클러스터드 EJB 컨테이너에서 RMI Smart Stub의 구현

서범수[0], 정승욱, 김성훈, 노명찬
한국전자통신연구원 전자거래연구부
(bsseo[0], swjung, saint, mcroh)@etri.re.kr

The Implementation of RMI Smart Stub in the Clustered EJB Container

Beom-Su Seo[0], SeungWoog Jung, SungHoon Kim, MyungChan Roh
Electronic Commerce Department
Electronics and Telecommunications Research Institute

요 약

EJB(Enterprise Java Bean) 컨테이너에서 대규모의 요청을 위해서는 여러 개의 EJB 컨테이너를 클러스터링 환경으로 구동시키는 것이 필요하다. EJB를 사용하는 클라이언트로는 JSP나 에플리케이션 클라이언트, 혹은 또 다른 빈이 될 수 있다. 이러한 클라이언트가 사용하는 EJB의 스텁을 클러스터 환경에 적합하도록 구현함으로써 트랜잭션 지역화, 장애 발생시 다른 서버로의 메소드 호출 전환, 동일 가상 머신을 탐지한 메소드 호출 최적화 등의 다양한 기능을 수행할 수 있다. 본 논문에서는 이러한 스마트 스텁의 필요성 및 구현에 대해 논의한다.

1. 서 론

웹 에플리케이션 서버는 웹 요청을 받아 처리하는 웹 서버, JSP나 Servlet을 구동시키기 위한 웹 컨테이너, 비즈니스 로직을 담당하는 EJB 컨테이너, 그리고 EJB 컨테이너와 연동되어 트랜잭션이나 보안 등을 처리하는 미들웨어 시스템으로 구성된다. Apache와 같은 웹 서버의 경우 대규모 클라이언트 요청을 클러스터링 되어 있는 다른 웹 서버로 분산시키는 모듈을 자체적으로 가지고 있으며, JSP/Servlet 처리 엔진인 Tomcat과 같은 웹 컨테이너도 클라이언트의 Http 세션에 담겨진 세션 정보를 클러스터링 되어 있는 다른 Tomcat 엔진들에게 복제할 수 있는 메커니즘을 가지고 있다. 웹 에플리케이션 서버를 클러스터링 하기 위해서는 웹 서버, 웹 컨테이너, EJB 컨테이너들을 어떠한 토폴로지로 묶을 것인지를 결정해야 한다[2,3]. 크게는 웹 서버와 웹 컨테이너, EJB 컨테이너를 각각 클러스터링 하는 방식과 웹 컨테이너와 EJB 컨테이너를 개별적으로 클러스터링 하지 않고 하나로 묶어 클러스터링 하는 방법이 있다[2]. 본 논문에서는 다양한 클러스터링 토폴로지 중 웹 컨테이너와 EJB 컨테이너를 하나의 노드로 묶어 클러스터링하는 토폴로지를 사용한다. 그 이유는 이후 밝히도록 하겠다. 그림 1은 제안한 클러스터링 토폴로지를 나타낸다. 클러스터링을 지원하는 컴포넌트로써는 클러스터링 환경에서 통신을 담당하는 클러스터 매니저가 있고, 빈의 상태 및 JNDI 트리를 복제하기 위한 컴포넌트가 EJB 컨테이너에 존재한다. 이들은 빈들의 상태와 JNDI 트리를 다른 노드에 복제하여 클라이언트의 요청이 부하 분배기에 의해 다른 노드로 전달되어도 동일한 상태 정보를 가지고 지속적으로 서비스를 할 수 있도록 지원한다. 웹 컨테이너 측에는 클라이언트의 Http 세션에 저장된 정보들을 다른 노드들에 복제하기 위한 컴포넌트가 있다. 본 논문에 논의할 스마트 스텁은 Http 세션에 담겨 다른 노드로 복제될 수도 있으며, 하나의 빈이 안에서 다른 빈을 호출하는 형식으로 사용될 수도 있다. 본 논문에서는 클러스터링을 지원하기 위해 클라이언트가 사용하는 EJB 빈의 스텁에 어떤 기능이 필요하며 이러한 기능들을 구현하기 위해 고려해야 할 사항들에 대해 논의한다.

