동적 코디네이터 기반의 능동형 복제를 통한 결함허용 시스템 설계

류승덕*, 이인환**
한양대학교 전자통신컴퓨터공학과
e-mail: *sdyou@csl.hanyang.ac.kr, **ihlee@hanyang.ac.kr

Design of a Fault-Tolerant System Using Dynamic Coordinator-Based Active Replication

Seungduk You*, Inhwan Lee**
Dept. of Electronics and Computer Engineering
Hanyang University

요 약

본 논문에서는 인터넷 기반의 예약 시스템 설계시 가용성(Availability)을 높이기 위해 능동형 레플리케이션(Active Replication)기법을 이용하였으며 서비스 요청에 대한 응답지연 시간을 줄이고자 레플리케이션 그룹 내에 동적 코디네이터(Dynamic Coordinator)를 이용하였다. 실험 결과는 코디네이터를 사용하였을 경우 33% 빠른 응답속도를 보였다. 그러나 코디네이터 선출 방식을 동적으로 하였을 경우 코디네이터 선출시 사용되는 많은 메시지로 인하여 79.98% 증가된 지연시간을 확인하였다.

1. 서론

오늘날 인터넷은 빠른 보급과 개선으로 많은 사람들의 주요 통신 매체가 되고 있다. 또한 금융, 유통, 교통, 미디어 등의 영역에서는 이미 인터넷을 통하여 사용자들에게 다양한 서비스들을 제공하고 있다.

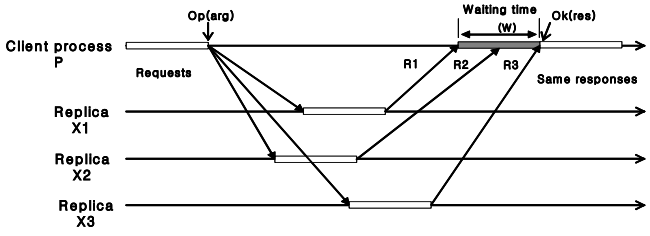
이와 같은 인터넷 서비스들의 시스템 설계시 중요하게 다루어지는 이슈로 고장 허용 기능이 있다. 고장허용 기능을 제공하기 위해 대부분의 시스템들은 서비스 객체를 중복으로 운영하는 소프트웨어 레플리케이션 기법을 사용하고 있다. 그러나 이러한 기법을 이용할 경우 고려해야 하는 사항들로는 첫째, 중복된 서비스 객체들이 가지고 있는 중복된 데이터들의 일관성을 유지 시켜주는 방법이 필요하다. 둘째, 중복된 데이터들의 일관성을 유지하는 과정에서 소요되는 시간들을 최소화 시켜 줌으로써 사용자들의 서비스의 사용속도를 높여 주는 방법이 요구된다. [1]

본 논문은 온라인 예약 서비스 등에서 활용될 수 있는 고가용성 및 개선된 응답속도를 제공해주는 결함허용 시스템을 설계 하고자 한다.

2. 관련 연구

2.1 소프트웨어 레플리케이션

시스템의 가용성을 높이기 위한 레플리케이션 기법으로 수동형 레플리케이션과 능동형 레플리케이션 기법이 있다. 그중에 그림1의 능동형 레플리케이션 방법은 복제된 서비스 객체 그룹 내에서 모든 서비스 객체들이 활성화 상태를 유지한다. 모든 서비스 객체는 사용자로부터 동일 요청을 받아 병행 처리되며 각자의 결과 값을 사용자에게 보내준다. [2]



(그림 1) 능동형 레플리케이션 모델

2.2 레플리케이션 구조

복제된 서비스 객체를 관리하는 방법으로 서버 내에서 자체적으로 관리하는 클라이언트 서버 구조(Two-tier arch.)가 있다. 그리고 서비스 객체를 관리하는 서버 측의 작업량을 줄이기 위해 중간 단에 프록시를 두어 관리하는 클라이언트, 프록시, 서버 구조(Multi-tier arch.)가 있다. [1][3][5]

