웹 응용을 위한 자바 질의 스텁의 구현 및 성능 평가

(Implementation and Performance Evaluation of Java Query Stub for WWW Applications)

최 원 익 † 김 형 주 †† 이 석 호 ††

(Wonik Choi) (Hyoung-joo Kim) (Sukho Lee)

요 약 WWW에 있어서 데이타베이스 통로는 필수적이다. 데이타베이스를 이용한 대부분의 웹 서비스는 CGI를 통해서 이루어지고 있는데 본 논문에서는 CGI가 갖는 문제점을 근본적으로 해결하기 위한 프레임웍으로서 JAQS(JAva Query Stub)를 제안하고 있다 JAQS를 통하여 이루어지는 서비스는 HTTP의 비연결성(connectionless) 및 무상태성(stateless)을 극복함으로써 상태 및 트랜잭션 관리를 지원할 수 있음은 물론 자바언어로 구현되어 플랫폼 독립적인 구조와 다중쓰레드를 최대한 활용하고 있다. 또한 JAQS와 함께 제안하고 있는 JAQS 관리자를 이용하면 데이타베이스가 분산되어 있는 환경에서도 효율적으로 동작할 수 있는 구조로 이루어져있다

Abstract Database gateway is absolutely vital to web-based services. Most web-based services with DBMS are implemented using CGI(Common Gateway Interface). This paper proposes a framework, dubbed JAQS(JAva Query Stub), as a solution for problems of CGI. JAQS not only supports state/transaction management by overcoming the connectionless and stateless nature of HTTP, but also effectively guarantees platform/DBMS-independence and multi-threading abilities because JAQS is implemented using Java. Also, using the JAQS Manager additionally proposed in this paper, JAQS can operate efficiently even in distributed database environments.

1. 서 론

1.1. 연구 배경

WWW(World-Wide Web, 이하 웹이라 함)[1]에 있어서 데이타베이스와의 연동은 필수적이다. 대부분의 대규모 웹 서비스들은 제공될 정보를 데이타베이스를 이용하여 관리할 필요성을 가지고 있다. 하지만, 웹이 분산 하이퍼미디어 방식의 정보 시스템으로서 많은 우수성을 갖는 반면, 아직까지 정보의 저장소로서 데이타베이스에 대한 연결 및 관리를 직접 지원하지 않고 있다. 따라서 클라이언트 측의 사용자들이 웹을 통해 질의를 요구하고 그 질의에 대한 결과를 얻기 위한 데이타베이스통로(database gateway)[2]는 웹과 데이타베이스를 통합하기 위한 핵심요소이다.

이러한 데이타베이스 통로의 구조는 데이타베이스를 접근하는 방식에 따라 서버쪽 확장과 클라이언트쪽 확장으로 나눌 수 있다[3]. 서버쪽 확장은 다시 CGI(Common Gateway Interface)[2] 실행화일방식, CGI 응용 서버 방식, 확장 API(Application Programmer's Interface)방식이 있으며, 클라이언트쪽 확장은 외부 뷰어를 이용하는 방식과 브라우져 확장 방식이 있다[3].

서버쪽 확장 방식은 모두 그 실행 코드가 플랫폼에 종속적인 것이며 아울러 대상으로 하고 있는 DBMS에 완전히 종속적인 것으로서 이식성이 매우 떨어진다. 또한, 서버쪽 확장은 웹 서버와 브라우져간의 프로토콜로서 HTTP (HyperText Transfer Protocol)[4]를 사용하게 되는데 이 HTTP는 비연결성(connectionless) 프로토콜이다. 따라서 클라이언트가 URL (Uniform

이 논문은 1997년 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음

† 학생회원 서울대학교 컴퓨터공학과
styxii@db.snu ac.kr
†† 종신회원 서울대학교 컴퓨터공학과 교수
hjk@oopsla snu.ac kr
shlee@comp.snu ac kr

논문접수 1998년 9월 7일
심사완료 · 1999년 5월 11일

Resource Locator)[5]을 통해 자료를 요청하고 웹서버는 요청 받은 자료를 전달하면 클라이언트와 서버와의 연결은 끊어지게 된다. 이러한 비연결성 때문에 과금관리, 사용시간관리, 상태 및 트랜잭션 관리 등을 필요로 하는 원격교육, 전자쇼핑, 실시간 통계 응용과 같은 서비스에 이러한 서버쪽 확장 방식의 데이타베이스 통로를 적용하려면 상태 및 트랜잭션 관리를 별도로 해야 하며[6] 이러한 방법은 근본적인 해결책을 제시하지 못하고 있다. 또한 서비스의 종류에 따라 동적 SQL문 처리가 반드시 있어야 되는 경우가 발생하는데 이때 SQLDA를 이용한 별도의 프로그래밍이 필요하게 된다.

클라이언트쪽 확장은 데이타베이스 통로가 클라이언트쪽에 구현되므로 클라이언트 성능에 따라 전체적인 시스템 성능이 좌우된다. 또한 클라이언트와 DBMS와의 통신비용이 서버쪽 확장에 비해 많고 수많은 이기종 클라이언트의 호환성문제로 서비스에 적용되기에는 현실적으로 어렵다

이에 본 논문에서는 이러한 문제점을 근본적으로 개선하기 위한 프레임웍으로서 JAQS(JAva Query Stub)를 제안하고 있으며 아울러 성능 비교를 통해 다른 데이타베이스 통로 구조 보다 성능이 우수함을 보이고 있다.

JAQS는 플랫폼과 DBMS에 독립적으로 동작하며 클라이언트와 서버가 항상 연결된 상태에서 동적인 질의문을 받고 그 결과를 돌려주면서 주어진 서비스를 행하기 때문에 HTTP의 비연결성을 극복할 수 있다. 또한 클라이언트와 서버간의 통신부하도 최소로 하고 있으며 자바(Java)[7][8][9] 언어를 이용하여 구현되어 플랫폼 독립적인 성질과 프로그램 동작에 있어서 병렬성을 최대한 보장하고 있다. 또한 본 논문에서는 JAQS와 함께 분산된 데이타베이스 환경에서도 이용할 수 있는 JAQS 관리자도 함께 제안하고 있다.

2. 관련 연구

2.1 웹 환경에서의 상태 및 트랜잭션 관리

HTTP는 웹 서버와 클라이언트가 하이퍼텍스트 문서를 송수신하기 위해 사용하는 프로토콜로서 비연결성(connectionless)과 무상태성(stateless)을 가장 큰 특징으로 들 수 있다. HTTP는 한번의 요청에 한번의 응답만을 하기 때문에 연결을 계속적으로 유지하지 않는다. 또한 HTTP는 연속적인 처리에 필요한 메모리를 별도로 보유하지도 않는다. 즉 현재의 상태정보를 연속적으로 넘겨주기 위해서는 모듈간의 정보전달 처리가 필요하다.