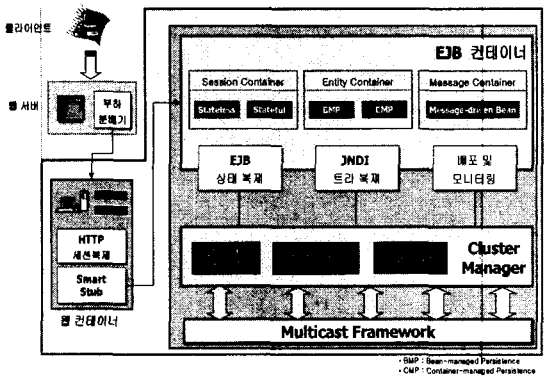

그림 1. 제안한 클러스터링 토폴로지

2. 스마트 스텁의 시나리오

클러스터링 토폴로지 환경에서 스마트 스텁은 크게 다음과 같은 기능을 수행한다.

첫째, 스마트 스텁은 메소드 호출 시 EJB 컨테이너에 장애가 발생할 경우 자신이 가지고 있는 다양한 정보를 이용하여 다른 노드에 존재하는 컨테이너에게 메소드 호출을 외뢰한다. 그림 2는 EJB 장애 탐지 후 메소드를 다른 노드로 전달하는 시나리오를 나타낸다. 이 경우 고려해야 할 사항이 있다. 우선은 호출되는 메소드가 다른 노드의 컨테이너를 호출하여 실행하여도 관계없는 메소드인지 혹은 클라이언트에게 정상적으로 오류로 보고해야 하는 메소드인지를 판단해야 한다. 읽기 위주의 메소드라면 다른 노드로 전달해도 관계가 없겠지만 다른 작업과 연관되어 쓰기 작업을 수행하는 메소드라면 클라이언트에게 오류 메시지를 보내는 것이 올바른 선택일 수 있다.

이것은 빈을 배포하여 시스템을 운영하는 관리자가 빈 배포시 배포 명세서에 이러한 정보를 명시하여야 하며, EJB 컨테이너는 스마트 스텁을 생성하여 클라이언트에게 전달하면서 이러한 정보를 넘겨준다. 또 다른 고려 사항은 스마트 스텁이 노드들에 대한 정보를 얻는 방식에 관한 것이다. 이 방식에는 정적 방식과 동적 방식이 가능하다. 정적 방식의 경우 배포시 미리 클러스터를 구성하는 서버에 대한 IP나 Port 정보를 설정한 후 스마트 스텁을 생성하면서 이러한 정보를 전달하는 방법이다. 동적 방식의 경우 노드 정보를 관리하는 객체와 실시간에 통신을 통해 클러스터를 구성하는 노드에 대한 정보를 가져오는 방법이다. 이 경우 노드에 대한 정보를 가진 객체가 다운 될 경우 메소드 단위의 호출이 불가능하게 된다는 단점이 있지만 클러스터를 동적으로 구성하는 경우 유용한 방식이다.