2.3 그룹 커뮤니케이션

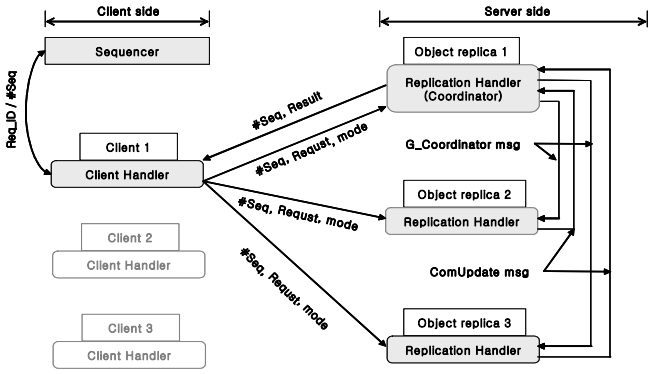
서비스 객체들은 중복된 데이터들 간에 일관성 유지와 그룹 내 객체들 간의 신뢰성 있는 통신을 위하여 다음과 같은 그룹 커뮤니케이션 기능들이 필요하다.[2][4]

먼저 메시지 순서(Ordering messages)를 통해 서비스 객체들은 서비스 요청에 관하여 모두 같은 처리 순서를 가져야 한다. 또한 하나의 메시지는 서비스 객체들에게 모두 전송된다는 원자성(Atomicity of message update) 보장해야 한다. 그리고 네트워크 문제로 손실된 메시지들을 다시 전송 시켜줌으로써 종료성(Termination)을 제공해야 한다. 끝으로 그룹 내에 서비스 객체가 새롭게 추가되거나 현재 존재하는 객체들의 상태 정보들을 알려주는 그룹 멤버십(Group membership) 기능이 필요하다.

3. 시스템 디자인

3.1 시스템 요구사항

시스템을 설계시 다음과 같은 요구사항이 있다. 첫째는 적절한 레플리케이션 기법의 선택이 필요하다. 예약 시스템은 시스템 고장 시에 빠른 복구를 필요로 하기 때문에 시스템 복구지연 시간이 높은 수동형 레플리케이션보다는 능동형 레플리케이션 기법이 적합하다. 둘째는 레플리케이션 구조의 선택이다. 프록시를 이용한 클라이언트, 프록시, 서버 방식은 서비스 객체들이 가지고 있어야 하는 부가적인 기능들을 대신처리해 주는 장점이 있다. 그러나 클라이언트와 서버를 연결해주는 프록시의 고장은 시스템 전체의 고장을 발생 시킬 수 있기 때문에 별도의 복제된 프록시들이 필요하다. 즉 클라이언트, 프록시, 서버 방식보다는 클라이언트 서버 구조가 중복 객체수가 적어진다는 측면에서 유리함을 보여준다. 셋째는 중복 데이터들의 일관성을 유지하기 위하여 메시지 순서를 결정해 주는 별도의 프로세스의 사용과 Totem, Isis, Jgroups, Sensei등과 같은 신뢰성 있는 멀티캐스트 Toolkit을 사용해야한다. 끝으로 고려해야 할 사항은 사용자의 응답시간을 줄여 주는 것이다. 그림 1에서 사



(그림 2) 코디네이터를 이용한 시스템 구성도

용자(P)는 서비스 요청 후 서비스객체로부터 일정 수의 동일한 응답을 기다림으로써 W만큼의 지연시간이 발생하게 된다. 이와 같은 정족수응답을 복제된 서비스 객체 그룹 내에서 보장해 줄 수 있다면 사용자는 W만큼의 지연시간을 줄일 수 있다.

3.2 시스템 구조

본 논문에서 설계하고자 하는 결함 허용 시스템은 그림 2와 같이 3가지 부분으로 구성 되어있다.

① 클라이언트 핸들러(Client Handler)

클라이언트 핸들러는 클라이언트를 구분 짓는 #cl_ID와 클라이언트 내에서 사용한 메시지 순번을 나타내는#cl_seq 값으로 구성된 Req_ID를 이용하여 시퀀서로부터 고유한 메시지 넘버(#seq)를 할당 받는다. 또한 복재된 서비스 객체들에게 사용자 메시지를 전송 및 재전송 해주는 역할을 담당한다.

