분산객체환경에서의 JDBC 3-tier 모델의 성능확대와 CORBA의 이용

이 진 용† · 전 순 미††

요 약

이질적인 시스템으로 구축된 분산 객체 환경에서 자바로 개발된 응용 시스템과 데이터베이스 관리 시스템의 연동성은 매우 중요하다.

연동을 지원하는 기존의 JDBC 제공 모델 중에서 분산환경에서 가장 많이 쓰이는 구조가 JDBC 3-tier 모델인데 이 모델은 자바애플릿 응용시스템의 적용에서 제한적인 보안 구조를 가진 연동관계의 형성을 요구한다는 것과 또한 클라이언트에서 미들웨어로 접속하는 규약의 종류에 따라서 매번 응용프로그램에서 별도의 설계 및 구현을 해야하는 부분이 부담으로 나타나는 문제점을 안고 있다. 본 논문에서는 이 문제들을 해결하기 위해서 CORBA 분산객체 환경을 이용하였으며 기존의 JDBC 모델을 지원하는 ORB-JDBC 하위 규약과 미들웨어를 설계하고 구현하여 기존모델의 성능을 확대시켰다. 즉, 새로 확대된 모델은 연동을 위해서 구현된 어떠한 자바 응용 프로그램에서도 시스템에 대한 보안 규제가 극복되며 또한 데이터베이스 관리시스템과의 연동관계에서 간단하게 응용프로그램만을 작성하는 것으로 연동의 작업이 이루어지게 되어 클라이언트의 제작비용과 시간을 줄여준다. 동시에 CORBA가 제공하는 DBMS 서버들 사이의 부하분배를 지원하는 기능도 있음을 모의실험을 통해서 확인한다

Property Enlargement of JDBC 3-tier Model Using CORBA Under Distributed Object Environments

Jin Yong, Lee† · Sun Mi, Jeon††

ABSTRACT

The property of connectivity between DBMS and application system developed in Java is very important under the distributed object environments with heterogeneous systems.

In these days, mostly used JDBC 3-tier model, which supports the connectivity, has two problems: the one is the restricted connectivity needed for the Java applet as application programs and the other is the additional efforts to design and implement in linking any application program with the middleware from being deficit of standard protocol.

In order to solve these problems, we have worked on Enlarging the properties of JDBC 3-tier model by applying CORBA distributed object environments.

Thus, the enlarged 3-tier model has an ORB-JDBC sub-protocol for the client sub-protocol and a middleware whose property is supported by CORBA.

† 준 회 원 : 인제대학교 정보컴퓨터학부
†† 정 회 원 : 인제대학교 정보컴퓨터학부 교수
논문접수 : 1998년 2월 10일, 심사완료 : 1998년 7월 9일

Hence, the model can be applied to JAVA Applet without considering the inherent security property as the JAVA Application and saved the extra efforts when developing any application systems in JAVA.

Moreover, the ability of load balancing provided by CORBA between database servers is confirmed on the suggested model through the simulation test comparing with the previous one.

1. 서 론

인터넷과 같은 수많은 종류의 컴퓨터가 존재하는 분산적이고 이질적인 전산망 환경에서의 응용 시스템 구현은 많은 비용을 요구한다[1]. 이러한 환경에서 자바 언어는 가상 기계라는 개념을 이용하여 이질적 시스템 환경에서 플랫폼(platform)과는 독립적으로 구현이 가능하게 해주는 기능을 제공해주고 있으며 또한 CORBA (Common Request Broker Architecture)[2][3][4]는 분산객체 환경을 위한 표준으로 자바언어와 연계하여 응용 시스템을 구축하는 경우에 많은 구현 비용을 절감시켜준다.[5][6]