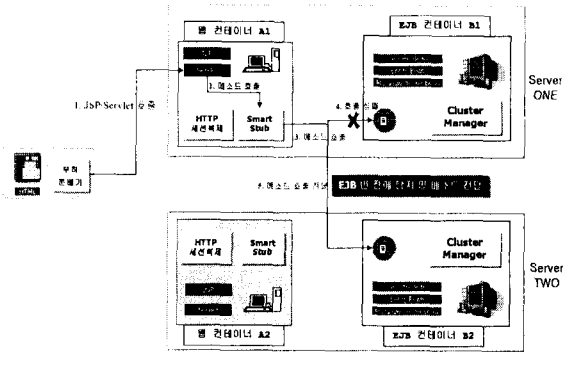

그림 2. 스마트 스텁의 장애 탐지 및 메소드 전달

둘째, 스마트 스텁은 이전 메소드 호출에서 종료되지 않은 트랜잭션이나 데이터베이스 커넥션이 존재한다면 연속된 메소드 호출은 트랜잭션이나 커넥션을 시작한 서버에게 의뢰하는 트랜잭션(커넥션) 지역화를 수행한다. 이러한 시나리오는 다음과 같은 상황에서 발생이 가능하다. 클라이언트가 JSP에서 UserTransaction을 시작하였거나 데이터베이스 커넥션을 얻고 EJB 스텁을 장바구니와 같이 Http 세션에 담았을 경우, 웹 컨테이너는 해당 Http 세션을 클러스터를 구성하고 있는 노드들에게 복제할 것이고, 웹 서버의 부하 분배기에 의해 다음 클라이언트의 요청이 다른 노드로 전달될 수 있다. 이때 해당 노드에 존재하는 복제된 스마트 스텁에서는 이전 메소드 호출 시에 트랜잭션이나 커넥션이 완료되지 않았음을 감지하고 자신을 최초 생성한 컨테이너를 호출한다. 이것은 WebLogic[4]과 같은 상용 서버에서도 동일한 방식을 취하는데, 트랜잭션이나 커넥션을 복제하는 것 자체가 많은 오버헤드를 유발시키며, 복제를 한다고 하더라고 트랜잭션 롤백이나 커넥션에 대한 롤백 정책이 더 많은 문제를 야기하기 때문이다. EJB 스펙[1]에서는 트랜잭션이나 커넥션을 얻은 후 다른 메소드에서 이를 사용한 후 또 다른 메소드에서 이를 종료할 수 있도록 되어 있기 때문에 스마트 스텁은 컨테이너에게 메소드 호출이 끝나면 해당 메소드에서 트랜잭션이 종료되었는지, 커넥션이 열려 있는지를 의뢰한 후 이 정보를 이용하여 다음 메소드를 호출 시 활용한다. 그림 1과 같은 토폴로지에서는 웹 컨테이너에서 EJB 컨테이너로 로컬 메소드 콜을 사용하여 이 정보를 얻을 수 있다. 다른 형태의 토폴로지를 적용한다면 이를 위한 다른 통신 메커니즘이 필요하며 이에 따른 오버헤드도 발생한다. 그림 3은 이러한 과정을 나타낸다.

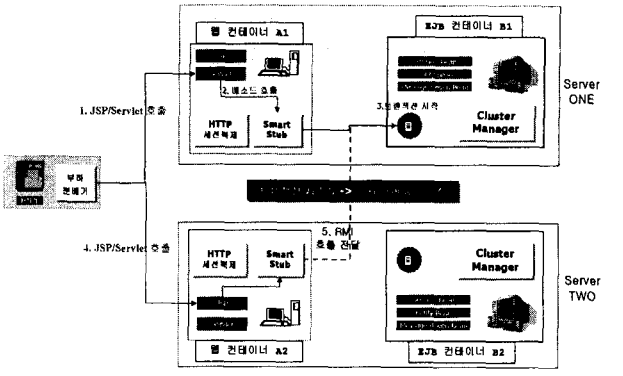

그림 3. 스마트 스텁의 트랜잭션 지역화

셋째, 스마트 스텁은 EJB 컨테이너와 같은 가상 기계에 존재한다면 RMI 메소드 호출을 사용하는 것이 아니라 로컬 메소드 콜을 사용하여 소켓 및 네트워크 자원을 현저히 감소시킨다. 모든 메소드 호출은 RMI 프로토콜을 따르므로 클라이언트가 서버를 호출할 때 서버에 접속하기 위한 소켓이 필요하다. OS에 따라 차이는 있지만 Windows 계열의 장비들은 동시에 사용할 수 있는 소켓에 제한이 있어 대규모의 요청시 소켓 연결 오류가 발생한다. EJB는 서버 측 컴포넌트이다. 즉, 한 빈의 클라이언트는 대부분 동일한 가상 기계에 존재하는 다른 EJB이다. 결과적으로, 컨테이너 내부적으로 동일 기계임에도 불구하고 RMI 프로토콜에 따라 네트워크 자원을 사용한다면 소켓 자원 및 호출 시간에 지대한 영향을 미치게 된다. 스마트 스텁은 그림 3의 경우가 아니라면 자신과 동일한 가상 기계에 존재하는 EJB 컨테이너을 인메모리 메소드 콜로 호출함으로써 소켓 자원의 절약과 메소드 호출시간을 최적화한다. 이것이 그림 1에서 제시한 토폴로지를 사용한 주요한 이유 중 하나이다. 웹 컨테이너와 EJB 컨테이너가 각기 다른 프로세스 혹은 다른 기계에서 구동된다면 발생하는 소켓의 오버헤드, 메소드 호출을 위해 RMI 내부적으로 수행하는 마샬링 및 언마샬링 작업 등에서 발생하는 다양한 오버헤드로 그림 1의 토폴로지에 비해 현저한 성능 저하가 예상된다. 물론 그림 1의 토폴로지의 경우 웹 컨테이너 혹은 EJB 컨테이너 중 하나만 시스템 다운되어도 해당 노드를 사용할 수 없는 단점이 있지만 이것은 더 많은 노드로 클러스터를 구성함으로써 해결이 가능하다. 그림 4는 이러한 메소드 호출 최적화 과정을 나타낸다.

