SRP RDBMS를 위한 Web 게이트웨이
(Web Gateway for SRP RDBMS)

최 일 환 † 이 상 철 † 김 형 주 ††
(Il-Hwan Choi) (Sang-Cheol Lee) (Hyoung-Joo Kim)

요 약 웹의 등장은 인터넷 인구의 엄청난 성장을 가져왔다. 이와 더불어 최근에는 가상 금융 및 전자 상거래 시장 등을 지원하는 웹 상의 데이타베이스 관련 상품들이 속속 등장하고 있다. 이에 기존의 데이타베이스 시스템과 클라이언트-서버 방식으로 동작하던 응용 프로그램들은 웹 환경에 적합한 상호 연동 방식을 요구한다.

본 논문에서는 관계형 DBMS인 SRP를 웹 상에서 동작시키기 위한 통로인 SWeS를 제시한다. SWeS는 CGI를 이용한 응용서버 방식으로 구현되었으며, 동적 서버 교체와 페이지 단위의 결과 전송 기법 등을 사용해 가용성 및 동시성, 확장성을 높이는 구조를 갖는다.

Abstract As the World Wide Web's popularity grows very rapidly, database applications that support internet commerce like virtual banking and electronic commerce come out with a rush. So database applications operating on the client-server architecture require new architecture which can support efficient operations on the Web environment.

In this paper we present SWeS, the relational database gateway which connects SRP RDBMS to Web. SWeS has CGI application-server architecture. SWeS provides efficient structure by using both dynamic server replacement scheme and page level transmission of results to achieve availability, concurrency, and scalability.

1. 서 론

1.1 연구 배경 및 필요성

CERN에서 처음 월드와이드웹(World Wide Web, 웹)을 발표한 이후, 웹은 사용의 편리함 등으로 인하여 인터넷 인구의 엄청난 성장을 가져왔고, 최근 인터넷에 접속하는 수단으로 가장 많이 사용되고 있다. 웹은 HTML이라는 표준언어[9]를 사용하여 다양한 플랫폼에서 동작할 수 있고, 손쉽게 네트워크 환경에서 GUI 환경을 구축할 수 있는 등의 장점으로 인하여 빠르게 대중화되어 가는 추세다. 이와 더불어 최근에는 가상 금융, 전자 상거래 등을 지원하는 웹과 연동하는 데이타베이스 관련 상품들도 속속 등장하고 있다. 이에 기존의 데이타베이스 시스템(DBMS, Database Management System)과 클라이언트-서버 방식으로 동작하던 응용들은 웹 환경에 적합한 상호 연동 방식을 필요로 하게 되었다.

웹과의 연동을 통한 데이타베이스 시스템 응용의 개발은 다양한 플랫폼을 지원하고, 다수의 사용자를 지원할 수 있고, GUI 환경을 손쉽게 구축할 수 있는 이득을 준다[11].

한편 웹은 문서 관리의 측면에서 지니는 약점을 데이타베이스의 뛰어난 문서 관리 기능을 통해 보완할 수 있는 이득을 얻을 수 있다.

이러한 웹과 데이타베이스를 연동하는 방법으로 데이타베이스 통로(gateway)가 많이 사용된다(그림 1 참조). 일반적으로 데이타베이스 통로를 구현하는 방법은 웹에서 동적인 웹 페이지 구성을 가능하게 하는 표준인 CGI를 이용하는 것이다. 이러한 CGI를 이용하여 데이타베

· 본 논문은 공업기반기술개발 사업 과제인 '웹 기반 그룹웨어 개발(96-1-28-3, 1996 11 1~1997.10.31)'과 산학연 공동기술개발 사업 과제 '미래로 D/B 통합 소프트웨어 시스템을 위한 SRP 확장(1997.1.1~1998.12.31)'에 의해 지원 받았음
† 학생회원 : 서울대학교 컴퓨터공학과
ihchoi@oopsla snu.ac.kr
sclee@oopsla.snu.ac kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@oopsla snu.ac.kr
논문접수 : 1997년 12월 12일
심사완료 : 1998년 7월 13일

이스 통로를 구축하는 경우, CGI 프로그램은 데이타베이스 시스템의 클라이언트로 동작하게 된다. 그러나 CGI를 사용하는 경우, 매번 사용자로부터 요청이 들어올 때마다 새로운 프로세스가 생성되는 문제점을 지닌다. 따라서 다수의 사용자를 가지는 웹 환경에서는 사용자의 수에 따라 프로세스 수가 계속해서 늘어나기 때문에 웹 서버에 너무 많은 부하(load)를 준다. 또한 매번 프로세스가 실행될 때마다 데이타베이스와 연결 및 연결 해제를 해야 하는 부하를 지니게 된다.

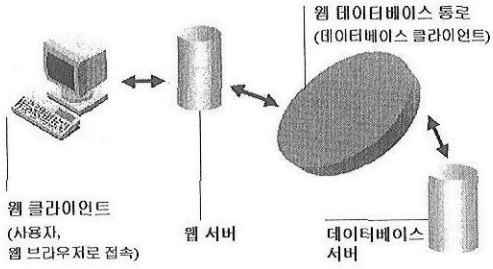


그림 1 웹 데이타베이스 통로

본 논문에서는 서울대학교 객체지향 시스템 연구실에서 개발한 관계형 DBMS인 SRP[12]를 웹과 연동시키기 위한 통로인 SWeS(SRP Web application Server)를 구현하였다. SWeS에서는 앞에서 언급한 CGI를 이용하는 방식이 지니는 문제점을 해결하기 위한 구조를 제안하고 구현하였다. 여기서는 SWeS 서버가 데이타베이스 시스템과 관련된 작업을 전담한다. CGI 프로그램을 통해 들어온 각 사용자의 요구는 SWeS 조정자를 통하여 대기상태(idle state)의 SWeS 서버에 적절히 할당된다. 특히, 한 사용자로부터의 연속적인 요구를 특정 서버가 전담해서 처리하지 않고 여유가 있는 서버에 분배해 주는 동적 서버 교체 방식을 통해 SWeS 서버의 가용성을 높이고, 전체 시스템의 효율성을 증대시킬 수 있다. 또한 SWeS 서버에서는 페이지 단위의 결과 전송을 통해 일정한 양의 요구만을 처리할 수 있으며, 이를 통한 부하 균형(load balancing)을 이룰 수 있다.

1.2 논문의 구성

2장에서는 현재 나와있는 여러 데이타베이스 통로들에 대해서 살펴보고, 유사 연구 분야로 인터넷에서의 트랜잭션 처리에 관련한 사항에 대해 간단히 살펴보기로 한다. 3장에서는 웹용 데이타베이스 통로의 여러 가지 고려사항을 알아보고, 보다 효율적인 웹용 데이타베이스 통로의 개발을 위해 고려할 점을 제시한다. 4장에서는 SWeS의 설계 및 구현에 대해 살펴보고, SWeS가 지니는 장단점을 제시한다. 5장에서는 결론 및 향후 연구 계획에 대해 살펴보기로 한다.

2. 관련 연구

2.1 관계형 데이타베이스에서의 웹 통로

Web/Genera[15]는 관계형 DBMS인 Sybase DBMS와 웹을 통합시켜주는 무료 소프트웨어이다. 이를 이용하여 기존의 데이타베이스에 대한 웹 전위 처리기를 만들거나 새로운 데이타베이스를 생성할 수 있다. 모든 동작은 sybfmt라는 프로그램을 통해서 이루어진다. sybfmt는 데이타베이스 통로로 동작하여, 주어진 인자들로부터 SQL문을 구성하여 수행한다. 또한 sybfmt은 기존의 데이타베이스에서 객체(object)들을 뽑아내어 선택된 포맷으로 출력해주는 일도 수행한다. 이렇게 함으로써 사용자는 기존에 구축되어 있는 데이타베이스에 별다른 작업 없이 웹을 통해 접근할 수 있다. 하지만 Sybase는 관계형 DBMS이므로 객체라는 개념을 갖고 있지 않기 때문에, sybfmt는 사용자가 스스로 작성한 스키마를 이용하여 데이타베이스에서 객체를 뽑아내는 방법을 사용한다.