오늘날 대부분의 응용 시스템 활용에서 데이터베이스 시스템과의 연계작업은 거의 필연적이다. 자바언어는 데이터베이스 시스템과의 자료수수를 위해 JDBC API(Application Programming Interface)[7]라는 표준 API를 지원하는데 자바의 데이터베이스 접근 규약(protocol)인 JDBC는 응용 프로그램을 갖는 클라이언트 시스템과 데이터베이스 시스템을 탑재하는 서버시스템과의 연계를 위해 2-tier와 3-tier 모델[8]을 제공해준다. 이 중에서 분산 객체 환경에서는 광역의 데이터베이스 지원을 위해서 클라이언트와 서버 시스템사이에 미들웨어를 거치는 3-tier 모델이 반드시 요구된다. 그러나 접속하려는 미들웨어와의 사이에 형성되는 규약에 대한 표준이 없으므로 이의 변화에 따라서 응용프로그램을 구현할 때마다 다시 적용해야하는 부담과 어려움이 있다. 또 이 모델은 인터넷 환경에서 자바애플릿과 자바애플리케이션의 구현을 지원한다. 그러나 자바애플릿에 대하여는 제약을 가진 보안모델을 요구하므로 미들웨어가 데이터베이스 시스템에 종속적으로 존재해야 한다는 결점을 가지고 있어 분산환경에서 구현에 어려움이 있다. 본 논문에서는 기존의 JDBC 3-tier 모델이 갖고 있는 이들 문제점들에 대한 해결방안으로서 CORBA를 기반으로 하는 개선된 JDBC 3-tier 모델을 검토하려 한다. 그 이유는 CORBA의 분산객체 개념을 도입한 미들웨어를 구현함으로 클라이언트에서 웹(World Wide Web)서버를 거치지 않고 새로 구축된 미들웨어를 거쳐 바로 데이터베이스 시스템으로의 입출력(input-output)이 가능하므로 웹서버의 종속성의 문제를 벗어날 수 있으며 따라서 자바애플릿에게 요구되었던 보안구조의 벽도 극복할 수 있다.

또한 JDBC를 지원하는 'ORB-JDBC'라는 하위규약(sub_protocol)를 설계하고 구현함으로 클라이언트에서 CORBA가 제공하는 미들웨어로의 연결을 위한 부담이 제거되어 응용프로그램의 작성이 간단하여진다. 추가적으로 본 논문에서는 CORBA를 이용하여 성능을 확대시킨 JDBC 3-tier 모델이 응용 시스템의 적용에서 위의 문제점들을 해결하는 것 외에도 데이터베이스 관리시스템 서버들 사이의 부하(load)에 대한 분배 능력의 지원도 모의실험을 통하여 확인한다.

2장에서는 관련 연구로서 JDBC 3-tier 모델에 대하여 알아보고 그 문제점을 지적하며, 3장에서는 문제해결을 위한 방법 제시와 설계를 기술하며 4장에서는 그 구현과 평가를 보인다. 끝으로 5장에서 결론과 향후 연구 방향을 제시한다.

2. JDBC 3-tier 모델

2.1 JDBC 하위규약의 형식

JDBC의 하위규약은 데이터베이스 관리 시스템과의 연결 방식에 따라 다음같이 크게 네 가지로 나누고 있다.[8]

- JDBC-ODBC Bridge plus ODBC driver : 이는 ODBC(Object Database Connectivity) 관리자로의 연결을 생성하여 ODBC 드라이버를 통해서 데이터베이스 관리 시스템에 연결하는 형식이다.
- Native API partly-Java driver : 이는 JDBC 호출을 데이터베이스 관리 시스템 고유의 연결 규약으로 변환하여 연결하는 형식이다.
- JDBC-Net pure Java driver : 이는 JDBC 호출을 데이터베이스 관리 시스템과는 독립적인 규

약으로 번역하고 이 호출을 처리하는 미들웨어를 구현하여 순수하게 자바로 구현된 클라이언트 응용 프로그램을 다양한 데이터베이스 관리 시스템과 연결 시켜주는 형식이다.

- Native-protocol pure Java driver : 이는 JDBC 호출이 데이터베이스 관리 시스템의 규약으로 곧바로 인식되도록 구현된 형식이다.

자바언어의 플랫폼 독립성과 분산 객체에 대한 지원을 가장 이상적으로 지원하는 방법은 순수한 자바언어(Pure Java)로 구현된 경우이지만 처리 속도 면에서는 데이터베이스 관리 시스템 자체의 규약을 이용하는 것이 상대적으로 더 효과적이다.

본 논문에서 설계한 'ORB-JDBC 하위규약'은 클라이언트와 미들웨어의 관계에 있어서는 'Native-protocol pure Java drivers'의 구현형식을 이용하고 있다. 이는 앞에서 기술한 바대로 이질적 분산환경에서 시스템의 독립성을 극대화 할 수 있으나 그 연결형식은 분산환경에서는 적합하지 아니한 2-tier 방식이다.