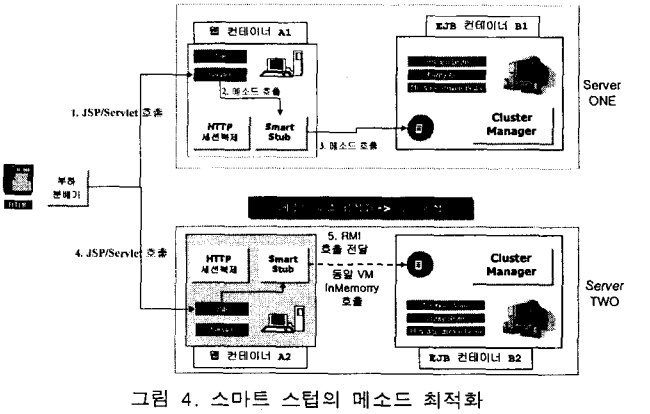

그림 4. 스마트 스텁의 메소드 최적화

3. 스마트 스텁의 구현

EJB를 사용하는 클라이언트는 다음과 같은 형태로 EJB를 사용한다.

```
InitialContext ctx = new InitialContext();
Object obj = ctx.lookup("LoginHome");
LoginHome loginHome =(LoginHome)
        PortableRemoteObject.narrow(obj,LoginHome.class);
Login login = loginHome.create();
login.myLogin(userId, passwd);
```

여기서 클라이언트가 가지고 있는 LoginHome이나 Login 객체가 실제로는 LoginHome과 Login 인터페이스를 구현하면서 2절에서 논의한 다양한 기능을 수행해야 한다. 이를 위해 자바의 reflection 메소드에서 제공하는 Proxy 클래스와 InvocationHandler 인터페이스를 사용한다. Proxy.newProxyInstance(ClassLoader, TargetClass, TargetObject)를 통해 반환되는 객체는 TargetClass로 캐스팅 될 수 있다. TagerObject는 InvocationHandler 인터페이스를 구현한 객체로써 TargetClass에 정의된 모든 메소드는 TargetObject의 invoke() 메소드로 호출이 전달된다. 스마트 스텁은 크게 홈 스텁과 리모트 스텁으로 구분된다. 이제 스마트 스텁의 생성 과정에 관여하는 Proxy 클래스의 이름을 따라 리모트 스텁을 SmartRemoteProxy로, 홈 스텁을 SmartHomeProxy로 사용하도록 한다. 그리고, 이들을 통칭할 경우는 스마트 스텁을 사용하도록 하겠다. LoginHome이나 Login 객체는 Proxy 객체를 통해 얻어진 프록시 객체로써 LoginHome, Login 클래스처럼 동작하며 모든 메소드 호출은 TargetObject인 SmartRemoteProxy, SmartHomeProxy의 invoke() 메소드로 전달된다. 이 메소드 내에서 2절에서 논의한 다양한 기능을 검사하고 수행할 수 있다. 그림 5는 Login EJB의 SmartRemoteProxy의 생성 및 호출 과정을 나타낸다. 여기서 SmartProxyFactory가 앞서 논의한 Proxy 객체를 생성하고 컨테이너가 가지고 있는 다른 노드에 대한 정보, 메소드에 대한 정보 등을 SmartRemote(Home)Proxy에 설정하여 클라이언트에게 전달하는 객체이다. 스마트 스텁 안에는 컨테이너와의 통신을 담당하는 RemoteInvocation이라는 객체가 존재한다. 이 객체는 스마트 스텁의 invoke() 메소드 안에서 컨테이너를 호출하기 위한 통신을 담당하는 RMI 객체이다. 클라이언트의 메소드 호출은 RemoteInvocation.invoke()를 통해 컨테이너의 RemoteInvocationImpl.invoke()로 전달되어 컨테이너를 호출한다. 그리고 컨테이너로부터 호출 결과를 받은 후 이를 InvocationResult라는 객체에 담아서 다시 스마트 스텁에게 전달하는데 이때, InvocationResult에는 컨테이너에서 전달된 호출 결과와 트랜잭션 종료 정보, 열려있는 커넥션에 대한 정보 등이 포함된다. 스마트 스텁이 부하 분산 정책에 의해 다른 노드로 복제가 된 후 메소드가 호출될 때, 트랜잭션이나 커넥션이 열려 있다면 로컬 컨테이너가 아닌 최초 RemoteInvocation이 생성되었던 컨테이너가 호출된다. 시작된 트랜잭션이나 열려진 커넥션이 없는 경우, 스마트 스텁 안에서는 최초 생성 당시 자바 VM ID와 메소드 호출 시점에서의 VM ID를 검사하여 동일 VM일 경우 컨테이너에 대한 레퍼런스를 시스템으로부터 얻어와 RMI 호출을 하지 않고 직접 로컬 컨테이너를 호출한다. SmartHomeProxy의 경우는 JNDI를 다른 형태로 조작하여 SmartHomeProxy를 얻을 수 있다[5]. 컨테이너는 구동 시 빈의 홈 스텁들을 JNDI에 바인딩함으로써 클라이언트가 룩업하여 사용하도록 준비한다. 홈 스텁의 경우 이름과 함께 홈 스텁을 바인딩하는 것이 아니라 RemoteInvocation RMI 스텁을 JNDI에 바인딩한다. 클라이언트에서 이를 룩업할 때 RegistryContext.lookup()에서는 SmartProxyactory를 통해 SmartHomeProxy를 생성하여 클라이언트에게 전달한다. 즉, SmartRemoteProxy는 서버에서 생성되지만 SmartHomeProxy는 클라이언트에서 생성되어 전달되며 이때 필요한 정보를 RemoteInvocation을 통해 서버로부터 얻어 사용한다.

