하였고, 포화(saturation) 현상은 발생하지 않음을 알 수 있다. warm standby 모드가 제공하는 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.

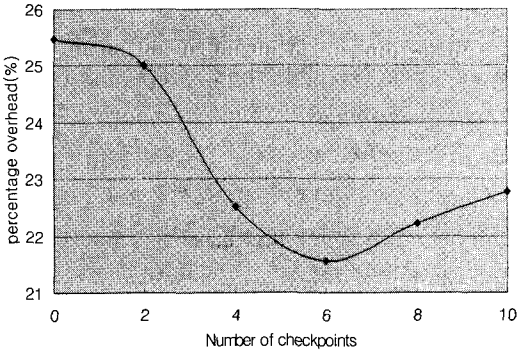

그림 8. 체크포인트 변화에 따른 결함복구 오버헤드
Fig. 8. The recovery overhead under various checkpointing.

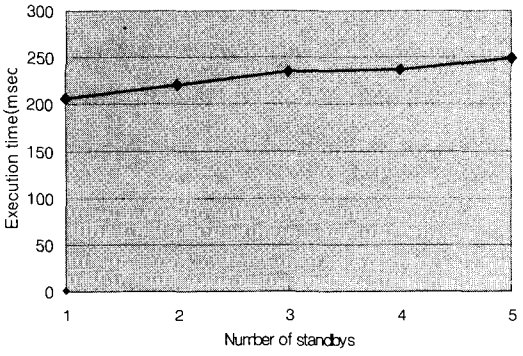

그림 9. Standby 개수 변화에 따른 실행시간
Fig. 9. The execution time under various standbys.

Ⅳ. 결론 및 향후 연구 과제

본 논문에서는 RMI를 기반으로 한 결함 허용 시스템 개발 환경인 에버그린을 제안하였다. 에버그린은 신뢰성 있는 분산 컴퓨팅을 지원하기 위해서 체크포인트와 롤백 복구 메커니즘을 이용하여 기존의 자바 언어를 확장한 형태로 설계되었으며, JDK(version 1.2)와 HORB(version 2.0)을 기반으로 구현되었다. 제안된 에버그린은, 현재, cold standby 와 warm standby의 두 가지 결함 허용 모드를 성공적으로 지원하고 있으며, 에버그린 환경 하에서 구현된 응용 프로그램들이 여러 가지 형태의 하드웨어 및 소프트웨어 결함에도 불구하고 지속적으로 동작함을 확인하였다.

일련의 실험을 통해서 에버그린의 성능을 검증하였고, 최적의 디자인 목표를 지원하기 위한 확장 가능성을 제시하였다. 보다 실제적인 환경에서 메시지 로깅의 오버헤드를 분석하기 위해 n-queens, tsp, Gauss의 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, 응용프로그램이 hot standby 모드에서 적절한 성능을 발휘하기 위해서는, 프로그래머는 분산 시스템에서의 파티션 개수를 고려하여 hot standby의 개수를 신중하게 결정해야 함을 확인하였다.

제안된 에버그린 시스템은 Replica Manager에 결함이 발생하였을 경우에는 결함 허용 기능을 제대로 수행하지 못한다는 결점을 가지는데, 이는 체크포인트와 롤백 복구 메커니즘을 채택한 결함허용 시스템이 반드시 해결해야 할 문제이다. 현재, 구현상에 나타난 여러 문제점 및 실험을 통해서 제시된 주요 쟁점들을 토대로 보다 성능이 향상된 차기 버전의 에버그린 시스템을 구현 중에 있다.

참 고 문 헌

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

[2] Thuuan L. Thai, Thuan L. Thai,Andy Oram, Learing DCOM, O'reilly, April, 1999.

[3] IBM corporation, "SOMobjects : A Practical Introduction to SOM and DSOM", International Technical Support Organization, July, 1994.

[4] Sun Microsystems, JDK 1.2 Documentation, <URL: http://www.javasoft.com/>

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

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

[7] Deron 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.

[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, Sept. 1998.

[9] K.P.Birman and R.VAN Renesse, Reliable Disributed Computing with the Isis Toolkit. IEEE Computer Society Press, 1994.

[10] Robbert van Renesse, Ken Birman, and Silvano Maffeis, "Horus: A Flexible Group Communication System", Communications of the ACM, Vol. 39(4), April, 1996.

[11] M. Rozier ET AL. "Chorus Distributed Operating Systems", Computing Systems Journal, The Usenix Association, Vol. 1(4), 1988.

[12] Sanjav P. A, 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, pp. 565~569, August, 2000.

[13] HIRANO, S., "HORB : Distributed Execution of Java Programs", Worldwide Computing and Its Applications, LNCS 1274, pp. 29~42, 1997.

[14] 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.

[15] Object Management Group, "Fault Tolerant CORBA", OMG TC Document, December, 1999.

[16] S. Maffies and D. C. Schmidt, "Constructing Reliable Distributed Communication Systems with CORBA", IEEE Communications Magazine, Vol. 14, No. 2, February, 1997

저 자 소 개

玄 武 庸(正會員)

1993년 : 경북대학교 컴퓨터 공학과 졸업(공학사). 1995년 : 경북대학교 대학원 컴퓨터 공학과 졸업(공학석사). 2000년~현재 : 충북대학교 대학원 전자계산학과 박사과정. 1995년~현재 : 대원과학대학 컴퓨터 정보통과 조교수. <주관심분야 : 컴퓨터 네트워크, 분산처리, 결함허용 시스템, Bluetooth technology>

金 湜(正會員)

1979년 2월 : 경북대학교 전자공학과(전산전공) 졸업(공학사). 1979년~1988년 : 국방과학연구소 선임연구원. 1990년 : Texas A&M 전산학과 졸업(공학석사). 1995년 : 경북대학교 대학원 컴퓨터공학과 박사수료. 1993년~현재 : 세명대학교 전산정보통신학부 교수. <주관심분야 : 컴퓨터 네트워크, 분산 처리, 결함허용 시스템, Bluetooth technology>

金 明 俊(正會員)

1979년 : 서강대학교 수학과 졸업(이학사). 1984년 : 플로리다 공대 전자계산과 졸업(공학석사). 1992년 : Texas A&M 전산학과(공학박사). 1993년~현재 : 충북대학교 전기 전자 및 컴퓨터 공학부 부교수. <주관심분야 : 실시간 시스템, 운영체제, 분산운영체제>