2.2 3-tier 모델의 구조와 문제점

(그림 1)은 JDBC에서 제공하는 2가지 모델을 나타낸 것이며 이 중에서 JDBC 3-tier 모델은 그림에서와 같이 클라이언트 응용 프로그램이 미들웨어 역할을 하는 응용 서버를 통하여 데이터베이스에 연결되는 구조이다[8]. 여기서 미들웨어의 역할은 클라이언트와 자체적으로 약속된 규약을 이용하여 통신을 하고 다른 한편으로는 클라이언트의 요구를 JDBC 관리자 계층을 통하여 데이터베이스에 전달하고 수행된 결과를 받아 다시 클라이언트에로 되 전달하는 일을 한다. 미들웨어는 데이터베이스에서 제공하는 규약으로의 빠르고 직접적인 연결을 하기 위해 주로 'Native API partly-Java driver' 형식의 JDBC 하위 규약을 사용하는 쪽이 우수하다 그러나 이러한 JDBC 하위규약은 클라이언트/서버의 2-tier 모델을 기본방식으로 하고있어 미들웨어를 이용하는 3-tier의 경우에 이용하려면 다른 네트워크 규약을 이용해서 구성해야한다.

일반적으로는 미들웨어에서 클라이언트와의 통신을 하기 위해 HTTP, 자바 RMI(Remote Method Invocation), 또는 CORBA를 이용하는 경우가 많다.

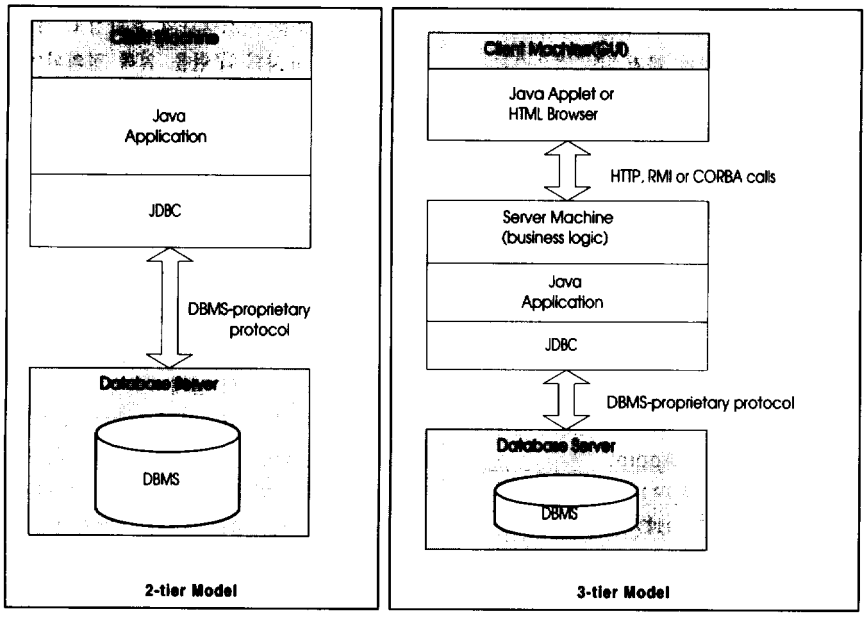
다양한 미들웨어의 사용으로 기존의 JDBC 3-tier 모델을 실제로 적용하는데 다음과 같은 두 가지 문제점을 가지고 있다. 첫째는 클라이언트와 미들웨어사이에 통신을 위한 표준 규약이 없다는 것이다. 따라서 응용 시스템의 구현을 위해서는 그때마다 반복적으로 미들웨어의 구현과 규약을 설계해야하기 때문에 구현 비용의 증가와 시간의 낭비를 피할 수가 없으며 이러한 문제는 이질적인 시스템이 광역화 될 수록 더욱 커지게 마련이다. 둘째는 자바애플릿을 이용한 응용 시스템에 적용하는 경우에 보안상의 제약 때문에 미들웨어는 언제나 웹서버와 동일한 시스템에 존재하여야 한다는 것이다. JDBC 3-tier 모델이 데이터베이스의 위치에 대한 투명성(transparency)은 보장해 주지만 반대급부로 미들웨어의 위치가 서버에 종속적이고 실제 자료는 미들웨어를 통하여 클라이언트에 전달되므로 웹서버와 미들웨어가 함께 존재하는 시스템의 조건에서는 웹서버에의 부하 집중에 대한 위험성은 본질적으로 내재되어 있는 것이다.