SQL을 지원하는 관계형 DBMS를 위한 무료 소프트웨어인 GSQL[7]과 WDB[10]는 각각 PROC과 FDF라는 양식 정의 파일을 만들어 사용한다. 사용자는 양식 정의 파일을 작성하여 사용자 질의를 위한 웹 상의 입출력 화면을 구성할 수 있다. 하지만 이러한 양식 정의 파일에서 지원하는 제한적인 형식만을 사용해야 하는 단점을 지닌다.

한편, MORE(Multimedia Oriented Repository Environment)[1] 시스템[4], Zelig 프로젝트[14], Tsukuba 대학의 DALgate CGI[3]등에서는 HTML 페이지의 동적 생성을 보여준다. 이들 시스템에서는 사용자로부터 로그인에 필요한 정보만을 받아들인 후, 그 다음부터는 시스템 데이타베이스를 분석하여 동적으로 HTML 페이지를 생성해 준다. 따라서 사용자는 SQL이나 다른 언어 및 프로토콜에 대한 별도의 지식 없이도 손쉽게 웹을 통해 데이타베이스를 생성, 관리할 수 있다. 이들 중 DALgate CGI에서는 전자 메일을 사용한 지연 응답(deferred reply)과 전송되는 데이타 양에 제한을 두도록 하는 방법을 통해 부하 균형을 이루는 구조를 보인다.

관계형 DBMS인 IBM DB2에서는 웹과의 연동을 위

1) MORE 시스템은 RBSE(Repository Based Software Engineering) 프로젝트의 한 부분으로 만들어졌다.

해 DB2 WWW Connection[8]을 구현했다. 여기서는 상호-언어 변수 치환(cross-language variable substitution)이라는 방식을 사용하여 HTML의 질의어 폼(form)의 생성과 출력, SQL의 질의 및 갱신 등의 모든 기능을 사용할 수 있도록 한다. 이를 통해 특정 응용을 위한 단일 데이타베이스 응용 프로그램이 가지는 문제점들을 해결해 준다. 특정 응용을 위한 단일 데이타베이스 응용 프로그램을 작성하는 경우, 응용 프로그래머가 CGI및 DBMS 프로그래밍 인터페이스에 대한 사전 지식을 지녀야 하는 문제를 갖는다. 또한 작성된 코드에 대한 이해가 힘들고, 추가되는 HTML의 새로운 기능이 응용 프로그램에 쉽게 도입되기 어렵다는 단점이 있다. DB2 WWW Connection에서 제공하는 상호-언어 변수 치환의 방식을 사용하면, 응용 프로그램 작성자는 HTML과 SQL만을 사용하여 응용 프로그램을 작성할 수 있다. 또한, HTML과 SQL부분이 각기 나뉘어 있기 때문에 쉽게 코드를 이해할 수 있다는 장점을 지닌다.

지금까지 살펴본 데이타베이스 통로들은 웹과 데이타베이스를 연동하는데 있어서 응용 프로그램을 작성하는 방법에 대부분 주안점을 두고 있다. 하지만 보다 효율적인 데이타베이스 통로를 만들기 위한 구조에 대한 언급은 별로 하고 있지 않다. 이는 주로 관계형 데이타베이스 상에서 구현된 초기의 웹 통로들이 구현 방법으로 단순히 CGI를 이용하여 직접 수행하고 결과를 넘겨주는 방식을 사용했기 때문이다. 최근에는 여러 곳에서 보다 효율적인 구조를 갖는 데이타베이스 통로를 구현하기 위한 노력을 하고 있다. 이러한 노력은 주로 객체지향 데이타베이스나 객체 관계형 데이타베이스에서 볼 수 있다.

2.2 객체지향 데이타베이스에서의 웹 통로

객체지향 DBMS인 O_2 시스템을 웹과 연동하기 위한 통로인 O_2Web[16]에서는 본 논문에서 구현하고 있는 SWeS와 비슷한 구조를 지닌다. 여기서는 사용자의 요구를 전달하는 URL에 직접 OQL문을 입력한다. O_2Web은 O_2Web gateway, O_2Web server, O_2Web dispatcher라는 세 개의 요소들로 구성된다. 웹 브라우저를 통해 웹 서버에게로 전달되어진 사용자의 요구는 O_2Web gateway를 통해 O_2Web server로 전달된다. 즉 실제 데이타베이스와의 연결은 몇 개의 서버(O_2Web server)가 담당함으로써 CGI로 생성된 연결 통로(O_2Web gateway) 프로세스의 크기를 줄이고, 이를 통하여 대용량의 요구를 처리하는데 있어서 CGI 프로세스로 인한 부하를 줄인다. 또한 O_2Web dispatcher를 두어 O_2 gateway로부터의 요구를 각 서버 별로 적절히 분배함으로써 부하 균형을 이루는 구조이다.

2.3 객체 관계형 데이타베이스에서의 웹 통로

객체 관계형 DBMS인 UniSQL사는 웹과의 연동을 위한 통로로 UniWeb[17]을 제공한다. UniWeb도 역시 O_2와 비슷한 구조를 보여준다. 여기서는 디스패처라는 프로세스가 사용자로부터 들어온 요구를 UniTCL 서버에게 전달해 주는 방식을 택하고 있다. UniWeb에서는 응용 프로그램을 작성하는 방법으로 Tcl 스크립트를 HTML에 삽입하는 방법을 사용한다. 이 방법을 통하여 Tcl이 지닌 확장성 및 프로그래밍 기능을 이용하여 HTML의 한계를 어느 정도 극복하는 동시에 손쉽게 데이타베이스로 접근을 할 수 있는 장점을 지닌다.

2.4 기타 - 인터넷에서의 트랜잭션 처리 관련

현재 사무 환경을 지원하기 위한 전산환경은 웹을 통해 모든 업무를 처리할 수 있도록 하는 인트라넷의 구축이 일반화되는 추세다. 그러나 기존의 웹 서버는 복잡한 플로우를 가지는 업무에 대한 지원이 어렵다. 또한 웹 서버가 클라이언트에 대한 상태를 유지하지 않기 때문에, 여러 페이지에 걸친 작업에 대한 단일 트랜잭션 보장이 어렵고, 복잡한 데이타베이스 응용을 작성하기가 어렵다. 이러한 결점을 극복하고자 기존의 서버의 기능을 확장하여 인트라넷/인터넷 환경에서 신뢰성 있고, 확장성이 좋은 트랜잭션 서버의 개발이 필요하다. 트랜잭션 서버를 통해 인터넷상에서 온라인 트랜잭션 처리(OLTP, On-Line Transaction Processing)를 위한 보다 유연하고 안정적인 기술을 확보하며, 이를 전자 상거래 시스템의 미들웨어로 활용할 수 있다.

현재 Microsoft Active Server Page[18], Oracle Web Application Server[19], Sybase Jaguar CTS[20] 등에서 컴포넌트를 통한 기능확장 및 트랜잭션 모니터링 등의 기능을 제공하는 트랜잭션 서버들의 개발이 활발하게 이루어지고 있다.

3. 웹용 데이타베이스 통로

웹 상에서 데이타베이스 응용 프로그램을 작성하는 것과, 일반적인 클라이언트-서버 환경에서 응용 프로그래밍을 작성하는 것과의 가장 큰 차이는 HTTP 프로토콜이 기본적으로 상태 비 보존(stateless) 프로토콜이라는데 있다[13]. 이는 웹 서버에서 클라이언트의 상태에 대한 정보를 관리하지 않음으로써 다수의 사용자가 접속하더라도 서버에 부하를 적게 하기 위함이다. 따라서 브라우저를 통해 접속하는 웹 클라이언트는 새로운 페이지를 볼 때마다 웹 서버에 새로운 연결을 해야만 한다. 웹 서버는 클라이언트의 연결에 관한 정보, 즉 상태

를 유지하지 않기 때문에 기존에 접속했던 클라이언트라도 언제나 새로운 클라이언트와 마찬가지로 처리된다. 이러한 구조는 트랜잭션 처리 등의 상태를 중시하는 데이타베이스 응용에는 부적합하다. 따라서 데이타베이스 통로라는 새로운 구조를 필요로 하게 되는 것이다.