이러한 문제를 해결하기 위한 방법으로서 트랜잭션 상태를 서버쪽에서 유지하는 방법과 클라이언트쪽에서 유지하는 방법으로 크게 나눌 수 있다. 서버쪽 유지 방법은 상태유지를 서버가 관리하는 방법으로써 최초에 사용자의 웹 페이지 요구 시 상태식별자를 전송하고 다음 요구에는 그 상태 식별자를 통해서 트랜잭션을 결정하는 것이다. 이러한 일련의 상태 식별자는 서버의 메모리나 화일 또는 데이타베이스에 선택적으로 유지할 수 있다. 그러나, 서버쪽에 관리 부담을 안겨주고, 트랜잭션의 완료 여부를 알 수 없는 문제점은 해결하지 못하고 있다. 상태 식별자로는 사용자의 IP주소나 쿠키(Cookie)[6]를 이용하고 있다.

클라이언트쪽 유지 방법은 웹 페이지간에 상태유지를 위한 상태 자료를 다음 요구에 결부시켜 트랜잭션 과정을 이어가는 방법이다. 그러나 이 방법은 상태자료들로 인한 통신비용이 증가하고 상태자료가 URL의 일부에 나타나므로 노출의 위험이 크다. 또한 상태 자료를 쿠키를 이용하여 저장할 수 있지만 한정된 웹 브라우져에 대해서만 적용가능하고 쿠키의 설정이 사용자의 결정에 의해 거부될 경우에는 무용지물이 되는 문제점이 있다.

2.2 데이타베이스 통로의 구조

데이타베이스 통로의 구조는 앞서 언급한 바와 같이 데이타베이스에 대한 접근방식에 따라 서버쪽 확장과 클라이언트쪽 확장으로 나눌 수 있다.

서버쪽 확장 방식에 속하는 CGI 실행화일 방식은 클라이언트의 각 요구마다 그에 대한 데이타베이스 통로 프로세스가 매번 생성되고 그 때마다 DBMS접속이 일어나는 구조이다. 이 데이타베이스 통로 프로세스는 Embedded SQL library도 포함하게 되므로 프로세스의 크기, 생성 및 종료, 프로세스간의 문맥교환 등의 프로세스의 관리가 서버에게 큰 부담이 된다. 따라서 시스템 자원 부족 현상이 빨리 나타나게 되고 그에 따라 성능도 급격히 저하된다. 초기 대부분의 서비스는 이러한 CGI 실행화일 방식을 사용하여 구축되었으며 그 예로는 Cold Fusion[10], Web/Genera[11], GSQL[12], WDB[13] 등을 들 수 있다[3].

반면 CGI 응용 서버 방식에서는 데이타베이스 통로가 디몬(daemon) 형태로 DBMS와 연결을 유지하고 있는 상태로 동작하고 클라이언트의 각 요구에 대한 CGI 프로세스(dispatcher)는 단지 요구 사항을 전달하는 일을 수행하므로 프로세스의 크기가 매우 작고 DBMS에 대한 접속이 일어나지 않는다. 그러므로 동시에 많은 프로세스가 생성되어도 시스템 자원 부족 문제가 쉽게 발생되지 않는다. 이러한 CGI 응용 서버 방식 시스템의 예로는 O2Web[14], MARS[15] 등을 들 수 있다[3].

확장 API방식은 웹 서버 프로세스와 웹 서버에서 제공되는 API로 작성된 응용 프로그램이 동적으로 링크 되어 하나의 프로세스로 실행되기 때문에 프로세스 관리비용을 줄일 수 있지만 구현 자체가 서버 API에 완전히 종속되며 현재 이러한 서버 API에 대한 표준화도 이루어지지 않고 있는 것이 단점이라 할 수 있다.

CGI 실행화일 방식은 프로세스 관리를 위한 부하로 인해 대규모 웹 서비스에서는 사용되기에 어려우며, 가장 좋은 성능을 보이는 확장 API방식은 데이타베이스 통로가 웹 서버의 확장 API로 구성되어 그 구현이 웹 서버에 종속되는 것이 문제점이라 할 수 있으며 아울러 앞서 설명한 두 가지 방식 모두 플랫폼과 DBMS에 종속적인 실행코드이기 때문에 이식성이 매우 떨어진다.

본 논문에서는 이러한 문제를 해결하기 위해 JAQS를 제안하였다. 구조면에서 이 JAQS는 CGI 응용서버 방식과 흡사하지만 다음과 같은 점에서 다르다.

(1) JAQS는 플랫폼과 DBMS에 독립적인 실행코드로 구성되어있다.

(2) JAQS에서의 dispatcher는 프로세스가 아닌 쓰레드로 동작한다.

(3) 또한 CGI 응용서버는 특정 DBMS에 종속적인 라이브러리로 구현되어 이기종 DBMS에 적용하기 위해서는 개발 시간과 노력이 중복되지만 JAQS는 Java언어와 JDBC[16][17] 드라이버를 이용하여 구현되었기 때문에 쉽게 이식할 수 있으며 플랫폼 독립적인 특성을 갖게 된다.

(4) 아울러 JAQS는 HTTP의 비연결성(connectionless), 무상태성(stateless)을 근본적으로 극복하여 상태 및 트랜잭션 관리를 필요로 하는 웹 서비스를 지원하고 있다.

(5) CGI 응용서버 방식 보다 월등한 성능을 보여준다. 5장에서 구체적인 성능 비교를 하고 있다.

(6) 동적(dynamic) SQL 프로그램이 요구하는 복잡한 과정이 필요 없는 사용이 편리한 클라이언트 API가 제공되므로 클라이언트는 쉽게 JAQS와 연결을 하여 DBMS를 기반으로 하는 서비스를 제공할 수 있다.

반면에 클라이언트쪽 확장 방식 중 외부 뷰어를 이용한 확장은 웹 브라우져가 필요시 외부 뷰어를 실행시키고 이 외부 뷰어가 서버쪽의 DBMS와 연결하여 서비스가 이루어지는 방식이다. 브라우져 확장 방식은 HTML 내에 포함된 스크립트 언어 또는 JDBC를 포함한 자바 applet 또는 ActiveX control[18]등을 통해 DBMS에 접근할 수 있는 기능을 제공한다. 스크립트 언어로서는 JavaScript[19], VBScript[20] 등이 있으며 DBMS접근 기능으로는 JDBC와 ODBC[21]를 들 수 있다. JDBC에 대해서는 2.3절에서 다시 설명하고 있다.

이러한 클라이언트쪽 확장은 데이타베이스 통로가 클라이언트쪽에 구현되므로 클라이언트의 성능에 따라 전체적인 시스템 성능이 좌우되며 수많은 클라이언트와 DBMS와의 통신비용이 서버쪽 확장에 비해 많은 단점이 있다.