② 시퀀서(Sequencer)

시퀀서는 내부적으로 {Req_ID, #Seq}로 구성된 Seq_Factory라는 배열을 관리하여 사용자로부터 발행된 메시지들에게 고유의 순서 값을 발행해 주는 일을 담당한다. 또한 서비스 객체의 복구 기능을 위해 잠시 #Seq 발행을 잠시 멈출 수 있다. [6]

③ 서비스객체 핸들러(Replication Handler)

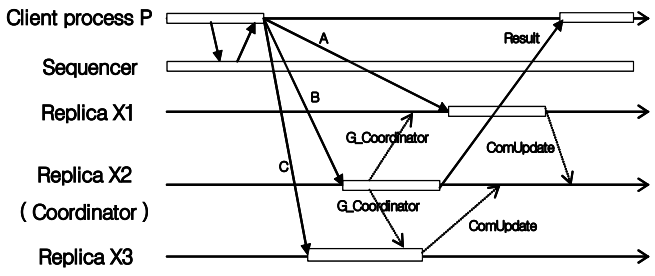
서비스객체 핸들러는 서비스 객체들을 관리하는 코디네이터 기능을 수행한다.

(a) 코디네이터의 역할

코디네이터는 그림 1에서 사용자 측면에서 발생하는 W만큼의 지연시간을 그룹 내의 코디네이터가 담당함으로써 생략 할 수 있다. 또한 다른 중복 서비스 객체(X1, X3)들로부터 사용자의 요청을 정상적으로 처리했음을 의미하는 "ComUpdate" 메시지를 받음으로써 서비스 객체들의 상태를 관리 할 수 있다.

(b) 코디네이터의 선출 프로토콜

서비스 객체들 중 코디네이터를 결정하는 방법은



(그림 3) 동적 코디네이터를 이용한 내부 동작

시스템 초기에 정적으로 특정 서비스 객체에게 부여할 수 있다. 그러나 정적으로 선출할 경우 그림 3에서 사용자가 발행한 서비스 요청 메시지(A, B, C)들이 항상 정적 코디네이터에게 가장 먼저 도착한다는 보장이 없기 때문에 서비스 요청 메시지를 가장 먼저 받은 서비스 객체가 코디네이터 역할을 수행하는 동적 코디네이터 방식이 효율적이다. 그림 4에서 살펴보면 사용자로부터 서비스 요청을 받은 각 서비스객체들은 먼저 'G_CORDI'라는 플래그 값을 확인하여 그룹 내 코디네이터가 존재하는지를 먼저 확인한다. 확인 결과 존재 하지 않을 경우에는 "G_Coordinator"라는 메시지를 다른 서비스 객체들에게 전송하여 자신이 그룹네 코디네이터임을 알린다. 그러나 다수의 서비스 객체들이 근소한 차이로 사용자 의 서비스 요청 메시지를 받게 되면 그룹 내에 여러 개의 코디네이터가 존재할 수 있게 된다. 이때는 초기에 각 서비스 객체들에게 할당된 코디네이터 우선순위 값(#Co_prt)을 비교하여 그룹 내 여러 코디네이터들 중에서 우선순위 값이 제일 높은 서비스 객체가 코디네이터로 결정 된다.

(c) 사용자, 서비스 객체, 코디네이터 간의 연동

사용자, 서비스 객체, 코디네이터는 그룹 내에서 MSG라는 메시지 객체를 이용한다. MSG 객체는 메시지의 타입과 수신자 결정을 나타내는 MSG_TYPE 필드와 서비스 객체들의 상태를 제어하기 위해 MSG_COMMAND 필드를 이용한다.