3.1 통로 설계 시 고려사항

보다 성능이 뛰어난 웹용 데이타베이스 통로의 설계를 위해서는 다음의 사항에 중점을 두어 설계를 하여야 한다.

확장성(scalability) 웹은 인터넷을 통해 누구든 접근이 가능하기 때문에 기존의 클라이언트-서버 환경에 비해 엄청나게 많은 수의 사용자를 가진다. 뿐만 아니라 사용자의 수는 앞으로 더욱 엄청난 규모로 증가할 것이다. 따라서 계속해서 늘어나는 사용자의 수를 고려할 때, 이들을 서비스하는데 있어서 데이타베이스 통로가 장애가 되어서는 안 된다.

성능(performance) 통로로 인한 성능 저하를 최소화해야 한다. 사용자로부터 요구를 처리하때 걸리는 시간이 웹과 데이타베이스 자체의 수행 시간에 주로 영향을 받도록 해야 한다.

가용성(availability) 웹용 데이타베이스 통로는 자체적으로 장애(failure) 상황에서도 동작할 수 있는 기능을 지녀야 한다.

동시성(concurrency) 웹용 데이타베이스 통로는 동시에 많은 수의 사용자를 서비스 할 수 있어야 한다.

독립성(independence) 웹용 데이타베이스 통로는 웹이나 데이타베이스의 변화에 영향을 받지 않는 완벽한 독립성을 보장해 주어야 한다.

유연성(flexibility) 웹용 데이타베이스 통로를 이용하여 작성된 응용 프로그램은 수정이 용이하고, 새로운 기능이 추가되는 HTML이나 SQL의 변화를 손쉽게 반영할 수 있어야 한다.

3.2 실행 구조별 비교

2장에서 살펴보았듯이 현재 나와있는 웹용 데이타베이스 통로들은 그 실행 구조에 따라 몇 가지 형태로 분류할 수 있다. 크게 서버 확장과 클라이언트 확장으로 나뉘며, 서버 확장은 CGI를 사용하는 방식과 웹 서버의 API를 이용하는 확장 API 방식 등이 있고, 클라이언트 확장은 외부 뷰어를 사용하는 방식과 브라우저를 확장하는 방식 등이 있다[5]. 여러 방식 중, CGI를 사용하는 방식과, 웹 서버의 API를 이용하는 방식이 일반적으로 많이 사용되고 있다.

2.1절에 나와있는 초창기의 웹용 데이타베이스 통로들은 주로 관계형 데이타베이스 상에서 CGI를 사용하여 구현되었다. 이 방식에서는 사용자로부터 요구가 들어올 때마다 새로운 CGI 프로세스를 생성시켜 수행해 준다. 따라서 다수의 사용자가 접속하는 웹 환경에서는 각 사용자마다 각각의 CGI 프로세스가 동작하여, 시스템에 상당한 부하를 가져온다. 이러한 문제로 인하여 최근에는 CGI 응용서버 방식, 혹은 웹 서버의 API를 이용하는 확장 API 방식들을 사용하여 문제를 해결하고 있다[2, 16, 17, 19].

CGI 응용서버 방식은 응용서버라는 데몬(daemon) 프로세스를 두어, 기존의 CGI를 사용한 방식의 문제점인 프로세스 실행 시 부하를 덜어주는 방식이다. CGI 프로세스가 클라이언트로부터 들어온 요구를 응용서버에게 전달하고, 응용서버로부터의 결과를 받아 이를 다시 클라이언트에게 돌려주는 일만을 수행한다. 따라서 프로세스 크기가 아주 작게 되어 많은 수의 프로세스가 동작하더라도 시스템에 큰 무리를 주지 않는다. 또한 한 사용자가 여러 요구를 연속해서 하는 경우에도 효율성을 높일 수 있다. 기존의 CGI를 사용하는 방식에서는 사용자로부터 각 요청이 들어올 때마다 일일이 데이타베이스와의 연결 및 연결해제를 해주어야 한다. 하지만 CGI 응용서버 방식은 한 서버가 특정 사용자의 요구를 처리하기 위해 일단 데이타베이스와 연결을 한 경우 그 요구의 처리가 끝나도 계속 연결을 유지하고 있다. 따라서 뒤이어 들어오는 연속되는 요구도 새로운 연결의 부하 없이 처리해 줄 수 있는 장점을 지닌다. 반면 사용자의 상태 관리에 대한 문제점을 지니는데 이에 대해서는 3.3절에서 자세히 다루기로 한다.

확장 API 방식의 경우 데이타베이스 응용 프로그램이 웹 서버 프로세스와 동적으로 연결되어 실행되므로 CGI 방식의 프로세스 관리비용이 줄어드는 장점을 지닌다. 반면, 특정 웹 서버에서만 실행되므로 이식성이 떨어지는 단점이 있다.

[1]에서는 이 세 가지의 방식에 대해 성능 평가를 보여주고 있는데, CGI 실행화일 구조보다 CGI 응용서버 방식과 확장 API 방식이 더 나은 성능을 보인다. 이중 CGI 응용서버 방식은 이 방식이 지니는 복잡한 프로세스 구조가 성능을 그리 크게 저하시키지 않을 뿐만 아니라, 기존의 웹 서버를 그대로 이용할 수 있기 때문에 이식성 또한 뛰어남을 알 수 있다.

3.3 웹 응용 프로그램에서의 트랜잭션 처리

현재의 HTTP 프로토콜을 이용하는 웹 응용에서는 웹 서버가 클라이언트에 대한 상태 및 연결 정보 등을 유지하지 않기 때문에, 웹 상에서 안정된 온라인 트랜잭션 처리 환경을 보장할 수 없다. 즉, 가상 금융 및 전자

상거래 등의 응용에서 응용 프로그램 개발자가 많은 위험 부담을 지게 되는 것이다. 따라서 이러한 문제들을 해결하기 위해 새로운 트랜잭션 서버들을 개발하고 있다(2.4절 참조).

단일 페이지 환경의 웹 응용 프로그램의 경우에는 별다른 상태를 유지할 필요가 없기 때문에, 상태를 유지하지 않는 HTTP 프로토콜이 아무런 문제를 가져오지 않는다. 하지만 여러 페이지를 갖는 응용 프로그램의 경우에는 상태를 유지하지 않는 것이 문제가 될 수 있다. 예를 들어 가상의 쇼핑 공간을 생각해 보자. 가상의 쇼핑 공간은 각 분야별로 여러 페이지에 걸쳐서 구성되고, 사용자는 여러 페이지를 이동해 다니면서 구매를 하는 형태를 갖는다. 사용자가 최종적으로 계산을 하는 페이지로 갔을 때, 응용 프로그램은 사용자가 이전 페이지에서 구입을 결정한 물품들에 대한 정보를 알고 있어야 한다. 따라서 클라이언트의 상태에 관한 정보를 관리하지 않는 현재의 HTTP 프로토콜 하에서는 응용 프로그램이 별도로 클라이언트에 대한 상태를 관리해야 한다.