2.3 CORBA

OMG(Object Management Group)에서 제정한 OMA (Object Management Architecture)는 분산 객체 환경에 대한 표준으로서 네트워크 상에 분산된 객체들의 전송을 위해 CORBA라는 객체 버스(Object Bus)를 제공한다. 즉, OMA는 응용 객체(Application Objects), 일반 기능(Common Facilities), 객체 서비스(Object Services)와 가장 중심적인 부분으로서 이런 객체들의 버스 역할을 하는 ORB(Object Request Broker)를 포함하고 있는 CORBA로 이루어져 있다.

CORBA 2.0 규정 이후로 인터넷 환경에서 ORB 간의 통신에 대한 표준인 GIOP(General Inter-ORB Protocol)가 정하여 졌고 이 가운데서도 TCP/IP를 기반으로 하는 규약에 대하여는 IIOP(Internet Inter-ORB Protocol)라고 명명하고 필수적으로 구현하도록 요구하고 있다. 이를 통하여 기존 CORBA 제품들 간의 상호작동이 가능하고 인터넷 환경에서는 ORB를 이용하여 보다 폭넓은 분산 객체에 대한 지원이 가능하다. 즉, 기존의 웹 클라이언트 내부에 ORB를 포함시켜 웹 클라이언트 상에서 수행되는 CORBA와 자바를 이용한 응용 프로그램이 웹 서버에 접근하지 않고 ORB와 IIOP의 기능으로 분산 객체에 대한 직접적인 접근이 가능하게 되었다.[2][6]

CORBA는 분산객체들 사이의 통신을 위해 기존의 프로그래밍 언어와는 독립적인 IDL(Interface Definition Language)을 정의하여 이용함으로써 클라이언트



(그림 1) JDBC 2-tier 모델과 3-tier 모델
(Fig. 1) JDBC 2-tier model and 3-tier model

에서의 객체 구현(Object Implementation)이 다양한 프로그래밍 언어로 가능하도록 지원하고 있다. 특히 새로운 CORBA2.0 규정에서는 자바언어에 대한 IDL 매핑(mapping) 표준안이 제정됨으로써 자바언어를 이용한 CORBA 분산 객체 환경의 구현이 가능하여 졌다. 이는 자바언어의 플랫폼 독립적인 특성과 CORBA의 분산 객체에 대한 지원을 통합할 수 있게 하여 이질적 시스템이 함께 존재하는 분산 네트워크 환경에서 보다 생산성 높은 응용 시스템 구축을 위한 해결방안이 가능해 진 것이다.

3. 하위 규약의 설계

3.1 JDBC 3-tier 모델의 문제 해결

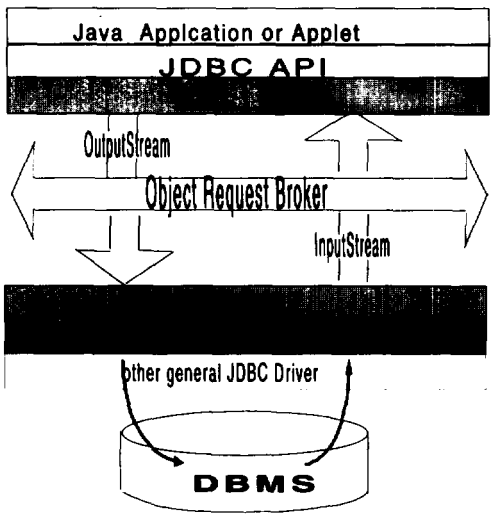
앞에서 지적한 JDBC 3-tier 모델의 문제점을 해결하기 위해서 본 논문에서는 CORBA를 이용한 클라이언트와 미들웨어간 규약을 설계하고 구현한다.

