프라이머리-백업 객체 그룹 지원을 위한 CORBA의 확장

신 범 주* · 김 명 준**

The Extension of CORBA for the Support of Primary-Backup Object Group

Bum-Joo Shin* · Myung-Joon Kim**

Abstract

To provide highly available services in the distributed object system, it is required to support the object group. The state machine approach and primary-backup approach are proposed as two representative approaches for support of object group. The primary-backup approach does not only give merits such as transparency of object group and non-deterministic execution but also require less resource than state machine approach. This paper describes an extension of CORBA that is required to support of the primary-backup object group. In this paper, the state of backup is synchronized with primary through the atomic multicast protocol whenever the request of client is executed at primary. As a result, it does not require message logging and check pointing. The object group of this paper also provides fast response time in case of failure of the primary since it makes primary election unnecessary. And through an extension of IDL, it makes possible to avoid consistency control depending on characteristic of application. A prototype has been implemented and the performance of object group has been compared with a single object invocation.

* 한국전자통신연구원 인터넷서비스 연구부
** 한국전자통신연구원 컴퓨터, 소프트웨어 연구소장

1. 서 론

클라이언트-서버 모델에 기반한 분산 객체 시스템에서는 여러 클라이언트들이 서버 객체를 공유하기 때문에 양질의 서비스를 제공하기 위하여 서버 객체의 가용성을 높이는 것이 바람직하다. 서버 객체의 가용성을 높이기 위하여 사용될 수 있는 가장 보편적이고 경제적인 방법 중의 하나는 서버의 기능을 분산 환경 내의 여러 노드에 중복시키는 것이다.

프라이머리-백업 방식(primary-backup approach)[7]은 중복된 객체들 중 프라이머리로 결정된 특정 하나의 객체만이 외부 서비스를 수행하고 나머지는 프라이머리에 문제가 발생할 경우를 준비하는 방식이다. 프라이머리-백업 방식은 상태 기계 방식(state machine approach)[19]에 비해 객체 그룹의 투명성을 쉽게 제공할 수 있고, 서버 객체의 비결정적 수행(non-deterministic execution)을 지원할 뿐 아니라 상대적으로 쉽게 구현될 수 있는 장점을 가진다.

본 논문에서는 프라이머리-백업 방식의 객체 그룹을 지원하기 위하여 CORBA[17]의 확장 구현한 내용에 대하여 기술한다. 본 논문의 객체 그룹은 클라이언트에 투명하게 제공되며, 기존 CORBA와 상호 운용된다. 프라이머리와 백업의 상태를 일치시키기 위하여 메시지 로깅(message logging) 및 체크 포인팅(checkpointing)을 사용치 않고, 클라이언트 요구 시 마다 프라이머리와 백업의 상태를 일치시키는 방법을 사용함으로써 프라이머리 고장 발생 시에 클라이언트의 대기 시간을 줄인다. 프라이머리와 백업의 상태를 일치시키기 위한 통신은 전용 프로토콜이 아닌 메시지 전송의 원자성을 제공하는 다중전송 프로토콜(atomic multicast protocol)[2]을 사용하기 때문에 상태 기계 방식도 쉽게 지원할 수 있게 한다. 또 객체 그룹 참조자(object group reference)에 프라이머리 후보 순서를 미리 할당하고, 이를 클라이언트의 객체 중개자와 공유함으로써 프라이머리 고장 시에 새로운 프라이머리 선정 과정이 필요치 않게 한다.

본 논문의 다음 장은 프라이머리-백업, 그리고 CORBA에서 객체 그룹과 관련된 기존 연구에 대해 분석하며, 3장에서는 본 논문에서 제시하는 객체 그룹의 동작 구조를 기술한다. 제 4장에서 객체 그룹 관리 기법을, 5장에서 IDL(Interface Definition Lanaguage)의 확장에 대해 기술한다. 6장에서는 프라이머리 객체의 고장 발생 시 클라이언트 요구를 지속시키는 복구 과정을 다루며, 7장에서는 구현 및 성능에 관해 분석한 후 8장에서 결론을 맺는다.