2.3 JDBC

앞서 언급한 바와 같은 데이타베이스 통로의 구조 외에 웹을 통해 DBMS에 접근할 수 있는 또 다른 구조중의 하나는 JDBC를 이용하여 목표 DBMS에 연결하여 원하는 서비스를 하는 방법이다. JDBC는 JavaSoft사에서 정의하고 있는 Java API로서 목표 DBMS에서 연결을 설정하고 SQL문을 실행시키기 위한 API이다[16]. JDBC는 Java언어로 구성된 클래스와 인터페이스의 집합이며 Java언어를 사용하여 데이타베이스 응용프로그램을 작성하기 위한 표준 도구들을 개발자에게 제공하고 있다. JDBC드라이버에는 JDBC-ODBC bridge driver, Native-API partly-Java driver, JDBC-Net pure Java driver, Native-protocol pure Java driver와 같이 4가지 부류로 나눌 수 있다[17].

JDBC-ODBC bridge driver는 ODBC 드라이버를 통해 JDBC기능을 제공한다. ODBC드라이버를 통한다는 것은 모든 클라이언트에 ODBC드라이버가 설치되어 있어야 한다는 의미이며 이 드라이버는 win32이진코드로만 지원되기 때문에 플랫폼 독립적인 특성을 가질 수 없다. Native-API partly-Java driver는 JDBC호출을 특정 DBMS의 클라이언트 API로 변환시킨다. 이 역시 JDBC-ODBC bridge driver와 같이 각 클라이언트에 DBMS에 종속적인 클라이언트 API의 이진 코드가 설치되어 있어야 하지만 특정 DBMS의 클라이언트 API로 직접 변환되어 실행되기 때문에 성능상의 이점이 있다.

JDBC-Net pure Java driver는 JDBC호출을 특정 DBMS API 호출로 변환하는데 DBMS에 독립적인 네트워크 프로토콜을 사용한다. 이 프로토콜은 DBMS에 전달되어 특정 DBMS프로토콜로 변역 되어 실행하게 된다. Native-protocol pure Java driver는 JDBC호출을 특정 DBMS의 프로토콜로 직접 변환한다.

JDBC를 이용하여 DBMS에 접근하는 구조에는 크게 2-계층 모델과 3-계층 모델이 있다[17][22][23][24]. 2-계층 모델은 JDBC를 이용하여 작성한 applet 또는 application이 직접 목표 DBMS에 연결되어 서비스하는 구조이다. 결과적으로 클라이언트가 연결을 요구할 때마다 DBMS에 새로운 연결이 생성되는 구조로서, 클라이언트 수가 많은 대규모 서비스인 경우 CGI실행화일 방식과

클라이언트 확장 방식이 가지고 있는 단점이 그대로 나타나게 된다. 즉, JDBC의 직접적인 이용은 과다한 DBMS의 연결 설정으로 인한 서버의 급속한 자원 고갈로 인한 문제와 클라이언트쪽 코드의 크기와 부담이 커지는 문제점을 갖게 된다. 이러한 문제점을 해결하기 위해서는 JAQS와 같은 중간계층이 필요하다.

3-계층 모델은 목표 DBMS에 연결을 항상 설정하고 있는 중간계층이 클라이언트와 DBMS사이에 존재한다 [23][24]. 클라이언트는 원하는 SQL문을 중간계층에게 전달하고 결과를 중간계층으로부터 돌려 받게 된다. JAQS는 이러한 3-계층 모델에서 중간계층을 구성하고 있다.

3. 설 계

이 장에서는 JAQS의 접근 방법을 통해 JAQS의 설계 요구 사항을 분석하고 JAQS의 구성요소인 SQL 관리자(SQL Manager), 질의스트림 관리자(QueryStream Manager), 쓰레드 관리자(Thread Manager)의 기능 및 동작 과정을 설명하고 있다.

3.1 JAQS의 구조

JAQS는 초기 기동시 목표 DBMS에 연결한 후 클라이언트의 접속을 기다린다. 클라이언트의 접속 요구 시 그 사용자에 대해 쓰레드를 생성하여 사용자와 사용 시간을 관리하면서 질의문을 처리하여 그 결과를 돌려준다. 클라이언트와 JAQS는 사용자의 로그인부터 로그아웃까지 항상 연결되어 있으면서 수백 바이트 이내에 불과한 질의와 그 결과만을 주고받으면서 최소한의 통신 부하로 클라이언트가 요구하는 서비스를 행하게 된다. JAQS는 하나의 프로세스가 디폰 방식으로 동작하고 그 프로세스 안에서 쓰레드를 최대한 활용하여 프로세스 생성 및 종료, 교체에 대한 비용을 최소화하고 있다.

JAQS의 클라이언트는 자바 applet 또는 자바 application일 수도 있고 C/C++로 작성된 응용프로그램 뿐만 아니라 TCP/IP 소켓을 지원하는 다른 프로그램언어로 작성된 응용프로그램일 수도 있으며 이는 적용되는 서비스에 대해 선택적으로 구현할 수 있다. 클라이언트는 JAQS와 소켓으로 연결한 후 이 소켓을 통해 질의를 전달하고 그 결과를 소켓을 통해 전달받는 것이 구현의 핵심이 된다. 그 결과를 어떻게 클라이언트 측에서 실현하는가는 부차적인 내용이다.

JAQS는 질의스트림 관리자(QueryStream Manager)와 쓰레드 관리자 (Thread Manager), 그리고 SQL 관리자(SQL manager)로 구성된다<그림 1> JAQS는 클라이언트와 서버사이의 중간 계층을 구성하고 있다[22][24].

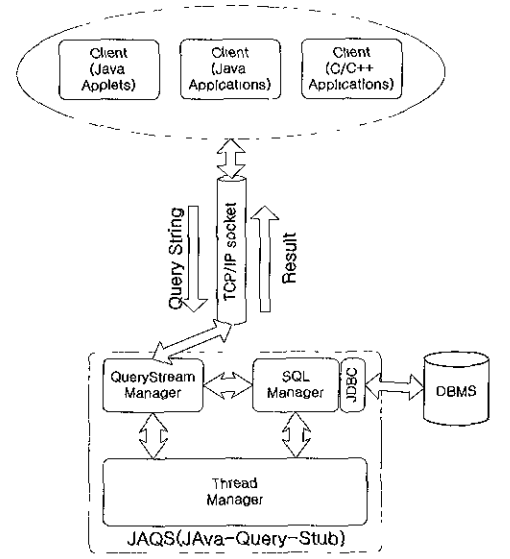

그림 1 JAQS의 구조

3.1.1 SQL 관리자

목표 DBMS와 연결을 한 상태에서 질의스트림 관리자로부터 넘겨받은 질의문을 처리한 후 결과 집합(result set)을 질의스트림 관리자에게 넘겨준다. 이 때 목표 DBMS는 JDBC 드라이버를 제공하는 DBMS이면 어느 것에도 연결이 가능하다.