이러한 클라이언트에 대한 정보, 즉 상태를 관리하는데 있어서 가장 큰 문제는 상태를 해제할 시점을 알기가 힘들다는 것이다. 그것은 사용자가 현재 계속 그 서버에 접근해 있는 상태인지, 아니면 이미 다른 서버로의 연결을 통해 그 서버를 떠나버렸는지를 알 수 있는 방법이 없기 때문이다. 앞서의 가상 쇼핑 공간의 경우로 다시 돌아가 보자. 이 쇼핑 공간을 A라고 하자. 사용자는 이곳 저곳 이동해 다니며 구매를 하다가 갑자기 생각이 바뀌어 다른 쇼핑 공간 B로 가 버릴 수 있다. 이 때, A 쇼핑 공간이 있는 웹 서버에서는 사용자에 대한 연결 정보를 갖고 있지 않으므로, 사용자가 그 쇼핑 공간을 떠나 B 쇼핑 공간으로 가버렸는지를 알 수 있는 방법이 없다. 그밖에 사용자가 쇼핑을 하는 도중 네트워크의 문제로 인하여 웹 서버로의 연결을 할 수 없는 상황이 발생할 수도 있다. 이러한 문제들 때문에 상태를 해제하는 방법으로 일반적으로 시간제한(timeout)을 사용한다. 하지만 사용자의 입장에서는 접속 중에 강제로 상태가 해제될 가능성이 있으며, 웹 서버의 입장에서는 불필요한 상태를 유지하게 되는 낭비를 초래할 수 있는 문제점을 여전히 지닌다. 상태를 해제하는 다른 방법으로 사용자에게 상태의 해제 여부를 물어 해결하는 방법이 있다. 하지만 이 방법은 사용자가 상태에 대한 개념을 갖고 있어야 하며, 악의를 가진 사용자에 의해 악용될 우려가 있기 때문에 적합한 방법이라 볼 수 없다.

따라서 이러한 문제의 해결을 위해서는 각 응용에 따라 가장 적합한 상태 유지 시간을 분석하고, 이를 토대로 하여 시간제한을 함으로써 사용자에게 최소한의 불이익이 돌아가도록 해야 한다. 다른 해결책의 하나로 이 절의 첫 부분에서 밝혔듯이 응용에 적합한 새로운 트랜잭션 서버를 개발하는 방법이 있다. 아울러 엄청난 속도로 성장하는 인터넷 및 인트라넷 시장에 적응할 수 있도록, 기존의 웹 서버에서 사용하는 HTTP 프로토콜의 변화가 필요할 것이다.

4. SWeS의 설계 및 구현

4.1 구조

4.1.1 전체 구조

SWeS는 CGI 응용서버 방식(3.2절, [5] 참조)으로 구현되었으며, 크게 세 가지 주요 요소로 구성된다. 웹 클라이언트인 사용자로부터의 요구를 서버로 전달하는 SWeS 중개자(SWeS agent), SWeS 중개자에게 적절한 서버를 할당해 주는 SWeS 조정자(SWeS coordinator), SRP 서버와의 연결 및 사용자로부터의 요구에 대한 실제 처리가 일어나는 SWeS 서버(SWeS server)[2]가 그것이다(그림 2 참조). SWeS 중개자는 CGI 프로세스로서, 사용자의 요구를 SWeS 서버에게 전달해주고, SWeS 서버로부터 건네 받은 수행결과를 다시 사용자에게 전달하는 일을 수행한다. 실제 데이타베이스 시스템과 관련된 동작은 데몬 프로세스인 SWeS 서버가 전담해서 수행한다. 이렇게 함으로써 CGI 프로세스의 크기를 줄이고, 다수의 사용자로부터 요구가 들어오더라도 시스템에 부하를 덜어준다. 특히 SWeS는 사용자로부터 연속되는 요구가 들어오는 경우, 매 요구가 들어올 때마다 여유가 있는[3] SWeS 서버를 찾아서 요구를 수행하는 구조를 갖는다. 반면, 비슷한 CGI 응용서버 방식으로 구현된 UniWeb의 경우는 특정 사용자의 요구를 한 서버가 전담하여 관리한다[2]. 즉 서버는 요구를 처리해 주던 사용자로부터 연속해서 들어오는 모든 요구를 처리해 준 후에야 비로소 다른 사용자의 요구를 처리해 줄 수 있다. 이 경우 서버는 사용자로부터 한참동안 요구가 들어오지 않더라도 그 사용자의 요구를 기다리며 다른 사용자의 요구를 처리해 주지 못하는 일이 발생한다. 결국 서버는 아무런 다른 일도 하지 못한 채 기다려야 하므로 시스템의 입장에서는 실제 동작하는

2) SRP[12]는 서울대학교 컴퓨터공학과 객체지향시스템 연구실에서 개발한 관계형 데이타베이스 시스템으로서 클라이언트-서버의 구조를 갖는다. SWeS 서버는 SRP 클라이언트의 응용 프로그램 도구를 사용하여 제작되었으며, SRP 서버에 연결하여 데이타베이스 관련 처리를 수행한다

3) 대기상태(4.3.1절 참조)를 말한다.

서버의 수가 하나 적어진 것과 마찬가지가 된다. 즉 서비스 해 줄 수 있는 사용자의 수가 그만큼 줄어들게 된다. 이러한 문제점을 해결하기 위해 SWeS에서는 한 사용자가 특정 서버를 독점하지 못하게 함으로써 SWeS 서버의 가용성을 높여주는 구조를 사용한다. 물론 사용자로부터 연속적인 요구가 들어왔을 때, 우선적으로 이전에 그 사용자의 요구를 처리해 주던 SWeS 서버가 요구를 처리해 줄 수 있을지를 살펴 최적화 된 동작을 하도록 해 준다. 만일 그 서버가 다른 사용자를 위해 할당되지 않은 상태라면, 계속해서 그 사용자의 요구를 처리해 주도록 함으로써 다시 데이타베이스에 연결하지 않아도 되는 장점을 살릴 수 있다.

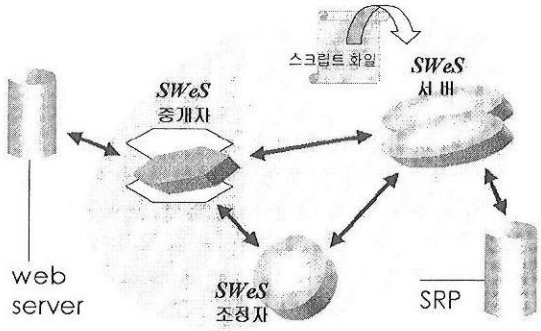


그림 2 SWeS의 전체 구조

4.1.2 동작 과정

그림 3은 SWeS의 동작 과정을 보이고 있다. 맨 처음 사용자로부터 요구가 들어오면 웹 서버에서는 SWeS 중개자에게 사용자의 요구를 전달해 준다. 이 때 SWeS 조정자는 SWeS 중개자가 현재 동작하는 SWeS 서버들 중에서 어떤 서버에 연결할 지를 결정해 준다. SWeS 중개자는 이 모든 요구를 SWeS 서버에 전달한 후, 결과를 기다린다. SWeS 조정자에서는 적절한 알고리즘에 따라, 주어진 사용자를 서비스해 줄 SWeS 서버를 선정하고, 선정된 서버를 중개자에 알려준다. SWeS 중개자는 이렇게 선택된 SWeS 서버에 연결한다. SWeS 서버에서는 중개자로부터 건네 받은 사용자의 요구를 가지고 질의문을 구성하여 데이타베이스 서버인 SRP 서버에 넘겨준다. SRP 서버에서는 질의문을 수행한 후 결과를 다시 SWeS 서버에 넘겨준다. SWeS 서버에서는 이 결과를 HTML 형식으로 구성하여 SWeS 중개자에 전달한다. 최종적으로 SWeS 중개자가 결과를 웹 서버에 전달해 줌으로써 사용자에게 결과 화면을 보여주게 된다.