이 규약은 기존의 JDBC 3-tier 모델에서 데이터베이스 관리시스템 서버에 따라 달리 사용해야 하였던 별도의 특정 규약과는 달리 JDBC의 하위 규약으로 작용하도록 설계되었으며 따라서 JDBC API 표준에 의거한 클라이언트 응용 프로그램의 구현만으로 바로 JDBC 3-tier 모델의 적용이 가능하며 또한 응용프로그램을 구현 할 때마다 발생되던 미들웨어의 반복적인 구현부분을 제거시켜 문제점을 해결한다. 이 방법에서는 웹서버를 전혀 거치지 아니하고 클라이언트에서 미들웨어를 거쳐 데이터베이스 관리시스템에 이르도록 설계되어 있어 기존의 JDBC 3-tier 모델이 안고 있었던 웹서버의 종속성문제를 해결하며 이와 함께 자바애플릿에서 나타나는 보안상의 제약이라는 부분도 완전하게 극복되어 진다.

더욱이 CORBA를 이용한 이러한 접근은 웹 클라이언트 내에 포함된 ORB를 이용함으로써 미들웨어의 위치에 대한 서버로부터의 독립성을 가질 수 있도록 해준다. 이러한 미들웨어에 대한 위치 독립성의 제공은 CORBA가 기본적으로 제공하는 부하 분배 기능과 함께 하며 이는 미들웨어에 접근하는 클라이언트를 적절히 분배시켜 주므로[9] 데이터베이스 관리시스템의 서버에 대한 적절한 부하분배(load balancing)의 이점도 가져온다.

3.2 ORB-JDBC 하위 규약

응용 시스템에서 미들웨어로의 연결 방법은 여러 가지가 있지만, HTTP는 웹에만 한정하는 경우라고 볼 수 있고 연속적인 연결 기능이 없는 등의 문제점을 가지며, 자바 RMI의 응용은 아직 표준화되지 않은 방법이다. 따라서 본 논문에서는 분산 객체 환경을 지원하는 CORBA를 이용한 JDBC 3-tier 모델의 구성을 설계한다.[10] (그림 2)는 본 논문에서 제안하는 성능이 확대된 JDBC 3-tier 모델의 시스템 구성도이며 이것은 자바를 이용한 기존의 모든 응용시스템이 'ORB-JDBC' 하위 규약을 이용하도록 되어 있다.



(그림 2) ORB-JDBC 하위프로토콜을 이용한 응용 시스템의 구성

(Fig. 2) Composed application system using ORB-JDBC sub-portocol

그림에서 음영 처리된 부분이 본 논문에서 구현한 부분이다. 응용 시스템과 미들웨어의 연결을 위해서 CORBA를 이용하여 구성한다. 이러한 연결을 위해서는 JDBC 표준 API의 인터페이스(interface) 객체를 CORBA IDL의 인터페이스로 전환하여 정의한다. 이 설계는 응용 시스템과 미들웨어의 연결을 위해 별도의 표준 정의 단계 없이 기존의 JDBC 표준 API 내부에서 연결을 정의하게 해준다. 즉, JDBC 표준 API 집합을 IDL 인터페이스로 대응시킴으로써 응용 시스템에서는 일반 JDBC 하위 규약을 이용한 구현과 같은 방법으로 구현이 가능하게 되며 동시에 이미 만들어진 기존의 응용 시스템과의 호환성도 갖게되는 것이다.

하위 규약의 구현을 위해 정의되어야하는 JDBC 표준 API에서 제시하는 인터페이스 객체의 종류를 열거하면 Driver, Connection, Statement, CallableStatement, PrepareStatement, ResultSet, ResultSetMetaData, DatabaseMetaData 이다. 이러한 객체를 그대로 이용하여 IDL을 구현함으로써 응용 시스템 내의 JDBC API 호출을 곧바로 미들웨어의 객체 구현의 메소드 호출로 대응시킬 수 있는 것이다.

3.3 미들웨어의 설계

미들웨어는 클라이언트에 대하여 객체 구현(Object Implementation)상태로서 작동한다. 즉, IDL 컴파일러에 의해 생성된 객체 구현 모듈로 구성되게 된다. JDBC API의 인터페이스 객체를 IDL 컴파일러로 컴파일하여 생성된 객체 구현 모듈은 데이터베이스 시스템에 연결하고 질의하며 그 결과를 처리하기 위한 또 다른 JDBC 하위 규약의 프로시져에 대하여 1 : 1로 대응하게 된다.