질의스트림 관리자로부터 넘어온 질의문은 DML 또는 DDL일 수 있다[25]. DML중 SELECT문일 경우 질의 결과는 0개이상의 행이며 UPDATE, DELETE, INSERT문의 경우 질의가 반영된 행의 개수가 그 결과가 된다. DDL의 경우는 질의 실행의 성공여부가 그 결과가 된다. SQL 관리자는 DML과 DDL 모두에 대해 자동적으로 해당되는 형식에 따라 결과를 문자열로 구성하여 이를 질의스트림 관리자에게 전달한다. 그 형식은 <표 1>과 같다.

표 1 결과 형식

DML	SELECT	(i번째 열,속성이름,문자열로 변환된 속성값)*(EOF)
	UPDATE DELETE INSERT	반영된 열의 개수(EOF)
DDL		(0|1)[메세지](EOF)

3.1.2 질의스트림 관리자

질의스트림 관리자는 클라이언트 측에서 넘어오는 질의문을 SQL 관리자에 넘겨주고 SQL 관리자로부터 넘어오는 결과를 소켓을 통하여 클라이언트 측에 전달해 준다. 이 질의스트림 관리자는 쓰레드 관리자에 의해 쓰레드로 생성되어 사용자마다 하나씩 할당되며 소켓으로 클라이언트 측과의 연결을 유지하면서 클라이언트 측으로부터 질의문을 받고 결과 집합을 클라이언트 측으로 되돌려주는 기능을 전담하게 된다.

질의스트림 관리자는 클라이언트 측에서 SQL문을 전달받으면 SQL 관리자의 메소드를 호출하게 되고 결과 집합을 돌려 받게 된다. 돌려 받은 결과 집합은 소켓을 통해 클라이언트 측에 전달된다.

3.1.3 쓰레드 관리자

JAQS에서 가장 핵심적인 역할을 하는 부분으로서 클라이언트의 접속 요구 시 질의스트림 관리자의 쓰레드를 생성시켜 접속을 요구한 사용자에게 할당한다. 사용자의 이름은 할당되는 쓰레드의 이름으로 부여되어 사용자 관리에 이용될 수 있다. 이 쓰레드는 생성시간과 종료시간을 가지게 되어 쓰레드의 연령을 알 수 있고 이는 곧 사용자의 사용 시간이 된다. 이 사용시간은 서비스의 성질에 따라 데이타베이스에 저장되거나 단순히 화일로 저장될 수 있다.

3.1.4 클라이언트

클라이언트는 JAQS와의 연결을 위한 네트워크 함수들과 소켓으로 질의문을 보내고 결과 집합을 받아 파싱하는 함수들이 기본 골격이 된다. 결과를 해석하는 것과 문자열로 넘어온 결과에 알맞은 형변환을 하여 응용에 맞게 사용하는 것은 클라이언트가 해야 될 일이다. 예를 들어 aggregate 함수 COUNT()를 이용하여 테이블의 열의 수를 요구하고 그 값을 파싱하여 얻은 정수로 형변환을 통해 그래프로 나타내든지 문자열을 그대로 표를 통해 나타내든지 하는 것은 클라이언트 측이 할 일이다

클라이언트는 Java applet 또는 application으로 구현될 수 있을 뿐만 아니라 소켓 라이브러리가 제공되는 프로그래밍 언어이면 어떠한 프로그래밍 언어로도 구현할 수 있다.

3.2 JAQS의 동작

JAQS는 초기 기동시 DBMS와 연결을 하고 질의를 처리할 수 있도록 SQL 관리자를 생성하고 클라이언트의 연결 요구를 기다린다. 사용자가 JAQS에 연결요구를 하면[<그림 2>의 (1)] JAQS는 사용자의 인증절차를 거쳐 쓰레드 관리자에게 질의스트림 관리자의 쓰레드의 생성[<그림 2>의 (2)]을 지시하고 그 사용자 요구에 대한 처리를 전담하게 한다. 그러나 불법적인 사용자에 대해서는 접속을 거부한다. 쓰레드 생성을 성공적으로 마친 쓰레드 관리자는 또다시 JAQS의 지시를 기다린다. 쓰레드 생성에 실패하였다면 정해진 오류메세지를 내보낸다. 성공적으로 접속을 마친 사용자가 질의를 보내면 이 질의는 질의스트림 관리자를 경유하여 SQL 관리자에게 보내지고 이 질의에 의해 DBMS가 그 결과 집합을 만들어 낸다. 만들어진 결과 집합은 SQL 관리자에 의해 질의문 형식에 맞는 결과 집합으로 변형되고 이는 질의스트림 관리자에 의해 클라이언트 측에 전달된다[<그림 2>의 (3)].

사용자가 로그아웃을 하면 그 사용자에게 할당되었던 쓰레드가 활동한 시간이 데이타베이스 또는 화일에 선택적으로 저장된 후 JAQS와 연결이 해제되고 그 쓰레드는 종료된다[<그림 2>의 (4)].

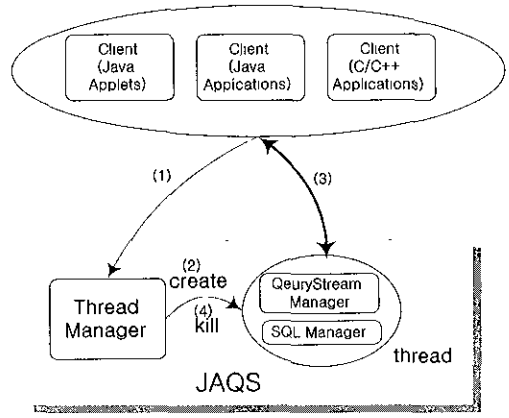

그림 2 JAQS의 동작

4. 구 현

이 장에서는 JAQS의 각 구성 요소에 대해 설계 명세를 제시하고 JAQS를 이용하는데 필요한 클라이언트 API에 대해서 설명한다.

4.1 SQL 관리자

SQL 관리자(SQL Manager)가 하는 일은 명확하다. SQL 관리자는 JAQS의 본체를 구성하고 있으며 생성시 목표 DBMS와의 연결을 설정하고 있다. 연결 설정 후에는 JAQS의 인스턴스를 생성하고 쓰레드 관리자를 기동시킨다. 질의스트림 관리자에 의해 호출되어 결과 집합을 생성하는 메소드는 동시에 다수의 질의스트림 관리자에 의해 호출되므로 이 때 결과 집합이 파괴되는 것을 방지하기 위해 동기화(synchronized) 메소드로 선

언되어 있다. 질의 스트림으로부터 넘겨받은 SQL문은 DBMS로 보내지고 그 결과를 가공하여 다시 질의스트림 관리자에게 돌려주게 된다.