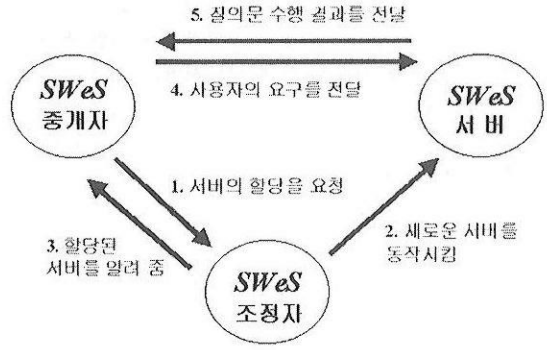


그림 3 SWeS의 동작 과정

4.1.3 메시지 전달 방식

SWeS 중개자와 SWeS 조정자, SWeS 중개자와 SWeS 서버는 각각 TCP 소켓을 사용하여 메시지를 주고받으며, SWeS 조정자와 SWeS 서버는 공유 메모리(shared memory)를 통해 서버들의 상태와 현재 상태를 유지하고 있는 사용자에 대한 정보를 관리한다. SWeS 조정자는 공유 메모리를 통하여 사용자의 요청을 서비스할 SWeS 서버를 결정하며, SWeS 서버는 공유 메모리로부터 사용자의 이전 상태에 대한 정보를 얻는다. SWeS는 한 사용자로부터의 연속적인 요구를 특정 서버가 전담하여 처리해 주지 않고 여유가 있는 서버가 처리해 주는 구조이다. 따라서 사용자에 대한 정보를 여러 SWeS 서버들이 공유할 수 있어야 하며

이를 공유 메모리를 통해 해결한다.

4.1.4 상태 정보 관리

공유 메모리에는 각 SWeS 서버와 사용자 별로 다음과 같은 정보들을 관리한다. 각 항목에 대한 좀더 상세한 설명은 4.3절에서 살펴보기로 한다.

우선 서버에 대한 정보로는 현재 서버의 상태와 요구가 들어온 사용자의 이름을 관리한다. 서버의 상태는 대기상태(idle), 준비상태(wait), 사용 중(busy)의 총 세 가지의 상태를 지니며, 이 정보를 바탕으로 조정자는 대기중인 서버를 사용자에게 할당해 준다.

사용자에 대한 정보로는 사용자의 이름이 있다. 사용자의 이름은 쿠키(cookie)[4]값으로 설정되며, 사용자[5]마

4) 쿠키는 HTTP가 지니는 상태 비 보존의 단점을 보완하고자 제안되었다[6]. 쿠키를 사용하여 웹 서버는 클라이언트와 상태 정보를 주고받을 수 있다.

5) 여기서 사용자란 특정 브라우저로부터 접속된 한 사용자를 말한다. 따라서 동일한 브라우저로부터의 계속되는 요청은 한 사용자로부터 연속된 요청이 들어온 것으로 판단한다.

다 할당된 고유한 이름이다. 이밖에 SRP 연결 시 사용되는 클라이언트 식별자, 요구를 수행했던 사용자의 질의어에 관련된 정보 등을 관리한다.

4.2 실행 방법

HTML에서 SWeS를 실행하기 위해서는 다음의 요청을 HTML 문서에 포함시켜 웹 서버에 전달해 준다

{SWeS 실행 화일 이름}/{스크립트 화일 이름}/{명령어}

여기서 {SWeS 실행 화일 이름} 부분은 SWeS 중개자를 불러주는 부분으로, 일반적으로 SWeS가 동작하는 웹 서버의 CGI 실행 화일을 부르는 방법과 동일하다.

{스크립트 화일 이름}은 웹 응용 프로그램을 위해 작성된 스크립트 화일의 이름을 말한다. SWeS에서 사용하는 스크립트 화일의 형식에 대해서는 4.2.2절에서 자세히 다루기로 한다.

{명령어}는 *input*, *output* 의 두 가지가 있다. *input*은 SWeS 스크립트 파일로부터 사용자 입력 부분을 읽어들일 경우에 사용되며, *output*은 수행 결과를 출력하는 경우에 사용된다[6].

스크립트 화일을 읽어서 사용자로부터 입력을 받아들이는 화면을 보여주는 예는 다음과 같다. 여기서 스크립트 화일의 이름은 *demo1.sws*인 경우이다.

http://gaston.snu.ac.kr/cgi-bin/swes/demo1.sws/input

4.2.1 스크립트 화일의 형식 비교

웹 상에서 데이타베이스 응용 프로그램을 작성하는 방법으로는 HTML 태그를 확장하여 데이타베이스 명령어를 지원하는 방법, HTML 문서 내에 데이타베이스에 접속할 수 있도록 확장된 Perl, Tcl 등의 스크립트 언어를 삽입하는 방법, 데이타베이스에서 지원하는 C/C++ 등의 일반 프로그래밍 언어에 웹의 CGI 출력을 지원하도록 하는 방법 등이 사용된다.

HTML 태그를 확장하는 방식은 쉽게 응용 프로그램을 작성할 수 있는 장점을 갖는다 그러나 이 방식은 HTML 자체가 가지는 프로그램 언어로써의 한계를 그대로 지니게 되는 단점을 갖는다. 즉 반복 수행을 위한 루프, 변수 선언, 프로시저 등에 대한 지원을 하지 않는다.

이러한 한계를 극복하고자 기존의 스크립트 언어를 HTML에 삽입하는 방식을 도입하기도 한다. 이 방식은 스크립트 언어가 갖는 다양한 프로그래밍 기법을 사용할 수 있다는 장점을 갖는다. 하지만 이 두 방법 모두가 유연성이라는 측면에서는 단점을 지니게 된다. 즉 HTML과 SQL이 변하게 되면, 새로이 변화된 부분을 다시 추가해 주어야만 하기 때문이다. 또한 스크립트 언어를 삽입하는 경우, 응용 프로그래머가 스크립트 언어에 대한 사전 지식을 필요로 한다. 이러한 문제점은 일반 프로그래밍 언어를 사용하는 방식에서도 발생한다.

이 방식은 기능 면에서 가장 강력하지만, 스크립트 언어에 비해 훨씬 복잡한 수준의 프로그래밍을 요구하게 된다는 단점을 갖는다.

따라서 응용 프로그램을 보다 작성하기 쉽고, HTML이나 SQL에 독립적인 환경을 제공하는 방식을 필요로 하게 되는데, DB2WWW Connection[8]에서 사용하는 상호-언어 변수 치환이라는 방식은 이런 점에서 흥미롭다. 여기서는 HTML 입력과 출력, SQL문 등을 각각의 부분별로 나누어 입력한 매크로 화일을 작성한다. 이 방식을 통해 HTML과 SQL에서 지원하는 모든 기능을 특별한 변환 없이 바로 사용할 수 있다. 또한 응용 프로그램의 작성자는 HTML과 SQL에 대한 지식 이외의 별다른 언어에 대한 지식을 필요로 하지 않는다. 물론 이 방식 역시 HTML을 사용하여 사용자 화면에 입출력하기 때문에, HTML이 지니는 프로그램 언어로써의 한계는 계속 지니게 되는 단점을 갖는다.

SWeS에서는 스크립트 화일을 작성하는 방식으로 앞의 DB2WWW Connection에서 사용하는 상호-언어 변수 치환 방식을 사용하여 프로그래밍 언어에 대한 별다른 지식 없이도 손쉽게 응용 프로그램을 작성할 수 있도록 한다.

4.2.2 SWeS 스크립트 화일

SWeS의 스크립트 화일은 총 네 개의 부분으로 구성된다. 각각 변수 정의 영역, 입력화면 정의 영역, 출력화면 정의 영역, 질의문 정의 영역으로 나누어진다. 변수 정의 영역에서는 접속할 데이타베이스 이름과, 데이타베이스의 사용자 이름, 그리고 결과화면에 출력되는 행의 개수를 지정한다. 입력화면 정의 영역에서는 일반적인 HTML문을 사용하여 사용자 인터페이스를 구성한다. 이를 통해 사용자로부터 입력 값을 받아들여 질의문 정의 영역에서 정의되는 질의문을 구성한다. 출력화면 정의 영역에서는 작업을 수행하여 얻어진 결과를 사용자에게 보여주기 위한 포맷을 정의한다. 입력화면 정의 영역과 마찬가지로 HTML문을 사용하여 구성한다. 질의문 정의 영역에서는 SQL문을 선언한다. 입력화면 정의 영역을 통해 사용자로부터 얻어낸 입력 값을 사용하여 SQL문을 구성한다.