응용프로그램의 실행 시에 명령줄(command line)에서 입력할 JDBC 하위 규약의 드라이버를 미들웨어가 인지하며 이를 메모리에 적재한 후 클라이언트와의 연결을 위해 대기하게 된다. 앞에서 기술한 바와 같이 CORBA로 구현된 미들웨어는 웹 서버의 위치에 관계없이 탑재될 수 있고 동시에 클라이언트로부터 데이터베이스 관리시스템의 서버로의 접근에 대한 부하의 할당문제도 CORBA 자체가 제공하는 분산객체에 대한 부하 분배 기능으로 부하편중의 위험이 제거된다.

4. 하위 규약의 구현

4.1 구 현

본 논문에서는 CORBA 구현 제품으로 Visigenic Inc.의 VisiBroker for Java를 이용하였다. 클라이언트와 미들웨어의 구현 도구로 Sun Microsystems Inc.의 JDK (Java Development Kit) 1.1.x를 이용하였다. VisiBroker라는 제품은 Netscape Communicator에서 직접 지원하는 CORBA 구현 제품으로서 웹에서 상당히 폭넓게 이용할 수 있는 제품이다.

'ORB-JDBC 하위 규약'은 CORBA 2.0 명세의 자바언어 매핑에 기준하여 IDL을 정의하고 이를 Visi-

```
interface Driver {
        Connection connect(in string url, in string info);
        boolean acceptsURL(in string url);
        string getPropertyInfo(in string url, in string info);
};

interface Connection {
        Statement createStatement();
        PreparedStatement prepareStatement(in string sql);
        CallableStatement prepareCall(in string sql);

        string nativeSQL(in string sql);
        boolean setAutoCommit(in boolean autoCommit);
        commit();
        rollback();
        close();
        boolean isClosed();

        DatabaseMetaData getMetaData();
        setReadOnly(in boolean readOnly);
        boolean isReadOnly();
        setCatalog(in string catalog);
        string getCatalog();

        setTransactionIsolation(in long level);
        long getTransactionIsolation();
        string getWarnings();
        clearWarnings();
};
```

(그림 3) 'Driver'와 'Connection' 인터페이스 객체를 위한 IDL 정의
(Fig. 3) IDL definition of interface object - 'Driver' & 'Connection' -

Broker for Java의 IDL 컴파일러를 이용하여 컴파일한 후 JDBC API 표준에 맞추어 구현된 클라이언트측의 IDL stub 부분과 미들웨어를 구성하는 객체구현 부분을 함께 JDK의 자바컴파일러를 통하여 컴파일 함으로써 최종적으로 구현이 완결된다.

'ORB-JDBC 하위 규약'의 구현을 위해 정의된 IDL 가운데 'Driver'와 'Connection' 인터페이스 객체에 대한 정의는 (그림 3)과 같다.

JDBC 표준 API의 인터페이스 객체 내에 대부분의 메소드들에 대한 정의를 포함하고 있으므로 이러한 구현은 라이언트 응용 프로그램에서 포함해야하는 JDBC 하위 규약에 관한 자바클래스 모듈의 구현을 간편하게 해주는 장점과 함께 그 크기도 작아짐으로써 클라이언트의 전체 크기를 줄일 수 있다. 특히 자바애플릿을 이용한 클라이언트 응용 프로그램의 구현 시에는 응용 프로그램의 전체 크기의 축소가 중요한 고려 사항 가운데 하나이므로 매우 유리하다.

미들웨어의 구현은 다음의 두 과정으로 이루어진다. 첫째는 데이터베이스 시스템과의 연결을 위해서 사용되는 다른 일반 JDBC 하위 규약부분을 적재하는 것이며 둘째는 'ORB-JDBC 하위 규약'을 통해 이루어지는 원격 프로시져 호출을 데이터베이스 시스템과 연결된 다른 JDBC 하위 규약의 메소드와 연결시키는 것이다. 이를 위해서 미들웨어는 CORBA를 위한 초기화를 수행하고 명령줄에서 입력받은 JDBC 하위 규약의 이름 및 URL을 이용하여 해당 하위 규약부분을 기억 장치에 적재한 후, 클라이언트의 원격 프로시져 호출을 대기하는 상태로 되는 것이다.