2. 기존 연구

프라이머리-백업 방식이 처음 소개된 이후 다양한 형태의 시스템들이 발표되었으며, 이 방식의 이론적 한계점에 대한 연구 결과도 발표되고 있다. 또 그룹 통신 기능과의 연관성 및 능동 복제 방식과의 비교에 대한 연구 결과들도 발표되고 있다. 그러나 이러한 연구 결과들은 CORBA와 같은 표준 시스템과는 관련성이 없는 독자적인 모델에서의 연구 결과들이며, 대다수 효율적인 프라이머리-백업 프로토콜에 집중되어 있다.

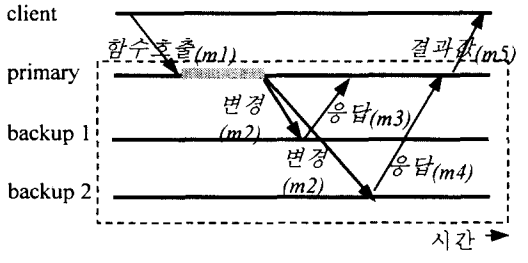
CORBA 환경에서 객체 그룹을 지원하기 위한 연구 결과들로는 OGS[10], Eternal[9], Electra[13] 그리고 New Castle 대학에서 개발한 시스템[15] 등이 있다. 이러한 연구 결과들 대다수가 상태 기계 방식을 사용하여 객체 그룹을 지원하고 있으며, 일부는 추가적 기능으로 프라이머리-백업 방식을 제공하고 있다. 따라서 이들 연구 결과들은 CORBA에서의 프라이머리-백업 방식의 구현 및 성능에 대한 구체적인 언급이

부족한 상태이며, 객체 그룹을 어떠한 형태로 CORBA의 구조에 접목하느냐에 집중되어 있다.

OMG에서는 CORBA에서 객체 그룹을 위한 표준을 제정하는 작업을 진행하고 있다. 현재 5개의 제안서[8, 9, 12, 16, 18]가 제출되었으며, 단일 안을 만들기 위한 작업이 진행되고 있다. 이들 제안서는 표준 규격의 특성 상 프로그래밍 인터페이스 및 구조적인 관점에 집중하고 있으며, 구현과 관련한 구체적인 부분은 언급하지 않고 있다.

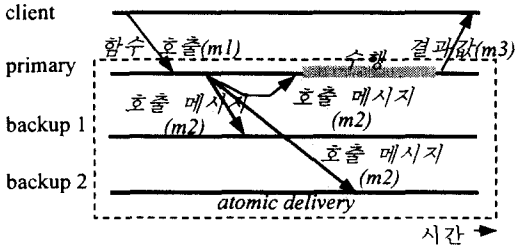
3. 동작 모델

프라이머리-백업 방식에서는 백업의 상태 변경이 프라이머리에 의해 관리된다. (그림 1)은 프라이머리-백업 방식의 동작 과정을 보여준다. 프라이머리는 클라이언트가 함수 호출 메시지를 수행한 후 백업들에게 상태 변경 메시지를 보낸다. 백업 객체들은 상태를 변경한 후 프라이머리에 확인 메시지를 보낸다. 모든 백업들로부터 확인 메시지를 받은 프라이머리는 결과 값을 클라이언트에게 돌려준다[7, 11]. 그러나 이러한 동작 과정은 메시지 전달의 원자성을 제공하는 다중 전송 프로토콜이 지원될 경우 매우 단순화될 수 있다.



(그림 1) 일반적인 프라이머리-백업 방식의 동작 과정

본 논문에서는 프라이머리가 백업 멤버들에게 상태를 전달하기 위하여 메시지 전달의 원자성을 제공하는 다중 전송 프로토콜을 사용한다. (그림 2)는 본 논문의 프라이머리-백업 동작 과정을 보여준다. 프라이머리는 클라이언트의 함수 호출 메시지를 수행하지 않고 백업 멤버들에게 전송한다. 이 때 메시지는 전송한 노드에도 전달된다. 프라이머리 및 백업들은 각각 전달된 메시지를 수행하며, 프라이머리는 그 결과 값을 클라이언트에 전송한다. 이 같이 메시지 전송의 원자성이 보장되는 경우에는 백업들이 확인 메시지를 전송할 필요가 없다. 이상의 과정에서 볼 수 있듯이 클라이언트의 요청 마다 프라이머리와 백업들이 각각 수행하기 때문에 본 논문의 방식에서는 메시지 로깅 및 체크포인팅이 필요치 않다.