4.2 질의스트림 관리자

질의스트림 관리자(QueryStream Manager)는 쓰레드 관리자에 의해 클라이언트 접속시 생성되며 입/출력을 위한 소켓을 할당한다. 클라이언트로부터 SQL문이 입력되면 결과 집합을 생성하기 위해 SQL 관리자의 메소드를 호출하고 돌려 받은 결과 집합을 소켓을 통해 클라이언트로 전달한다. 클라이언트로부터 접속해제 요구가 오면 활동을 중단한다.

4.3 쓰레드 관리자

JAQS의 인스턴스 기동시 쓰레드 관리자(Thread Manager)가 생성된다. 쓰레드 관리자는 생성시 서버소켓을 할당하여 클라이언트의 접속 요구를 기다리게 된다. 클라이언트의 접속 요구가 들어오면 자신을 복제하여 질의 스트림 관리자를 생성하고 자신은 다시 클라이언트의 접속을 기다린다. 복제된 질의 스트림 관리자는 앞에서 설명한 바와 같이 클라이언트와는 소켓으로 연결하여 SQL문을 받고 그 결과를 돌려주게 된다. 또한 쓰레드 관리자는 자신을 복제하여 질의스트림 관리자를 생성할 때 사용자 관리와 사용시간 관리를 위한 정보를 유지하게 된다. 이 정보는 화일 시스템 또는 데이타베이스에 선택적으로 저장될 수 있다.

JAQS의 기동 후 전체적인 JAQS의 동작 모습은 <그림 3>과 같다.

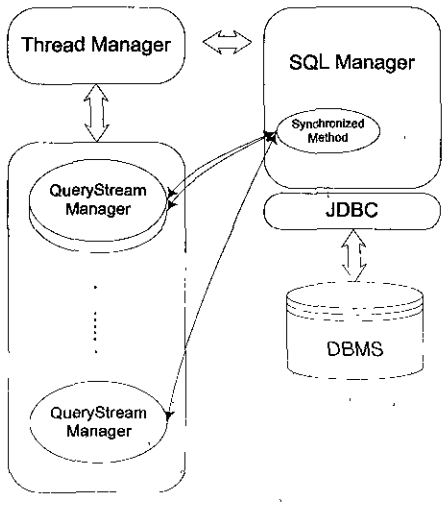

그림 3 JAQS의 기동 후 모습

4.4 클라이언트 API

클라이언트는 JAQS와의 연결, 질의, 결과파싱, 연결해제 등의 기능을 하는 API를 제공받는다. JAQS를 통해 서비스 하는 클라이언트는 제공받은 간단한 API를 통해 DBMS를 이용한 대규모 웹 서비스를 할 수 있기 때문에 JDBC 또는 DBMS종속적인 드라이버 등을 필요로 하지 않는다. 이러한 API는 JDBC와는 달리 Java 뿐만이 아닌 C, C++ 등의 여러 프로그래밍 언어로 구현되어 제공될 수 있다. 또한 이러한 API들은 JDBC API보다 가벼우므로 초기 통신 부하도 적으며 JDBC를 이용하여 구현하는 것 보다 쉬운 형태로 제공된다.

클라이언트는 JAQS와의 연결설정 함수를 호출하여 JAQS와 연결한다. 연결이 설정되면 원하는 질의문을 JAQS에게 전달한 후 그 결과를 string형태로 돌려 받게 된다. 클라이언트는 보낸 질의문에 따라 그에 맞는 결과 집합(result set) 파싱 함수를 호출한다. SELECT문인 경우 애트리뷰트 이름을 인자로 넘겨주면 그 값이 string형태로 변환되며 복수개의 튜플이 선택된 경우 index를 통하여 행(row)을 선택할 수 있다. UPDATE, INSERT, DELETE문인 경우 반영된 행의 개수가 역시 string형태로 반환되며 DDL문인 경우 DBMS가 그 DDL문을 실행하고 내보내는 메세지를 string형태로 반환 받는다.

클라이언트는 위와 같은 간단한 API를 통해서 JAQS와 연결하고 DBMS에 접근하므로 써 DBMS를 기반으로 하는 웹 서비스를 할 수 있다.

5. JAQS의 성능

이 장에서는 JAQS의 성능을 평가하기 위한 성능 평가 구조 및 환경을 제시하고 먼저 JAQS의 성능을 평가한 후 CGI 실행화일 방식과 CGI 응용서버 방식의 성능과 비교한다.

5.1 성능 평가 방법 및 환경

JAQS의 성능을 평가하기 위한 성능 평가 구조는 <그림 4>와 같다. 이 도구는 Java언어로 자체 구현하였다.WebSTONE[26]과 Webpest[27]와 같은 대표적인 웹 서버 성능 평가 도구들을 사용하지 않은 이유는 이러한 성능 평가 도구들이 JAQS의 성능을 측정하기에 적합하지 않기 때문이다.

자체 구현된 성능 평가 도구는 큐잉 네트워크 모델을 기반으로 하였다. 각 요청들은 독립적으로 요구된다고 가정하고 있으며 이들 요구는 지수 분포로 모델링되었다. 평가 환경은 하나의 서버 시스템과 하나의 클라이언트 시스템으로 이루어지며 기동시 요구들의 평균값인

평균요청시간간격을 명세한 화일을 읽어들여 요청한 수만큼의 쓰레드를 지수 분포로 생성하고 그 쓰레드는 독립적으로 요청을 보내고 평균응답시간을 구하여 결과 화일을 작성하여 준다. 여기서 평균응답시간이란 각각의 요청이 있은 후부터 그 결과의 전체가 클라이언트 시스템으로 도착할 때까지 경과된 시간의 평균을 구한 것이다.

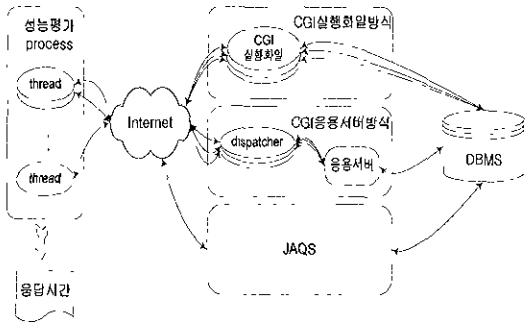

그림 4 성능 평가 구조

본 성능평가에서는 100개의 요청을 8000ms, 5000ms, 2000ms, 1000ms, 500ms, 100ms, 10ms, 1ms의 평균요청시간간격으로 보내고 그 평균응답시간을 구하였다. 평균요청시간간격이 작을 수록 서버의 부하는 증가하게 되고 그에 따라 평균응답시간은 길어질 것을 예상할 수 있다. 먼저 JAQS의 성능을 평가하고 CGI 실행화일 방식 및 CGI 응용서버 방식과 성능을 비교한다.