(d) 코디네이터 고장 감지 및 처리

그림 3에서 코디네이터(Replica x2)는 다른 중복 서비스 객체로부터 "ComUpdate" 그룹 메시지들을 받는다. 이 과정에서 중복 서비스 객체(Replica x1은 정해진 timeout 시간 동안 "ComUpdate"에 대한 ACK를 확인 하지 못할 경우 코디네이터(Replica X2)를 고장으로 판단한다. 가장 먼저 그룹네 코디네이터의 고장을 인식한 서비스 객체는 앞에서 언급한 코디네이터 선출 프로토콜을 이용하여 그룹 내에 코디네이터로 선출되게 된다.

```
23 When (deliver["G_MESSAGE",<#seq, #Co_prt, MSG_Command>] from coordinator) do
24      wait until((#seq <= #ExpectedSeq) and (MSG_Command is "G_Coordinator"))
25          if(G_CORDI is EMPTY) then
26                  G_CORDI := FALSE;
27          else if(G_CORDI is TRUE) then
28                  if(My Coordinator priority > #Co_prt)
29                      Ignore MSG_Command of "G_Coorinator" delivered from other replica;
30                  else
31                      G_CORDI := FALSE;
32          else
33                  G_CORDI := G_CORDI;
```

(그림 4) 코디네이터 선출 슈도코드

4. 시스템 구현

4.1 구현 환경

시스템의 주요 기능은 예약을 신청하는 쓰기 과정과 자신의 예약 정보를 볼 수 있는 읽기 기능으로 추상화 하였다. 시스템은 윈도우XP 환경에서 JDK1.4.2를 이용한 java application으로 구현 되었다. Sequencer, Client Handler, Replica Handler들은 각각 Sequencer.java, Client.java, Replica.java로 구현 되었다.

4.2 그룹 커뮤니케이션 사용

Java 환경에서 사용할 수 있는 JGroups toolkit을 사용하였다. **[4]** 그림 5를 보면 connet() 함수를 통해 그룹에 참여 할 수 있다. 문자열 props을 이용하여 그룹 커뮤니케이션을 초기화를 설정한다. 그리고 멀티케스트 기능과 메시지 재전송을 담당하는 send() 함수를 이용하여 메시지를 전달한다. 그룹 내 서비스 객체로부터 새로운 메시지가 호출 시에 receive()함수를 이용하여 원하는 메시지를 처리 할 수 있다.

```
public class client implements MessageListener, MembershipListener{
    ....
    String props="UDP(mcast_addr=228.8.8.8;bind_addr=127.0.0.1):"
                 +"PING:FD:STABLE:NAKACK:UNICAST:" +
                 "FRAG:FLUSH:GMS:VIEW_ENFORCER:STATE_TRANSFER:QUEUE";
   MSG msg_temp;
   Object recv_msg;
public static void main(String[] args) throws Exception{
   Channel channel=new JChannel(props);
   channel.connect("Online Reservation Group");
   ....
   channel.disconnect();
   channel.close();
}
public void write_request()
{  ....
   msg_temp = new MSG();
   msg_temp.Seq = seq;
   msg_temp.request = "WRITE REQUEST!!";
   send_msg=new Message(null, null, msg_temp);
   channel.send(send_msg);
   ....
}
public void receive(Message msg)
{
   recv_msg=msg.getObject();
   if(recv_msg instanceof MSG)
   { System.out.println("Received " + recv_msg); }
   ...
}}
```

(그림 5) JGroups을 이용한 그룹 커뮤니케이션

```
public class MSG implements Serializable{
                    int Seq;
                    int mode;
                    int Co_prt;
                    String MSG_Type;
                    String MSG_Command;
                    String request;
                    String result;
                    IDENTIFIER Req_ID;
                    Vector backup;
}
```

(그림 6) 그룹 메시지 필드

4.3 그룹 메시지 정의

그룹 내에서 메시지들은 Jgrops에서 제공하는 Message object를 통하여 운반된다. Message객체는는 사용자가 정의한 String, Object들을 필드로 담을 수 있다. 본 논문에서는 그림 6과 같이 메시지 객체를 정의 하였다.