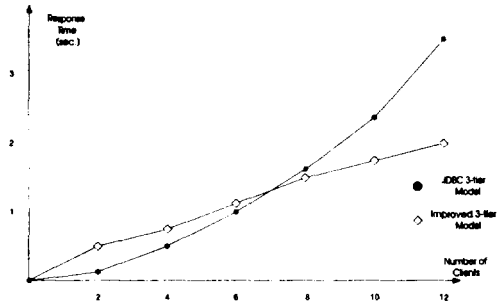
4.2 적용 및 평가

인터넷 환경과 같은 다양한 이질적 시스템이 혼재하는 경우에 자바를 이용한 응용 시스템의 구현이 비용과 시간을 절약하는 좋은 방법이 된다. 특히 데이터베이스 시스템과의 연동 요구가 발생할 때, 본 논문에서 설계 및 구현한 'ORB-JDBC 하위 규약'과 미들웨

어는 이미 작성된 자바 응용 시스템이 존재하는 경우에도 쉽게 적용이 가능하다. 더구나 자바애플릿의 보안구조에 대한 제한사항도 CORBA를 이용한 JDBC 3-tier 모델에서 극복되므로 웹 클라이언트를 사용자 인터페이스로 하는 경우에도 적용이 가능하다.

이제, JDBC 3-tier 모델의 여러 성능 가운데서 기존의 모델과 CORBA를 이용하여 확대된 모델 사이에 웹을 통하여 정보를 제공하는 경우 전송 속도의 차이를 비교 평가하기 위해서 시행한 모의실험에 대해서 기술한다. 평가 방법은 Solaris 2.4와 2.5를 기반으로 하는 두 대의 SUN Sparc 계열의 시스템에 각각 설치한 PostgreSQL 데이터베이스 시스템과 연동하여 정보를 제공하는 웹서비스에서 접속하는 클라이언트의 수가 점차 증가하는 경우에 서비스 제공속도를 비교 평가한다.

PostgreSQL에서는 3만여 영어단어와 그 한글의미를 가지고 있는 자료를 제공하고 자바로 만들어진 응용프로그램이 JDBC API를 통해 사용자가 입력한 단어의 의미를 찾아오는 것으로 비교평가 환경을 구축하였다. 실제로 미리 50개의 영어단어를 제공하고 이들을 프로그램 내에서 차례로 읽어들여 자바의 멀티쓰레드의 기능을 이용해 생성된 각 쓰레드가 독립적으로 동시에 데이터베이스에 질의를 전달하고 이에 대한 의미를 가진 한글단어를 가져오는 과정을 반복하였다. 이에 대한 결과는 (그림 4)에 나타나 있다. 기존의 3-tier 모델의 경우에는 클라이언트의 수가 증가하여 서비스 요청이 늘어나면 서비스의 속도가 현저히 감소하나 본 논문에서 제시한 'ORB-JDBC 하위규약'과 미들웨어를 이용하여 성능이 확대된 3-tier 모델의 경우에는 그 감소 정도가 상대적으로 비교적 일정하였다. 이러한 결과는 응용프로그램의 구현의 편의성과 더불어 분산 객체로서의 데이터베이스에 대한 부하분배 기능의 지원 성능 역시 개선되었음을 의미한다고 판단할 수 있다.



(그림 4) 기존의 모델과 개선된 3-tier 모델의 평가
(Fig. 4) The evaluation between JDBC and improved 3-tier model

그러나 분산객체의 환경과 그렇지 아니한 경우의 비교는 이러한 단순비교로는 큰 의미를 나타내지 못한다. 즉, 이 모의실험은 수치적 표현을 위한 한가지 방법으로서는 의미를 가지지만 본 논문에서는 구현한 부분은 이러한 수치적인 비교평가 보다는 이 기종 분산객체 환경에서 자바언어의 특징인 플랫폼 독립적인 시스템 환경의 구현을 이루었고 'ORB-JDBC 하위규약'을 구현하여 자바에 의한 응용시스템 중에서 자바애플릿 만이 갖는 제한적인 보안구조를 극복하여 데이터베이스 서버에 대한 미들웨어의 종속성이라는 개념을 제거하였으며 또한 CORBA가 제공하는 데이터베이스 서버의 투명성을 확보하므로 기존의 JDBC 3-tier 모델의 장점을 유지하면서 문제점을 해결하였으며 CORBA가 제공하는 장점을 접목하였다는 점에 그 차별성을 가진다고 볼 수 있다.