JAQS의 성능평가를 위한 테이블 스키마는 <표 2>와 같다. 테이블 test는 100,000개의 튜플로 구성되어 있으며 id는 1부터 순차적인 정수 값을 갖는다. name, address, phone, email은 임의의 문자열을 갖는다. age는 난수발생기를 이용하여 1부터 100이하의 정수 값을 갖는다.

성능 평가에 사용된 질의문은 <표 3>과 같다. Q1은 WHERE절을 통하여 유일한 튜플을 반환하는 SELECT문으로서 가장 단순한 형태의 질의문이다. Q2는 DBMS의 aggregate function을 사용한 질의문으로서 두개의 숫자가 반환되는 질의문이다. Q3은 유일한 튜플이 update되는 UPDATE문을 선택하였다. Q4는 25개의 튜플을 반환하는 SELECT문을 선택하였다. 성능평가에 쓰인 서버로는 Axil320(SuperSparc 75Mhz 1CPU), RAM32M를 사용하였고 클라이언트는 Pentium 100Mhz, RAM 32M를 사용하였고 서버와 클라이언트는 이더넷으로 연결하였다. DBMS는 Oracle 7.2를 사용하였으며 JDBC는 Oracle JDBC driver Ver 7.3.4.0.1을 사용하였다.

표 2 성능평가를 위한 테이블 스키마

```
create table test (
        id        number(6.0) primary key,
        name      varchar(20),
        address   varchar(200),
        phone     char(14),
        email     varchar(100),
        age       number(3.0)
);
```

표 3 성능평가에 사용된 질의문

Q1	select id, name, address, email from test where id=random() % 100000;
Q2	select count(*), avg(age) from test where age > random() % 100;
Q3	update test set age = id where id=random() % 100000;
Q4	x := random() % 100000; select id, name, address, email from test where id>x and id<x+25;

5.2 JAQS의 성능 평가 결과

각 질의문에 대한 성능 평가 결과는 <표 4>와 같다. 여기서 결과 집합의 크기는 각 요구에 대한 결과 집합들의 평균 크기이며 질의문의 복잡도는 그 질의문을 수행하는 데 필요한 DBMS자체 연산의 양이며 일반적으로 수행시간에 비례한다

표 4 Q1, Q2, Q3, Q4에 대한 JAQS의 성능측정결과

질의문	결과 집합의 크기 (bytes)	평균응답시간 (ms)	질의문의 복잡도	연산종류
Q1	220	225	간단	읽기
Q2	232	12,847	복잡	읽기
Q3	19	206	간단	쓰기
Q4	6,000	1,359	간단	읽기

Q1의 결과 집합의 크기가 평균 220바이트, 그리고 Q3은 평균 19바이트였고 <표 4>와 <그림 5>에서와 같이 197ms에서 255ms사이의 안정된 평균응답시간을 보이고 있다. Q2의 경우 평균요청시간간격이 5000ms이상일 때 안정된 성능을 보였는데 이는 본 성능 평가에 사용된 서버의 DBMS가 Q2를 수행하기 위해 3000ms에서 4000ms사이의 시간이 소요되므로 요청시간 간격이 5000ms이상일 때는 JAQS에서의 요구대기가 필요

없게 되기 때문이다. Q2와 같이 복잡한 질의문의 평균 응답시간은 서버의 성능에 따라 크게 좌우되고 있다.

반면, Q4는 DBMS의 연산에 대한 시간보다는 결과 집합의 크기 때문에 평균응답시간이 많이 소요되는 경우이다. Q4의 결과 집합의 크기는 테이블 test의 25개 튜플로서 약 6,000바이트 정도로 다른 질의문의 결과 집합보다 상대적으로 크다.

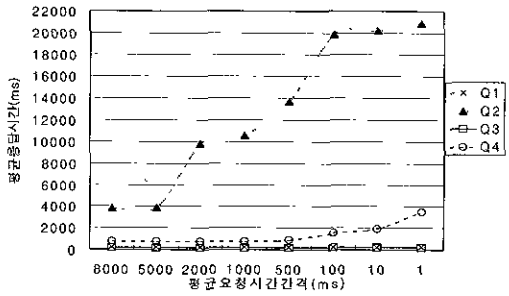

그림 5 Q1, Q2, Q3, Q4에 대한 응답곡선

JAQS의 성능은 비교적 간단한 질의문의 경우 300ms 이내의 응답시간을 보였고 질의문의 복잡도가 높을 때에는 서버의 성능에 의존하며 질의문의 결과 집합의 크기가 클 때에는 네트워크 부하로 인한 성능 저하가 발생한다.

5.3 다른 데이타베이스 통로 구조의 성능과의 비교

비교 대상은 CGI 실행화일 방식과 CGI 응용서버 방식이다. Q1의 경우 CGI 실행화일 방식은 500ms이하의 평균요청시간 간격부터 급격히 저하된 성능을 보여주고 있으며 CGI 응용서버 방식 역시 성능이 저하되고 있는 반면 JAQS는 300ms이내의 좋은 평균응답시간을 보여주고 있다. 세 가지 방식 모두 요청에 대한 실패는 보이지 않았다. Q1과 같이 단순한 질의문에 대해서도 저조한 성능을 보이는 것을 알 수 있다<그림 6>.

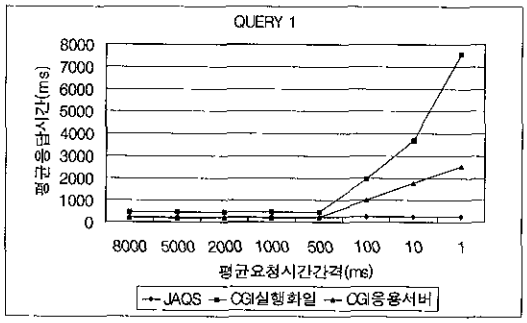

그림 6 Q1에 대한 성능 비교

<그림 7>은 Q2에 대한 성능 곡선이다. 100개의 요구에 대해 50%인 50개의 요구가 실패로 처리되면 그 이하의 평균요청시간간격에 대한 평가는 하지 않았다. CGI 실행화일 방식은 5000ms의 평균 요청시간 간격부터 거의 모든 요청에 대해 실패하고 있다. 이보다 조금 나은 성능을 보이고 있는 CGI 응용서버 방식은 2000ms의 평균요청시간간격부터 급격한 성능 저하를 보여주고 있다. JAQS는 20000ms내외에서 실패 없이 요청에 대한 서비스를 하고 있음을 보여주고 있다.