(그림 2) 원자성 지원 다중 전송에 기반한 동작 과정

클라이언트-서버 모델에서는 클라이언트가 서버와 연결하기 위한 방법이 제공되어야 한다. CORBA에서는 클라이언트가 서버 객체와 접속하기 위하여 IOR(Interoperable Object Reference)이라는 객체 참조 구조와 네이밍 서비스(naming service)를 이용한다. 본 논문의 객체 그룹도 동일한 방법을 사용한다. 객체 그룹의 모든 멤버들이 객체 참조 구조(object reference)에 표현되며, 이를 네이밍 서비스에 등록한다. 클라이언트는 네이밍 서비스로부터 객체 참조 구조를 얻은 후 이를 이용하여 서버 객체에 접속하며, 프라이머리 객체의 고장 발생 시에도 이 객체 참조 구조를 이용하여 다음 프라이머리에 접

속하여 계속 수행하게 된다.

4. 그룹 관리

본 논문의 프라이머리-백업 복제는 동일한 프로그램이 여러 노드에서 수행되는 모델을 사용한다. 각 프로그램들은 수행 시에 그룹 이름을 기반으로 하여 프로세스 그룹을 구성한다. 그룹은 형상 파일(configuration file)에 정의된 고정 멤버(static membership)로 구성되며, 고장 발생 후에 복구된 멤버의 재가입이 허용되지 않는다.

그러나 CORBA의 클라이언트-서버 관계는 객체를 기반으로 동작한다. 클라이언트는 객체 참조 구조를 사용하여 서버 객체에 접속하고 필요한 함수를 호출한다[17]. 객체는 동적으로 생성되기 때문에 각 프로세스에서 서로 다른 참조 구조를 가진다. 각 프로세스에서의 복제 객체를 하나의 참조 구조로 표현할 수 있기 위하여 프로세스와는 별도로 객체 그룹을 관리하는 것이 필요하다. 본 논문은 프로세스 및 객체 그룹의 두 종류 그룹을 관리한다. 프로세스 그룹은 다중 전송 채널에 의해 관리되며, 객체 그룹은 객체 중개자가 관리한다.

4.1 멤버 관리

본 논문에서 그룹(프로세스 및 객체 그룹)은 객체 중개자(ORB : Object Request Broker)에 등록할 때 제공되는 스트링(string) 타입의 이름으로 구분하며, 이를 위하여 객체 중개자의 프로그래밍 인터페이스를 확장한다. CORBA에서는 서버 프로그램이 클라이언트의 함수 호출을 서비스할 준비가 되었을 때 ***impl_is_ready*** 인터페이스를 통해 객체 중개자에 알리며, 새로운 객체를 생성하였을 때 ***obj_is_ready***인터페이스를 통해 객체 중개자에 등록한다. 본 논문에서는 ***impl_is_ready***를 확장하고 ***obj_group_is_ready*** 인터페이스를 제공함으로써 프로세스 및 객체 그룹을 관리한다. (그림 3)은 확장된 인터페이스를 사용한 서버 프로그램의 예를 나타낸다. 프로그램은 *PrimaryBackupChannel* 이라는 프로세스 그룹으로 동작하며, *Benchmark*란 객체 그룹을 지원한다. ***impl_is_ready***가 호출될 때 객체 중개자는 다중 전송 채널에 연결하며, 최초로 호출한 프로세스가 프라이머리가 되고 나머지 멤버들은 백업으로 동작한다. 프라이머리 프로세스의 모든 객체들은 프라이머리 객체가 된다.

```
package benchmark;
public class Server{
  public static void main( String[] args ){
    ORB orb = ORB.init();
    BOA boa = orb.BOA_init();
    Object b = boa.create( new benchImpl(),
                  "IDL:benchmark/bench:1.0");
    . . .
    boa.obj_group_is_ready(b,"Benchmark");
    . . .
    boa.impl_is_ready("PrimaryBackup");
  }
}
```