5. 실험 및 결과

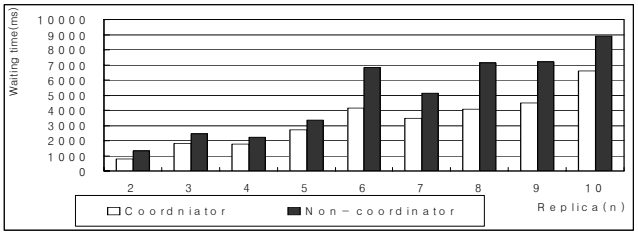
실험은 동일한 LAN에 있는 두 대의 펜티엄Ⅳ 1.7G 512MB의 PC에서 실험하였다. Sequencer.java, Client.java 프로세서는 하나의 PC에서 실행되었으며 나머지 PC에서 Replica.java 프로세서들이 실행되었다. 복재된 서비스 객체 수는 2개부터 10개까지 실행하였다. 먼저 코디네이터를 사용했을 시에 성능 차이를 비교하였다. 다음으로 코디네이터 선출 방식의 차이를 두어 시스템의 지연 시간을 측정하였다.

5.1 코디네이터 유무에 따른 성능 차이

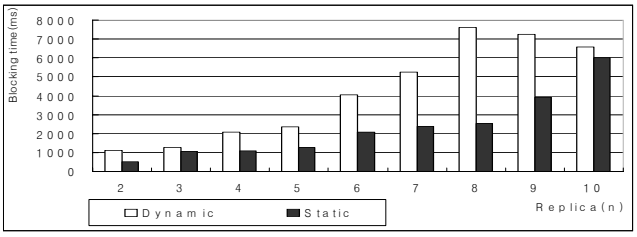
그림 7에서 코디네이터를 사용하였을 경우 평균 33% 응답속도가 단축 되었다. 서비스 객체 수가 5개 이하일 때까지는 24.4%, 서비스 객체가 6개를 넘어갈 경우부터는 코디네이터를 이용할 경우 사용자에게 35.4% 단축된 응답속도를 제공해 주었다.

5.2 코디네이터 선출 방식에 따른 지연 시간

그림 8에서 응답 지연 시간은 평균적으로 79.98% 증가하였다. 서비스 객체 개수가 5개 이하일 경우 73%, 서비스 객체가 6개를 넘어갈 경우 동적인 방식으로 코디네이터를 선출하는 것이 정적인 방법보다 응답 지연시간이 81% 증가하였다.



(그림 7) 코디네이터 유무



(그림 8) 코디네이터 선출 방식

6. 결론 및 추후 과제

본 논문에서는 동적 코디네이터를 이용한 결함허용 시스템을 제안하여 시스템의 가용성과 응답속도를 높이고자 하였다. 제안된 알고리즘의 성능을 실험해본 결과 코디네이터를 이용한 방식이 평균적으로 33% 빠른 응답속도를 보여주었다. 그러나 동적으로 코디네이터를 선출하는 방식은 정적인 방식보다 많은 메시지를 발생시켜 79.98% 지연된 응답속도를 보여주었다. 즉 서비스 객체들이 같은 LAN에 놓여있을 경우 동적인 방식이 더 비효율적임을 확인하였다.

향후 연구 과제는 시스템 가용성을 위하여 필요한 최적의 서비스 객체 수를 찾는 과정과 코디네이터 선출 알고리즘을 개선하기 위하여 지속적인 연구가 이루어져야 할 것이다.

참고문헌

[1] P. Felber and A. Schiper. “Optimistic active replication”,

[2] R. Guerraoui and A. Shiper. “Software-based replication for fault tolerance”, IEEE Computer, pp69-70, April 1997.

[3] R. Baldoni, C. Marchetti, and S. Tucci-Piergi ovanni. “Asynchronous Active Replication in Three-tier Distributed Systems”, PRDC 2002.

[4] Reliable group communication, http: // www. Jgrou ps.org/javagroupsnew

[5] A. Kumar, P. Jalote and D. Gupta, “A Fault -tolerant System for Java/CORBA Objects”2005

[6] R. Baldoni, C. Marchetti, and S. Tucci-Piergi ovanni. “A Fault-Tolerant Sequencer for Timed Asynchronous Systems”, LNCS 2400, pp. 578-588, Euro-Par 200