5. 결론 및 향후 연구

본 논문에서는 이 기종 분산객체 환경에서 기존의 JDBC 3-tier 모델이 나타내고 있던 문제점들을 살펴보았고 이러한 문제점들을 CORBA를 이용하는 'ORB-JDBC 하위 규약'과 미들웨어를 설계 및 구현하여 개선하였다. 즉, 미들웨어와의 연결을 위한 'ORB-JDBC 하위 규약'은 JDBC의 하위 규약으로서 작동함으로써 별도의 표준이나 기존 응용 프로그램의 변경이 없이도 자바 응용 프로그램 상에서 적용이 가능해 졌으며 'ORB-JDBC 하위규약'의 구현으로 자바에 의한 응용시스템 중에서 자바애플릿의 데이타베이스 시스템에 대한 접근의 제한성을 극복시켰으며 동시에 데이터베이스 시스템과 연결하는 미들웨어는 ORB와 IIOP를 통해 서버의 위치 투명성을 확보하며 부하의 분배 기능도 있음을 알게 되었다. 즉, 이러한 구현 결과를 적용하는 비교 모의실험을 한 결과, 클라이언트의 수가 일정하게 증가하는 때에도 기존의 모델을 적용한 시스템은 급격한 성능 저하를 보이는 반면 개선된 모델을 이용한 시스템의 경우에는 비교적 완만한 감소를 보였다.

향후, 미들웨어의 성능 개선을 위해 클라이언트의 수의 변동에 유기적으로 변하는 미들웨어를 생성하고

동시에 미들웨어와 데이터베이스 시스템간의 연결을 관리하는 사용자 인터페이스를 가지는 미들웨어 관리자에 대한 연구가 진행되어야 할 것이다.

참 고 문 헌

[1] Visigenic Software Inc., *Distributed Object Computing in the Internet Age,* http://www.visigenic.com/prod/vbrok/wp.html, p11, 1997.

[2] Object Management Group, *CORBA 2.0 Specification*, http://www.omg.org, 1996.

[3] Thomas J. Mowbrayd., Ron Zahavi, *The Essential CORBA: System Integration Using Distributed Objects*, John Wiley & Sons Inc., 1995.

[4] Jon Siegel, *CORBA Fundamentals and Programming*, John Wiley & Sons Inc, 1996.

[5] Amjad Umar, *Object-Oriented Client/Server Internet Environments*, Prentice-Hall PTR, 1997.

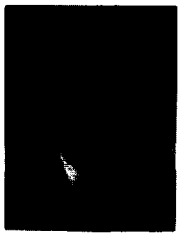
[6] Robert Orfali, Dan Harkey, *Client/Server Programming with JAVA and CORBA*, John Wiley & Sons, 1997.

[7] Sun Microsystems Inc., *The JDBC API version 1.20 Documentations, JDK 1.1.4 Documentations*, 1996.

[8] JavaSoft, *JDBC Guide: Getting Started*, JavaSoft, 1997.

[9] Visigenic Software Inc., *VisiBroker for Java 3.0: Programmer's Guide, VisiBroker for Java 3.0 Development Kit*, 1997

[10] 이진용, 전순미, "CORBA 분산 객체 환경을 위한 JDBC 하위 드라이버의 설계 및 구현", 한국정보과학회 가을 학술발표논문집 Vol.24, No.2, pp.201-204, 1997



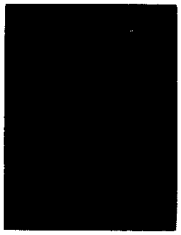
이 진 용

1996년 2월 인제대학교 전산학과(이학사)

1998년 2월 인제대학교 대학원 전산학과(이학석사)

현재~(주) 넥슨 재직

관심분야 : 분산객체시스템, 영상정보처리, OODBMS



전 순 미

1973년 2월 부산대학교 전자공학과(공학사)

1980년 2월 부산대학교 대학원 전자공학과(공학석사)

1985년 2월 경북대학교 대학원 전자공학과(공학박사)

1978년~1985년 부산전문대학 전자과 부교수

1986년~현재 인제대학교 정보컴퓨터학부 교수

관심분야 : 분산 객체 지향시스템, 시스템 모델링, 등