(그림 3) 서버 프로그램 예

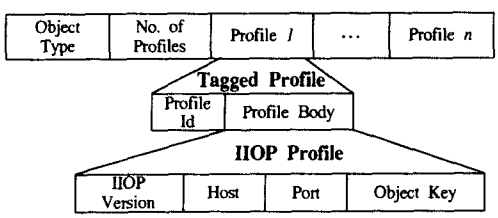
서버 프로세스에서 객체를 생성하고 이를 객체 중개자에 등록하면, 객체 중개자는 클라이언트가 이 객체에 연결하기 위하여 사용할 객체 참조자를 생성한다. 객체 참조자는 객체 중개자에 의해 동적으로 생성되며, 객체 중개자에 의해 객체와 연결되기 때문에 동일한 프로그램이 수행될 경우에도 다를 수 있다. 본 논문에서는 객체가 생성되어 객체 중개자에 등록될 때 객체 그룹 참조자를 생성하기 위하여 객체에 관한 정보(노드, 포트, 객체 키)를 모든 프로세스 그룹

에 전송한다. 정상적으로 동작되는 모든 멤버로부터 새로운 객체에 관한 정보가 도착하면 프라이머리는 객체 그룹 참조자를 생성한다. 따라서 모든 멤버들이 객체 그룹의 멤버에 대한 정보를 가지게 된다.

4.2 그룹 참조

CORBA에서는 클라이언트가 서버에 접속하기 위해서 사용되는 IOR(Interoperable Object Reference)라는 객체 참조 구조를 정의하며, IIOP는 IOR에 포함되는 정보에 대하여 구체적으로 정의한다. 즉 IIOP에서는 IOR이 노드, 포트 그리고 서버 객체를 구분 할 수 있는 키에 대한 정보를 제공하도록 정의하고 있다[17]. 클라이언트는 IIOP를 이용하여 서버와 통신하기 때문에 IIOP에서 정의한 객체 참조 구조를 사용하여 서버 객체 그룹의 멤버들 중 하나에 접속하여야 한다.

CORBA는 이 기종간의 상호 운용성을 지원하기 위한 표준이기 때문에 객체 그룹을 지원하는 경우에도 상호 운용성을 제공하는 것이 바람직하다. 객체 참조 구조를 변경하는 것은 상호 운용을 불가능하게 할 수 있기 때문에 본 논문에서는 IIOP에서 사용하는 객체 참조 구조를 변경하지 않고, 단지 의미를 확대하여 사용한다. (그림 4)는 본 논문에서 사용하는 객체 그룹 참조 구조를 나타내고 있다.



(그림 4) 객체 그룹 참조 구조

객체 그룹을 표현하는 객체 참조 구조는 그림과 같이 여러 개의 프로화일(profile)을 가지며, 각각의 프로화일은 멤버에 관한 정보를 표현한다. 프로화일에 나타나는 순서가 프라이머리 후보 순위이다. 즉 제일 처음 나타나는 프로화일은 프라이머리를 나타내며, 다음은 프라이머리 후보이다. 이 같은 구조는 객체 그룹을 지원하지 않는 경우와 동일하기 때문에 객체 타입을 표현하는 필드에 특정 표시어를 추가하여 객체 그룹을 나타내는 IOR임을 구분할 수 있게 한다. IIOP에서는 객체 참조 구조에 적어도 하나 이상의 프로화일을 가져야 한다고 정의되어 있기 때문에 이 구조는 CORBA IIOP에서 정의된 객체 참조 구조와 동일한 것으로 인정될 수 있다[17].

5. IDL 확장

백업 객체들의 상태를 프라이머리와 일치되게 유지하기 위하여 요구되는 오퍼레이션들은 중복을 허용하지 않는 시스템에 비해 상대적인 성능 저하를 일으키게 하는 요인이 된다. 따라서 프라이머리의 상태에 영향을 미치지 않는 오퍼레이션의 경우에는 상태 일치 과정을 생략함으로써 성능을 향상시킬 수 있다. 그러나 이와 같은 특성은 응용 시멘틱에 종속적이기 때문에 프로그래밍 시점에 반영될 수 있도록 지원하여야 한다[1]. 본 논문에서는 프로그래밍 시에 응용의 특성을 반영할 수 있도록 IDL을 확장한다.

```
(88)  <op_attribute>  ::=  "oneway"
                       |   "readonly"
```

(그림 5) IDL 문법의 확장

(그림 5)의 IDL Grammar는 OMG의 문서 CORBA/IIOP 2.3로부터 관련된 문법만을 발췌한 것이다. 추가된 문법은 굵은 글자로 표시된 readonly이다. 확장된 문법은 객체의 멤버 함수

가 readonly로 선언될 수 있음을 의미하며, read-only로 선언된 멤버 함수는 해당 객체의 내부 상태를 변경하지 않기 때문에 백업들의 상태를 변경할 필요가 없다. 따라서 상태 일치에 소모되는 시간을 단축함으로써 수행 성능을 향상시킬 수 있다.