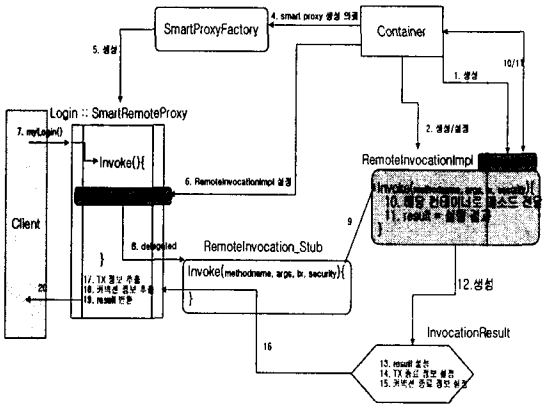

그림 5. SmartRemoteProxy의 구동 방법

4. 결론

본 논문에서는 EJB 클러스터링 환경에서 트랜잭션을 지역화하고, 메소드 레벨에서 다른 클러스터 노드에게 메소드를 전달하거나 컨테이너와 동일 VM에 존재할 경우 RMI 등의 통신 프로토콜을 거치지 않고 메소드를 호출할 수 있도록 지원하는 스마트 스텁에 대해 논의하였다. 향후 다양한 형태의 클러스터링 토폴로지 환경으로 스마트 스텁을 확장하여 구현하는 것이 필요하다. 지면 관계상 많은 부분을 논의하지 못하였지만 다른 논문을 통해 SmartHomeProxy 구현을 위한 JNDI의 조건과 고려 사항, 동일 VM 결정 메커니즘, 동적 및 정적 클러스터링 환경에서 대체 노드 결정 방법 등에 대해 추가로 논의하도록 하겠다.

5. 참고문헌

[1] Linda G. Demichiel 외 2인, "Enterprise Javabeans Specification, Version 2.0," Final Release, Sun Microsystems, 2001

[2] Abraham Kang, "J2EE Clustering, Part 1," http://www.javaworld.com/javaworld/jw-02-2001/jw-0223-extremescale_p.html

[3] Abraham Kang, "J2EE Clustering, Part 2," http://www.javaworld.com/javaworld/jw-08-2001/jw-0803-extremescale2_p.html

[4] BEA WebLogic Server 6.0, http://www.bea.com.products/weblogic/server/index.shtml

[5] Guy Gur-Ari, "Empower RMI with TRMI," http://www.javaworld.com/javaworld/jw-08-2002-jw-0809-trim_p.html