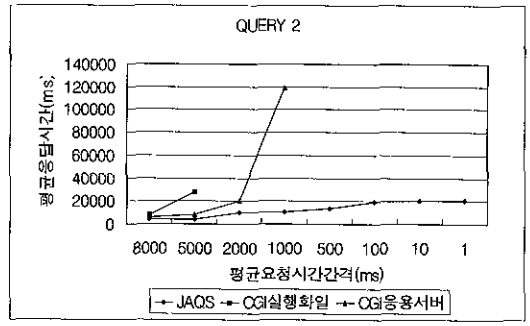

그림 7 Q2에 대한 성능 비교

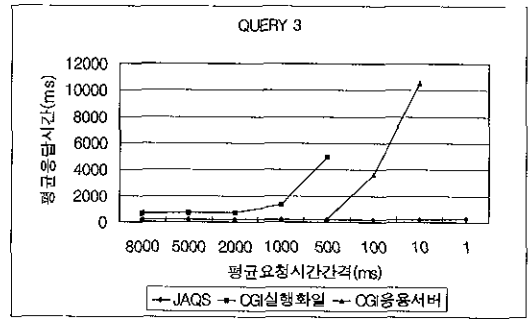

그림 8 Q3에 대한 성능 비교

<그림 8>은 Q3에 대한 성능 곡선으로서 CGI 실행화일 방식은 500ms의 평균요청시간간격 이하부터 그리고 CGI 응용서버 방식은 10ms의 평균요청시간간격 이하부터 요청에 대한 서비스를 못하고 있다. JAQS는 300ms 내외에서 실패 없이 요청에 대한 서비스를 하고 있음을 보여주고 있다.

<그림 9>는 Q4에 대한 성능 곡선으로서 CGI 실행화일 방식은 500ms의 평균요청시간간격부터 급격히 성능이 저조해지고 100ms 평균요청시간간격 이하부터는 서비스를 못하고 있다. 그리고 CGI 응용서버 방식은

6000ms내외의 평균응답시간으로 실패 없이 서비스를 하고 있다. 그러나 JAQS는 4000ms내외의 평균응답시간으로 실패 없이 요청에 대한 서비스를 하고 있음을 알 수 있다.

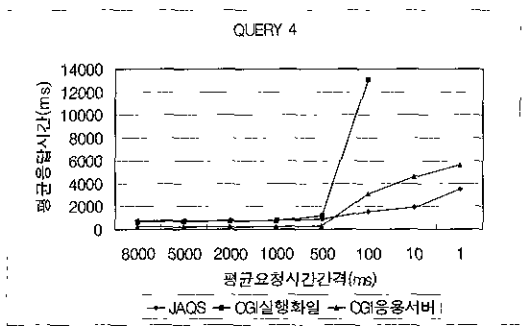

그림 9 Q4에 대한 성능 비교

이상에서 본 바와 같이 CGI 실행화일 방식은 요청에 대한 프로세스의 개수 및 크기 그리고 그에 따른 과도한 프로세스 관리비용으로 인하여 시스템 자원이 급속히 부족하게 되고 그로 인하여 질의문에 관계없이 가장 저조한 성능을 보여주고 있으며 CGI 응용서버 방식은 CGI 실행화일 방식보다는 나은 성능을 보이고 있지만 그 역시 JAQS의 성능에는 크게 못 미치고 있다.

JAQS는 Q1, Q2, Q3, Q4의 네 가지 질의문과 같이 여러 종류의 질의문에 대해서 실패 없이 서비스를 하고 있는 사실은 주목할 만한 것이며 이는 JAQS를 이용한 서비스의 신뢰성을 보장하게 된다. 또한 네 가지 질의문의 모두에 대해 가장 짧은 평균응답시간 내에 서비스를 하고 있으며 이는 JAQS가 쓰레드를 이용하여 동작에 있어 병렬성을 최대한 보장하고 서버의 자원을 가장 효율적으로 사용하고 있기 때문이다. 이러한 JAQS의 성능은 대규모 웹 서비스를 할 수 있는 가능성을 보여주고 있다.

6. 분산 데이타베이스 환경에서의 JAQS

JAQS는 앞서 설명한 바와 같이 DBMS와 플랫폼에 독립적인 특성을 갖고 있다. 이러한 JAQS의 특성을 분산 환경으로 확장시키면 클라이언트는 손쉽게 지리적으로 분산되어있는 데이타베이스에 접근할 수 있다. <그림 10>과 같이 지리적으로 분산된 데이타베이스에 각각 연결되어 산재하는 JAQS는 JAQS 관리자(JAQS Manager)와 연결될 수 있다.

JAQS 관리자를 통해 연결된 사용자들에게는 분산된 데이타베이스를 선택적으로 접근할 수 있게 한다. 이 때 자바 applet인 경우 그 applet을 가지고 있던 서버의 데이타베이스에만 접근 가능하므로[16][17] 이러한 경우 JAQS 관리자의 역할은 필수적이다. 신뢰(trusted) applet인 경우 application처럼 동작하여 다른 서버의 데이타베이스에도 접근할 수 있지만[16][17] 이 또한 각각의 데이타베이스에 연결이 새로 생성되어 접근 비용이 증가되므로 JAQS 관리자와 같은 중간계층이 필요하게 된다.

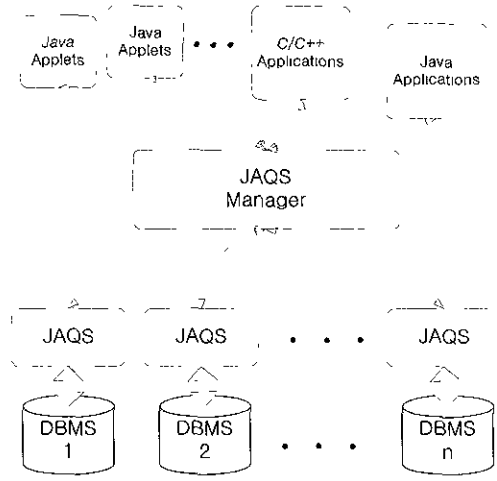

그림 10 분산 데이타베이스 환경에서의 JAQS

JAQS 관리자는 초기 기동시 산재되어 있는 JAQS의 정보에 따라 각각의 JAQS와 연결을 하고 클라이언트 측의 선택에 따라 JAQS와의 연결을 중재해주는 JAQS 선택스위치 역할을 하여 분산된 이기종 DBMS에 대한 연결을 대행하여 준다. 따라서 클라이언트는 여러 이기종 DBMS에 라이브러리 또는 드라이버들에 의존하지 않고 JAQS 관리자를 통해 DBMS에 접근할 수 있다.