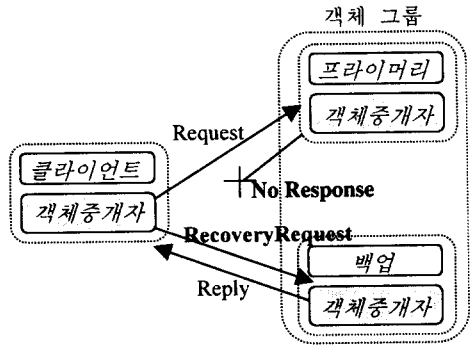
```
interface account{
    long   debit(in long value);
    long   credit(in long value);
    readonly long   amount();
};
```

(그림 6) 확장된 문법의 인터페이스 예

Readonly 함수의 구현은 스켈레톤 클래스에서 해결하는 방법과 ORB 수준에서 해결하는 방법이 가능하다. 전자의 방법은 후자의 방법에 비해 메시지의 내용 분석에 추가적인 시간이 소모되기 때문에 본 논문에서는 후자의 방법을 통해 구현하였다. readonly 멤버 함수의 경우에는 클라이언트 스텁이 오퍼레이션 호출 메시지의 멤버 함수 이름에 ***object_group_readonly*** 표시어를 추가하여 전송하고, 서버 ORB에서 이를 인지하여 해당 멤버 함수를 다중 전송 하지않고 호출한다. 이 같은 방법은 전자에 비해 성능이 향상될 수 있는 반면 IDL 컴파일러의 확장이 필요하다. (그림 6)은 추가된 IDL 문법을 이용하여 기술된 인터페이스 정의 예를 보이고 있다.

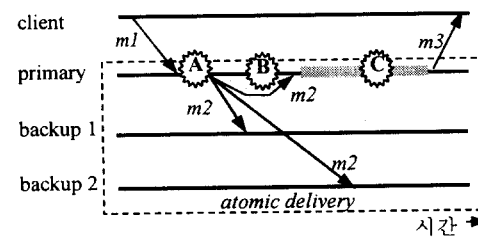
6. 고장 복구

객체의 중복을 지원하는 이유 중 하나는 프라이머리 객체에 고장이 발생한 경우에도 객체의 가용성을 제공하는 것이다. 클라이언트에게 프라이머리의 고장 투명성을 제공하기 위하여 두 가지의 과정이 요구된다. 새로운 프라이머리의 선출 그리고 고장 발생시에 전달된 클라이언트의 요구를 새로운 프라이머리가 계속 지원할 수 있게 하는 것이다.



(그림 7) 프라이머리 고장 발생 시 복구 과정

본 논문의 복구 과정은 (그림 7)과 같다. 클라이언트의 객체중개자는 객체 참조 구조에서 우선 순위가 가장 높은 백업 객체에 고장이 발생한 프라이머리에 전송하였던 메시지를 새로운 메시지 타입(RecoveryRequest)으로 재전송한다. 클라이언트의 복구 메시지를 받은 새 프라이머리는 해당 메시지가 복구 큐에 존재할 경우 결과 값을 전송하고, 그렇지 않을 경우 수행한다. 그러나 이러한 과정은 상태 불일치를 초래할 수 있기 때문에 프라이머리에 오류 발생 시점에 따라 점검되어야 한다. (그림 8)은 고려되어야 할 프라이머리의 고장 발생 시점을 나타낸다. A는 클라이언트로부터 메시지를 받고 이를 백업들에게 전송하기 전이며, B는 전송 도중에, 그리고 C는 메시지에 따른 상태 변경 중에 발생한 경우이다.