6) 각 부분에 대해서는 4.2.2절을 참조하기 바란다.

그림 4는 이러한 양식에 따라 스크립트 파일을 작성한 예를 보여준다.

```
# variable declaration
[var][
    DBNAME = srpbook;
    USER = yakobo;
    OUTROW = 10;
]

# HTML format: User-input FORM
[input][
<html>

<head><title>Demo1</title></head>

<body>
<p><h2>This is demo1 of <i>SWeS</i></h2>
<p><form method="post" action="http://gaston.snu.ac.kr/
                        cgi-bin/swes/demo1.sws/output">
Input keyword<br>
<input type=text size=10 name="name"><br><hr>
<input type=submit>
<input type=reset><br>
</form>
</body>

</html>
]

# HTML format: Showing result
[output][
<html>

<head><title>Result of Demo1</title></head>

<body>
<p><h2>Below is the result of Demo1:</h2>
<table border=1>
    <tr><th>num <th>name</tr>
    [row]{sqlbook}[
        <tr><td>$(C0)<td>$(C1)</tr>
    ]
</table>
</body>

</html>
]

# SQL part
[sql]{sqlbook}{
select *
from book
where name like '%$(name)%';
]
```

그림 4 SWeS 스크립트 파일

4.3 동작 방식

4.3.1 SWeS 서버의 상태(state)

SWeS 서버는 동시에 동작할 수 있는 최대 개수를 미리 지정해 줄 수 있다. 이 개수는 관리자가 시스템의 부하에 따라 결정한다. 각 서버는 동작하는 상황에 따라 대기상태, 준비상태, 사용중의 세 가지 상태를 갖는다.

대기상태는 서버가 사용자로부터의 요구를 기다리고 있는 상태이다. SWeS 조정자는 대기상태에 있는 SWeS 서버를 사용자에게 할당한다. SWeS 서버는 SWeS 중개자를 통해 사용자의 요구가 들어오면 자신의 상태를 사용중인 상태로 설정한다. SWeS 조정자에서는 SWeS 서버를 한 사용자에게 할당을 한 이후에 다른 사용자가 이 서버에 할당이 되지 않도록 해 주어야 한다.

SWeS 중개자가 조정자로부터 연결할 서버를 할당받았지만, 이 서버에 연결이 이루어지기 이전까지는 그 서버의 상태는 대기상태로 남아 있게 된다 따라서 SWeS 조정자에서 새로이 요구가 들어온 다른 사용자에게 이미 할당한 서버를 다시 할당해 줄 가능성이 생긴다. 이를 방지하기 위해 SWeS 조정자에서는 이미 할당한 서버를 준비상태로 둔다.

SWeS 서버는 모든 처리를 끝내고 다시 자신의 상태를 대기상태로 놓아 다른 사용자의 요구를 기다릴 수 있도록 한다.

4.3.2 SWeS 서버의 할당 과정

SWeS 조정자에서 SWeS 서버의 할당은 다음과 같은 과정을 통해 이루어진다.

1. 현재 동작하고 있는 SWeS 서버 중, 요청이 들어온 사용자의 이전 요구를 처리하고 있는 서버를 찾는다. 그러한 서버를 찾은 경우 서버가 위치하는 기계의 주소와 포트 번호를 SWeS 중개자에게 넘겨준다.
2. 서버를 찾지 못한 경우는 새로운 사용자로부터의 요청이 들어온 경우이거나, 교체된 상태에 있는 기존 사용자(swapped out user)[7]인 경우이다. 이 경우에는
   (1) 현재 동작하는 서버의 개수가 동작할 수 있는 최대 서버의 개수보다 작은 경우
      ㄱ. 대기상태의 서버가 있으면, 그 서버의 주소와 포트 번호를 SWeS 중개자에게 넘겨준다.
      ㄴ. 대기상태의 서버가 없으면, 새로운 서버를 실행시키고 이 새로운 서버의 주소와 포트 번호를 SWeS 중개자에게 넘겨준다.
   (2) 이미 최대 개수의 서버가 동작하는 경우
      ㄱ. 대기상태의 서버가 있으면, 그 서버의 주소와 포트 번호를 SWeS 중개자에게 넘겨준다.
      ㄴ. 대기상태의 서버가 없으면, 대기상태의 서버가 생길 때까지 기다린다. 이후 대기상태의 서버가 생기면 그 서버의 주소와 포트 번호를 SWeS 중개자에게 넘겨준다.

4.3.3 SRP 연결 관리

앞 절에서의 과정을 거쳐 사용자의 요구가 SWeS 서버로 들어오면, SWeS 서버에서는 SRP 데이타베이스

7) 이전에 접속했던 사용자 중 SWeS에서 그 사용자에 대한 정보를 계속 유지하고 있는 경우를 말한다.

서버에 연결하여 사용자의 요구를 처리한다. 이때 SWeS 서버는 각 사용자의 상황에 따라 SRP 서버로의 각기 다른 연결 방법을 사용한다. 이는 SWeS 서버가 사용자의 요구를 일정 량만큼[8] 처리해 준 다음, 다시 다른 사용자의 요구를 처리해 주는 방식을 사용하기 때문이다(4.1.1절 참조).

SWeS 서버가 SRP 서버로의 연결을 위해 다음의 두 가지 방법을 사용한다. 기존의 SRP 서버에 이미 존재하는 클라이언트의 세션에 다시 연결하는 방법과 새로운 세션을 생성하는 방법이 그것이다. SWeS 서버가 처음 SRP 서버에 연결하는 경우, SWeS 서버와 SRP 서버 간에 소켓 연결이 생성된다. 그리고 나서 SRP 서버에는 연결된 클라이언트에 대한 세션이 생성된다. SRP 서버는 생성된 세션을 가리키는 클라이언트 식별자를 SWeS 서버에게 반환한다. 클라이언트의 세션은 외부적으로 연결해제 명령이 수행될 때까지 SRP 서버에 유지된다. 따라서 또다른 SWeS 서버가 클라이언트 식별자를 사용하여 이미 존재하는 클라이언트 세션에 연결할 수 있다.

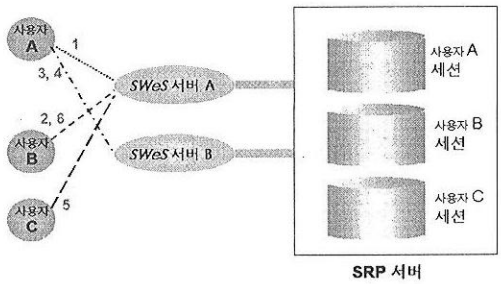


그림 5 SRP 연결 관리

그림 5에는 여러 가지 상황에서 SWeS 서버가 SRP 서버에 어떻게 연결하는지를 보여주기 위한 예가 나와 있다. 여기서 나오는 번호는 사용자로부터 요구가 들어온 순서를 나타낸다. 각 번호순으로 다음과 같은 일들이 일어나게 된다. 이 때 동작할 수 있는 최대 SWeS 서버의 개수는 2개이고, 초기 상태에는 동작하고 있는 SWeS 서버가 없다고 가정한다.