JAQS 관리자를 통하여 JAQS 관리자에 연결된 사용자에게는 현재 동작중인 JAQS에 대한 정보가 제공되고 사용자는 그 정보에 따라 질의를 선택적으로 JAQS에게 보낼 수 있고 그 결과를 얻어내어 서비스에 이용할 수 있다. JAQS 관리자는 초기 기동시 JAQS환경에 대한 명세화일을 읽어들여 각 JAQS와 연결을 설정한다. 연결 설정에 성공하여 현재 활동중인 JAQS에 대한 정보는 클라이언트의 접속 시 제공된다. 클라이언트는 제공된 JAQS에 대한 정보를 통하여 원하는 JAQS를 선택하여 SQL문을 던지고 그 결과를 돌려 받는다.

7. 결 론

본 논문에서 제안한 JAQS는 웹과 DBMS연동에 있어서 플랫폼과 DBMS에 독립성을 제공하고 있으며 HTTP의 비연결성(connectionless)과 무상태성(stateless)을 근본적으로 극복하여 기존의 데이타베이스 통로 구조로는 지원하기 어려운 사용자관리, 과금관리와 같이 상태 및 트랜잭션 관리를 필요로 하는 웹 서비스를 지원할 수 있도록 하고 있다.

JAQS는 클라이언트가 서버 측의 DBMS에 접근해야 하는 경우 DBMS에 대한 연결을 네트워크에 대한 연결로 사상하여 가장 효율적인 접근 구조를 제공하고 있다. 클라이언트는 JAQS와 연결을 통하여 DBMS에 대한 접근 작업 없이 DBMS로부터 손쉽게 동적SQL문의 결과 집합을 얻어낼 수 있다. 웹 브라우저상의 Java applet이나 ActiveX control은 JAQS를 통하여 DBMS를 기반으로 하는 대규모 서비스를 행할 수 있다.

JAQS는 서버와 클라이언트간 통신 부하를 최소로 하고 있으며 하나의 프로세스가 디몬형태로 동작되고 그 프로세스 내에서 쓰레드를 최대한 활용하고 있는 구조로 병렬성을 최대한 보장하며 플랫폼에 독립적으로 동작한다.

또한 JAQS의 성능을 평가하고 CGI 실행화일 방식과 CGI 응용서버 방식과 같은 데이타베이스 통로 구조의 성능과 비교하였다. 예견했던 바와 같이 JAQS는 다른 데이타베이스 통로 구조에 비해서 월등한 성능을 보여주고 있으며 DBMS 자체의 성능과 결과 집합의 크기에 성능이 좌우됨을 알 수 있다.

JAQS는 JAQS 관리자(JAQS Manager)와 함께 분산되어 있는 이기종 데이타베이스를 통해서도 효율적인 웹 서비스를 제공할 수 있다.

참 고 문 헌

[1] T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielson and A Secret, "The World-Wide Web.," Communications of the ACM, Vol. 37, No. 8, pp. 76-82, August 1994.

[2] D Robinson, The WWW Common Gateway Interface Version 1.1, Internet draft, January 1996

[3] P.-C. Kim, "A Taxonomy on the Architecture of Database Gateways for the Web," Proc. of the 13th ICAST and 2nd ICMIS, Shaumburg, IL, pp. 226-232, April 1997.

[4] T Berners-Lee, Hypertext Transfer-HTTP/1.0, Internet RFC 1945, May 1996.

[5] T. Berners-Lee, Uniform Resource Locator, Internet RFC 1738, December 1994

[6] D. M. Kristol, Proposed HTTP State Management Mechanism, Internet Draft, June 1996.

[7] Ken Arnold, James Gosling, The Java Programming Language, Addison-Wesley, 1996.

[8] James Gosling, Bill Joy, Guy Steele, The Java Language Specification, Addison-Wesley, 1996.

[9] URL : http://java.sun.com

[10] URL : http://www.allaire.com

[11] URL : http://gdbdoc.gdb.org/letovsky/genera/genera.html

[12] URL : http://www ncsa.uiuc.edu/SDG/People/jason/pub/gsql/starthere.html

[13] URL : http //archive eso.org/wdb/html

[14] URL : http://www.o2tech.fr

[15] URL : http://www.progress.com

[16] Graham Hamilton, Rick Cattell, JDBC : A Java SQL API version 1.20, Sun Microsystems, January 1997.

[17] Graham Hamilton, R. G. G. Cattell, Maydene Fisher, JDBC Database Access with Java A Tutorial and Annotated Reference, Addison-Wesley, 1997.

[18] URL : http://www.microsoft.com/com/activex.asp

[19] URL : http://www.microsoft.com/scripting/default.htm

[20] URL : http://www.netscape.com/eng/javascript

[21] URL : http://www.microsoft.com/data/odbc

[22] L Latham, "Client/Server Computing - Strategic Directions, Tactical Solution," Inside Cartner Group This Week, Vol. X, No. 20, Gartner Group, May 1994.

[23] R. Schulte, "Two-Tier vs. Three-Tier Trade-Offs," SMS:K-401-1564, SMS Research Note Gartener Group, December 1994.

[24] R. Schulte, "Three-Tier Software Architecture in Perspective," SMS:K-401-1565, SMS Research Note Gartener Group, December 1994.

[25] Jim Melton, Alan R. Simon, Understanding The New SQL, A Complete Guide, Morgan Kaufmann, 1993.

[26] G. Trent and M. sake, "WebSTONE : The First Generation in HTTP Server Benchmarking," Silicon Graphics, February, 1995.

[27] S. Srinivasan, "Webpest - A Tool to Evaluate Hypertext Server Performance," Fourth International World Wide Web Conference, Boston, 1995.

최 원 익
1996년 서울대학교 컴퓨터공학과 졸업(공학사). 1998년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사). 1998년 ~ 현재 서울대학교 대학원 컴퓨터공학과 박사과정. 관심분야는 멀티미디어 데이타베이스, 웹, 다차원 공간 인덱스

김 형 주
제 5 권 제 1 호(C) 참조

이 석 호
1964년 연세대학교 정치외교학과 졸업. 1975년, 1979년 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979년 ~ 1982년 한국과학원 전산학과 조교수. 1982년 ~ 1986년 한국정보과학회 논문편집위원장. 1986년 ~ 1988년 한국정보과학회 부회장. 1988년 ~ 1989년 미국 IBM T.J.Watson 연구소 객원교수. 1988년 ~ 1990년 데이타베이스연구회 운영위원장. 1989년 ~ 1991년 서울대학교 중앙교육연구전산원 원장. 1994년 한국정보과학회 회장. 1997년 ~ 현재 한국학술진흥재단 부설 첨단학술정보센터 소장. 1982년 ~ 현재 서울대학교 컴퓨터공학과 교수. 관심분야는 데이타베이스, 멀티미디어 데이타베이스