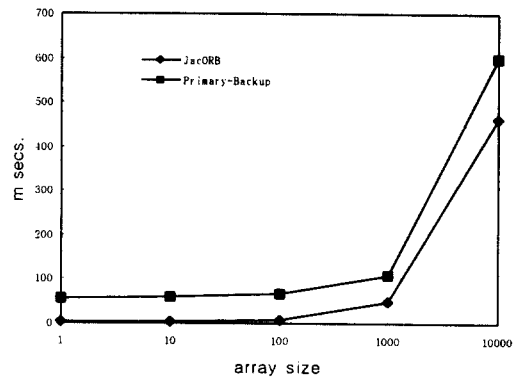
(그림 8) 고려되어야 할 고장 발생 시점

A의 경우는 새로운 프라이머리가 클라이언트의 복구 메시지를 다시 수행하면 해결되며, C의 경우에는 새로운 프라이머리가 이미 수행했기 때문에 결과 값을 클라이언트에 전송하면 해결된다. 그러나 B의 경우 일부 백업은 메시지를 받고 수행했을 수도 있고 일부는 메시지를 받지 못했을 경우도 있다. 이러한 경우가 방지될 수 있도록 다중 전송 채널이 원자성을 지원하는 것이 요구된다. 즉 백업들이 아무도 메시지를 받지 않은 경우 A와 같은 경우로 해결될 수 있으며, 모두 받은 경우에는 C의 경우로 해결될 수 있다.

7. 구현 및 성능 분석

본 논문의 시제품은 공개 소프트웨어인 JacORB[5]와 MTP-2[4]를 이용하여 구현하였다. JacORB는 CORBA 2.0을 지원하는 Java 언어로 구현된 CORBA이다. MTP-2는 MTP 규격[2]을 구현한 트랜스포트 계층의 다중 전송 프로토콜로서 메시지 전송의 원자성을 지원하며, socket과 유사한 인터페이스를 제공하는 특성을 가진다. JacoORB는 Java로 구현된 반면 MTP-2는 C 언어로 구현되어 있기 때문에 이를 JacORB에 연결하기 위하여 Java 인터페이스를 지원하도록 확장하였다.

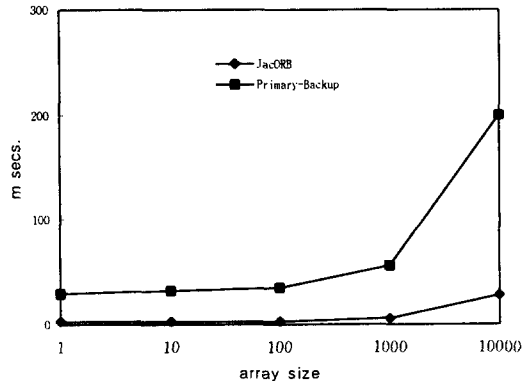
성능 시험을 위한 응용으로는 JacORB의 성능 시험에 사용되었던 benchmark 프로그램[5, 6]의 서버 프로그램을 전체 동작 과정의 의미를 변경하지 않고 단지 객체 중복을 지원할 수 있게 수정하였다. 벤치마크 프로그램의 서버 객체는 세 개의 함수를 제공한다. 하나는 매개변수 및 결과 값이 없는 단순 함수이다. 이 단순 함수는 함수 호출을 위한 메시지 전달 과정의 성능에 대한 시험을 위하여 사용된다. 두 번째 함수는 정수 배열을 매개 변수로 하며, 결과 값으로 매개 변수로 주어진 배열을 돌려주는 기능을 수행한다. 세 번째는 두 번째 함수와 유사하나 매개 변수 및 결과 값으로 정수 변수 한 개를 멤버로 갖는 Struct 배열을 사용하는 함수이다. CORBA IIOP에서는 매개 변수의 마샬링(marshalling) 방법이 기본 타입과 struct 타입을 구분하여 동작하기 때문에 이 같은 함수는 성능 시험으로 가치가 있다.