1. 맨 처음 SWeS로 사용자 A로부터 요청이 들어온 경우다. 이 경우에는 SWeS 조정자에서 SWeS 서버 A를 처음 동작시키고 이 서버를 사용자 A에게 할당해 준다. 사용자 A로부터 첫 번째 요구를 전달받은 SWeS 서버 A는 SRP 서버와의 연결을 통해 새로운 소켓을 생성한다. 그리고 SRP 서버에서는 사용자 A에 대한 세션을 생성된다. SWeS 서버 A는 사용자 A의 요구에 대한 수행을 마친 후, 그 결과를 SWeS 중개자에게 넘겨준다. 그리고 나서 SWeS 서버 A는 다른 사용자의 요구를 처리하기 위한 '대기상태'로 들어간다. 이 때 SWeS 서버 A는 사용자 A에 대한 클라이언트 식별자와 지금까지의 수행상태를 공유 메모리에 기록한다. 이렇게 기록된 정보들은 이후 사용자 A로부터 다시 요구가 들어왔을 때 사용된다.
2. 1번 과정의 수행이 끝나고 새로운 사용자 B로부터의 요구가 SWeS 서버 A로 들어온 경우이다. 이 때 SWeS 서버 A가 이전에 요구를 처리해 주었던 사용자 A는 교체(swap out)되어 공유 메모리에만 정보를 유지하게 된다. 하지만 이 경우 SWeS 서버 A와 SRP 서버와의 소켓 연결을 해제하고 새로운 소켓을 만들지는 않는다. 이미 생성된 소켓을 그대로 사용하고, 단지 사용자 B를 위한 새로운 세션을 SRP 서버에 생성한다. 물론 이 때도 앞서의 사용자 A에 대한 세션은 SRP 서버 내에 계속 유지된다.
3. 2번 과정을 수행 중에 다시 사용자 A로부터 요구가 들어온 경우를 생각해 보자. 현재 SWeS 서버 A는 사용자 B의 요청을 처리해 주고 있는 중이다. 이 경우 SWeS 조정자는 새로운 SWeS 서버 B를 동작시키고, 이 서버 B를 사용자 A의 요구를 처리해 주도록 할당해 준다. 이 때 SWeS 서버 B는 SRP 서버와의 연결을 통해 새로운 소켓을 생성한다. 하지만 사용자 A에 대한 세션은 SRP 서버 내에서 유지하던 사용자 A의 세션에 연결이 된다. 이는 SWeS 서버 B가 공유 메모리를 통해 사용자 A에 대한 정보[9]를 얻어내고, 이를 사용하여 SRP 서버에 연결함으로써 이루어진다.
4. 연속해서 사용자 A로부터 요구가 SWeS 서버 B로 전달된 경우이다. 이때 SWeS 서버 B는 이미 SRP 서버의 사용자 A의 세션에 연결되어 있는 상태이다. 따라서 SRP와 다시 연결할 필요가 없이 바로 사용자의 요구를 처리해 준다. 이렇게 함으로써 수행시간이 단축되는 성능향상을 가져올 수 있다.

8) 4.2.2절에 나오는 '변수 정의 영역'에서 이 값을 조정할 수 있다. 보다 자세한 설명은 4.3.4절을 참조하기 바란다.

9) 클라이언트 식별자, 수행 상태 등

5. 다른 사용자 C로부터의 요구가 SWeS 서버 A로 전달될 경우, 앞서 2번의 과정과 똑같은 동작이 사용자 C에 대해 일어난다.
6. 마지막으로 사용자 B로부터 요구가 SWeS 서버 A로 전달된 경우를 살펴보자. 현재 SWeS 서버 A, B는 각각 사용자 C, A에 대한 요구를 처리해 준 후 대기상태로 들어가 있다고 가정하자. 이 경우에도 SWeS 서버 A와 SRP와의 소켓 연결을 해제하고 새로운 소켓을 만들지 않는다. 기존에 존재하는 소켓을 이용하며, 사용자 B에 대한 세션도 SRP에 유지되고 있으므로 새로운 세션을 생성하지 않는다. SWeS 서버 A는 공유 메모리로부터 사용자 B에 대한 클라이언트 식별자를 얻어낸 후, 이미 존재하는 소켓을 이용하여 사용자 B에 대한 세션에 연결을 해 준다. 그리고 나서 이전에 사용자 B에 대해 수행했던 작업의 다음 부분부터 계속해서 수행된다.

4.3.4 부하 균형

SWeS에서는 질의에 대한 결과로 나오는 행의 개수를 지정해 줄 수 있다. 사용자가 질의에 대한 수행 결과의 크기가 매우 클 경우, 결과를 사용자에게 전달해 주는 과정이 상당한 시간이 걸리게 된다. 또한 SWeS 서버가 오랫동안 사용중의 상태로 머무는 문제를 갖는다. 예를 들어 최대로 동작하는 SWeS 서버의 개수가 3개이고, 각 서버가 각기 다른 3 명의 사용자의 요구를 처리해 주고 있다고 가정하자. 이 3 명의 사용자가 모두 수행 결과의 크기가 매우 큰 질의를 수행한다면, 그 이후로 들어온 사용자들은 SWeS 서버가 수행을 다 끝마치고 대기상태가 될 때까지 무한정 기다리고 있어야 한다. 더군다나 그렇게 해서 사용자에게 전달된 수행 결과는 그 크기가 매우 크기 때문에 사용자의 클라이언트 환경, 즉 웹 브라우저의 여러 페이지에 걸쳐 나타난다. 하지만 사용자는 우선 한 페이지를 살펴보고 난 후 다시 다음 페이지를 살펴보는 식의 동작을 취하게 될 것이다. 따라서 처음부터 결과를 페이지 단위로 나누어 처리한다면 보다 효율적일 수 있다.

처리해 주는 결과의 크기가 한 페이지 단위로 줄어들면, 사용자의 입장에서는 결과를 얻는데 걸리는 시간이 단축될 수 있다. 데이타베이스에서 질의에 대한 결과를 얻는 시간이 줄어들기 때문이다. 아울러 한 사용자가 자신이 전송 받은 결과를 살펴보는 동안 SWeS 서버는 다른 사용자의 요구를 들어줄 수 있기 때문에, 동시에 더 많은 수의 사용자의 요구를 들어줄 수 있다. 사용자는 전송된 결과를 살펴본 다음, SWeS 서버에 새로운 요구를 전달함으로써 다음 페이지의 결과를 받아볼 수 있다.

이러한 페이지 단위의 전송은 스크립트 화일의 '변수정의 영역'에서 OUTROW 값을 통해서 지정해 준다(4.2.2절 참조). 응용 프로그램 개발자는 자신의 응용 프로그램 특성에 맞추어서 적절한 페이지 크기를 지정해 준다. 여기서 지정된 값은 전송되는 결과 테이블의 행의 개수를 말한다. 예를 들어 총 결과 테이블이 100행이고 OUTROW값이 10이라면, 한 페이지로 10행씩 전송된다. 결과의 마지막 행이 들어있는 페이지가 전송되기 전에는 결과 화면에 '다음 페이지'에 대한 요구를 할 수 있는 버튼이 자동적으로 생성된다. 사용자는 이 버튼을 눌러서 다음 페이지를 요구할 수 있다. 그림 6은 그림 4의 스크립트 화일을 실행한 화면으로, 수행 결과의 맨 첫 페이지를 보여주고 있다. OUTROW값이 10으로 설정되어 있으므로, 첫 10 행이 출력된다.

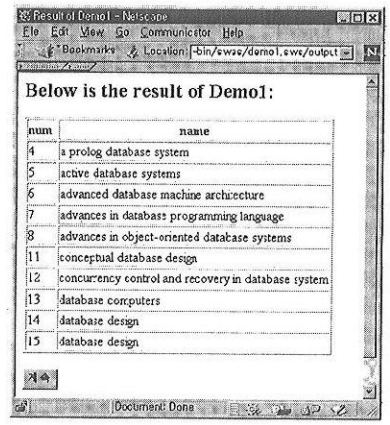


그림 6 SWeS의 수행 화면

SWeS 서버는 사용자의 요구를 처리하기에 앞서, 스크립트 화일의 OUTROW 값을 살펴보고, 여기서 지정된 개수만큼의 처리를 수행한 후 대기상태로 들어가 다른 사용자의 요구를 기다린다. 이 때 어디까지 수행했는가에 대한 정보를 공유 메모리에 기록해 둔다. 이후 다시 그 사용자로부터 다음 페이지에 대한 요구가 들어올 경우, 그 사용자에게 할당된 SWeS 서버가 공유 메모리로부터 그 사용자에 대한 정보를 읽어 들여 계속해서 다음 부분을 처리한다.