(그림 9) Struct 배열을 사용한 성능비교

성능 시험은 Solaris 2.5.1.이 수행되는 3대의 SUN Ultra-1 과 Solaris 2.7이 수행되는 SUN Enterprise-450을 100Mbps 스위칭 허브로 연결된 환경에서 JDK1.1.7을 사용하여 시행하였다. JacORB 성능 측정은 SUN Ultra-1에 서버를, 그리고 Sun Enterprise-450에 클라이언트를 위치시킨 환경에서 시행하였다. 프라이머리-백업은 SUN Ultra-1에 객체 그룹 멤버를 각 한 개씩 둠으로써 전체 3개의 중복 객체가 수행될 수 있도록 하고 클라이언트를 SUN Enterprise에 위치시켜 측정하였다. 실제 측정에서는 멤버의 수에 따른 결과 값을 구하였으나, 멤버 수의 변화에 따라 성능의 변화가 거의 없었기 때문에 그림에 나타낼 수가 없었다. 시험은 200 번을 수행시킨 결과 값을 평균하였다. 시험 결과를 (그림 9), (그림 10)에 나타내었다. (그림 9)는

struct 배열을 매개 변수로 사용하는 함수를 배열의 크기에 따라 측정한 결과를, (그림 10)은 정수 배열을 사용하는 함수의 측정 결과를 나타내었다. 첫 번째 함수를 사용한 측정 결과는 두 번째 함수를 사용하는 측정에서 배열의 크기를 1로 주었을 때와 동일하게 나타나기 때문에 본 장에서 결과를 별도로 나타내지 않았다.



(그림 10) Integer 배열을 사용한 성능 비교

(그림 9)와 (그림 10)에 나타나는 성능의 차이는 메시지 변환 과정에 소요되는 시간이다. IIOP에서는 기본 타입의 경우에는 변환 과정이 필요치 않는 반면 확장 타입의 경우에는 객체로 취급되므로 변환 함수가 호출되어야 한다. 즉 (그림 10)은 메시지 전송에 소요되는 성능을 보여준 반면 (그림 9)는 데이터 변환에 소요되는 시간이 추가된 것이다.

상기 그림에 나타난 성능 측정 결과는 성능을 향상시키기 위하여 보다 많은 연구가 필요함을 보여 주고 있다. 본 논문에서 제안한 readonly 함수는 한 가지 방법일 수 있다. 많은 응용에서 자료를 갱신하는 오퍼레이션보다 자료를 읽는 경우가 많기 때문에 응용의 특성을 고려하여 readonly 함수로 선언할 경우 성능은 보다 개선될 수 있다. 당연히 예측되는 결과이지만 read-only 함수의 성능 측정은 확장 전의 JacORB의 성능과 동일함을 볼 수 있었다.

8. 결 론

본 논문은 CORBA에 프라이머리-백업 객체 그룹 지원 기능을 추가한 내용을 기술하였다. 확장된 CORBA는 클라이언트에 그룹 투명성을 제공할 뿐 아니라 프로그래밍 인터페이스의 확장이 거의 없기 때문에 기존 응용을 객체 그룹 지원 응용으로 쉽게 변환할 수 있게 한다. 또 IDL 확장을 통해 응용의 특성을 반영할 수 있게 함으로써 프라이머리-백업 사이의 메시지 교환을 최소화될 수 있게 하여 수행 성능을 향상시킬 수 있는 방법을 제안하였다.

본 논문의 객체 그룹은 클라이언트의 요구를 프라이머리와 백업들이 동시에 수행케 함으로써 프라이머리와 백업들이 항상 일치된 상태를 유지케 한다. 이 같은 동작 모델은 메시지 로깅 및 체크포인팅을 필요로 하지 않기 때문에 구현의 단순성을 제공하고 최악 서비스 반응 시간을 줄일 수 있게 한다. 또 클라이언트가 객체 그룹 멤버들의 정보가 표현된 객체 참조 구조를 가짐으로써 프라이머리 고장 시에 별도의 프라이머리 선출 과정과 이의 전달 과정이 필요치 않게 하였다.

그러나 구현된 시제품의 성능 시험 결과는 기존의 CORBA와 비교할 때 객체 그룹을 지원하기 위하여 감수하여야 하는 성능 저하가 예상보다 크다는 점을 보여주고 있다. 이 같은 성능 저하를 줄이기 위한 방법 중의 하나는 최적화된 프라이머리-백업 프로토콜을 제공하는 것이다. 최적화 프로토콜의 연구는 향후 계획 중 하나이다. 또 성능 향상을 위하여 메시지 로깅과 체크 포인팅을 적용하여 응용에 따른 최적 성능을 제공할 수 있는 시스템에 관한 연구가 진행 중이다.

참 고 문 헌

[1] 신범주, 이동현, "대등관계 복제객체 모델을 지원하는 분산 객체 프로그래밍 언어의 설계 및 구현", 정보과학회 논문지(C) 5권4호, 1999.

[2] Armstrong S. et. al., "Multicast Transport Protocol," DARPA RFC 1301, 1992.

[3] Becker, T., "An Approach to Transparent Fault Tolerance for Client-Server Models," Technical Report, U. Kaisersalutern, 1992.

[4] Bormann, C., Ou, J., Gehrcke. H-C., Kerschat, T. and Seifert, N., "MTP-2 : Towards Achieving the S.E.R.O. Properties for Multicast Transport," Technical Report of TU-Berlin 1994.

[5] Brose, G., "JacORB : Implementation and Design of a Java ORB," Procs. of DAIS'97, 1997.

[6] Brose, G., "JacORB Performance compared," http://www.inf.fu-berlin.de/~brose/jacorb/performance/results_09html, 1998.

[7] Budhiraja, N., et al, "The Primary-Backup Approach," in Distributed Systems, ACM Press, 1993.

[8] Ericsson, Iona Technologies and Nortel Networks, "Fault Tolerant CORBA," OMG Document orbos/98-10-10, 1998.

[9] Eternal Systems and Sun Micro Systems, "Fault Tolerance for CORBA Version 1.0-Initial Submission," OMG Document orbos/98-10-08, 1998.

[10] Felber, P., Garbinato, B., and Guerraoui, R., "A CORBA Object Group Service," Technical Report 97-223, Ecole Polytechque Fdrale de Lausanne, 1997.

[11] Guerraoui, R. and Schiper, A., "Software-Based Replication for Fault Tolerance," IEEE Computer, 1997.

[12] Highlander Communications et al., "Fault Tolerant CORBA using Entity Redundancy," OMG TC Document orobos/98-10-09, 1998.

[13] Maffeis, S., "Adding Group Communication and Fault-Tolerance to CORBA," USENIX, 1995.

[14] Miller, C. K., "Multicast Networking and Applications," Addison Wesley, 1999.

[15] Morgan, G., Shrivastava, S.K., Ezhilchelvan, P.D., Little, M.C., "Design and Implementation of a CORBA Fault-Tolerant Object Group Service," Technical Report of New Castle Unv., 1998.

[16] Objective Interface Systems, "Fault Tolerant CORBA Through Entity Redundancy," OMG TC Document orbos/98-10-03, 1998.

[17] OMG, "The Common Object Request Broker : Architecture and Specification," Revision 2.3, OMG, 1998.

[18] Oracle Corporation, "Fault Tolerance RFP," OMG TC Document orbos/98-10-13, 1998.

[19] Schneider, F. B., "Replication Management Using The State Machine Approach," in Distributed Systems, ACM Press, 1993.

[20] Landis, S. and Maffeis, S., Building Reliable Distributed Systems with CORBA, Theory and Practice of Object Systems, Vol.3, No. 1, John Wiley, April 1997.

[21] Wang, L. and Zhou, W., "Primary-Backup Object Replication in Java," IEEE TOOLS98, 1998.

■ 저자소개

신 범 주

경북대학교 전자공학과를 졸업하고, 동 대학원에서 공학석사, 공학박사 학위를 취득하였다. 현재 한국전자통신연구원(ETRI) 인터넷분산처리연구팀에서 Fault Tolerant CORBA를 개발하고 있으며, 주요관심분야는 분산객체시스템, 고장감내 소프트웨어, 인터넷 분산 컴퓨팅 및 모바일 컴퓨팅 등이다.

김 명 준

서울대학교 계산통계학과를 졸업하고, KAIST에서 이학석사, 그리고 프랑스 Nancy 제1대학교에서 박사학위를 취득하였으며, 한국전자통신연구원(ETRI)에서 시스템S/W 연구개발을 담당하였다. 현재 ETRI 컴퓨터·소프트웨어연구소장으로 재직하고 있으며 주요관심분야는 데이터베이스, 분산시스템, 실시간DB 및 소프트웨어공학 등이다.