4.3.5 사용자 상태의 해제

현재 버전의 SWeS에서 사용자의 상태가 유지되는 경우는 결과를 페이지 단위로 전송하는 경우이다. 즉,

한 스크립트 화일에 대해서만 사용자의 상태를 유지한다. 따라서 결과의 마지막 행이 들어있는 페이지를 전송한 후, 공유메모리에 유지하던 사용자에 대한 정보를 삭제한다. 아울러 SRP 서버에서 유지하던 이 사용자에 대한 세션도 함께 삭제한다. 이후 그 사용자로부터 다시 요구가 들어오더라도 새로운 사용자로 인식되어 동작한다.

이렇게 사용자의 상태를 관리하는 경우, 여러 스크립트 화일에 걸쳐서 사용되는 특정 사용자의 상태나 설정된 변수 값 등은 유지할 수 없다. 현재 SWeS에서는 변수 값들을 다음 스크립트 화일을 부를 때 인자 값으로 넘겨주어 해결한다. 하지만 사용자의 세션은 각 스크립트 별로 SRP 서버에 생성되었다가 삭제되었다 하는 일이 반복된다. 다음 버전에서는 이 부분에 대한 추가적인 지원을 계획하고 있다.

3.3절에서도 지적했듯이 사용자가 페이지 단위로 결과를 보다가, 갑자기 다른 웹 페이지로 이동해버리는 경우에 대한 해결책은 현재 지원하고 있지 않다. 따라서 이 경우에는 특정 시간동안 사용자로부터 연결이 없으면 유지하고 있는 사용자의 정보를 삭제하는 시간제한의 방법으로 해결한다.

4.4 SWeS의 평가

4.4.1 SWeS의 평가

SWeS 웹 통로는 지금까지 앞에서 살펴본 구조를 통하여 기존의 CGI 실행화일 방식으로 구현된 웹 통로들(2.1절 참조)에 비해 성능 향상을 가져올 수 있다. 뿐만 아니라 4.1.1절에서도 언급했듯이 한 사용자가 특정 서버를 독점하지 못하게 함으로써, 비슷한 방식인 CGI 응용서버 방식으로 구현된 다른 웹 통로들(2.2절, 2.3절 참조)에 비해 다음의 장점을 갖는다.

확장성(scalability)및 동시성(concurrency) 이는 4.3.4절에서 설명한 부하 조절방식으로 얻을 수 있다. 즉 한 SWeS 서버가 사용자의 요구를 처리해 주는 시간이 일정해 짐으로써, 특정 사용자의 독점 사용을 막고, 동시에 처리해 줄 수 있는 사용자의 수를 늘릴 수 있다. 따라서 SWeS는 대규모의 사용자 수를 가지는 웹 환경에 적합한 구조를 지닌다.

가용성(availability) SWeS 서버는 한 사용자의 요구를 전담해서 처리해 주지 않는다. 즉, 한 사용자가 요구에 따라 여러 SWeS 서버에 연결될 수 있다. 요구가 들어올 당시 대기상태에 있는 서버가 사용자의 요구를 처리해 주기 때문이다. 따라서 어떤 SWeS 서버에 장애가 일어났다 하더라도 사용자에게는 아무런 영향을 주지 않고 다른 서버에서 계속 요구를 처리해 줄 수 있다(4.3절 참조).

유연성(flexibility) SWeS에서 사용하는 스크립트 형식은 추가의 프로그래밍 언어에 대한 지식을 필요로 하지 않는다. 기존의 HTML과 SQL만을 사용하여 응용 프로그램을 작성할 수 있다. 또한 HTML과 SQL에 새로운 기능이 추가되더라도 응용 프로그램의 변경을 요구하지 않는 장점을 갖는다(4.2.1절 참조).

성능(performance) SWeS는 기본적으로 CGI 응용서버 방식을 사용하기 때문에 여러 사용자로부터 요구가 들어와도 시스템에 많은 부하를 주지 않고, 연속되는 사용자의 요구를 효율적으로 처리해 주는 등의 장점을 갖는다(3.2절 참조). 또한 SWeS에서는 페이지 단위의 결과 전송을 지원하며, 이를 통해 수행 결과의 크기가 큰 경우에도 사용자에게 빠르게 결과를 보여줄 수 있다.

반면, SWeS에서 구현한 여러 SWeS 서버가 한 사용자의 요구를 처리하는 구조는, 특정 서버가 한 사용자의 요구를 전담하여 처리하는 구조에 비해 사용자의 상태 정보를 유지하는 것이 더 힘들다는 단점이 있다. 이러한 문제로 인하여, 현재 SWeS에서는 여러 스크립트 화일에 걸치는 한 사용자의 상태를 유지할 수 없다(4.3.5절 참조).

또한 최악의 경우 매번 사용자의 요청마다 다른 SWeS 서버가 할당되고, 이로 인하여 할당된 SWeS 서버는 매번 다시 SRP 서버에 유지되는 사용자의 세션에 연결해야 하는 추가적인 부하를 가질 수 있다는 단점을 갖는다. 현재 SWeS에서는 4.1.1절의 끝 부분에서 언급했듯이, 연속되는 사용자의 요구를 한 서버가 처리해 줄 수도 있도록 함으로써 최적화 된 동작을 지원한다.

4.4.2 성능 평가 결과

이 절에서는 기존의 CGI 응용서버 방식으로 구현한 통로와 SWeS와의 간단한 성능 평가 결과를 제시한다. 본 결과에서 사용한 클라이언트는 실제 웹 환경이 아니라 응용서버와 동일한 기계에서 동작하도록 하여 실험하였다. 응용으로는 각 사용자마다 총 네 번의 연속된 검색 요구를 수행하도록 하였고, 동시에 여러 사용자가 요구를 하는 경우를 사용자 수를 늘려가며 실험하였다. 또한 웹 환경을 고려하여 한 사용자가 요구에 대한 결과를 받은 후, 일정 시간 후에 다음 요구를 한다고 가정하였다. 따라서 사용자는 결과를 받은 5초 후에 다음 요구를 하도록 설정하였다. 그리고 동작하는 응용서버의 최대 수는 3개로 제한하였다.

그림 7은 사용자 3명의 요구가 동시에 수행되는 경우를 보여준다. 그래프에서 알 수 있듯이 기존의 CGI 응용서버 방식과 SWeS의 방식과 별 차이가 없다. 이는

동작하는 응용서버의 수가 사용자의 수와 같기 때문에, 두 방식이 동일하게 동작하기 때문이다. 반면 그림 8에서는 기존 방식의 경우 특정 사용자의 총 수행시간이 현저히 떨어짐을 보여준다(수행 시간 표준 편차: 9.44). 이는 동시에 요구를 하는 사용자는 모두 5명인데 반해 응용서버의 수는 3개인데 기인한다. 따라서 기존 CGI 응용서버 방식의 경우에는 응용서버를 차지하고 있는 3명 중 어느 하나의 수행이 모두 끝난 이후에야 나머지 2명이 비로소 서비스를 받을 수 있게 되기 때문이다. 그러나 SWeS 방식에서는 동적 서버 교체를 사용하기 때문에, 모든 사용자가 비교적 공평한 서비스를 받을 수 있다(수행 시간 표준 편차: 0.62). 다만 서버 교체로 인한 부하로, 수행 시간이 기존 방식에 비해 조금 더 오래 걸린다. 하지만 이는 4.4.1절에서 언급한 바와 같은 장점들로 인하여 상쇄될 수 있다. 그림 9에서 보면 사용자의 숫자가 늘어날수록 기존 방식의 불균형은 더욱 심해짐을 알 수 있다(수행 시간 표준 편차: 21.24). 반면 SweS 방식의 경우는 사용자의 숫자가 늘어나도 모든 사용자에게 비교적 일정한 수행 시간을 제공할 수 있는 장점을 보여준다(수행 시간 표준 편차: 1.24).

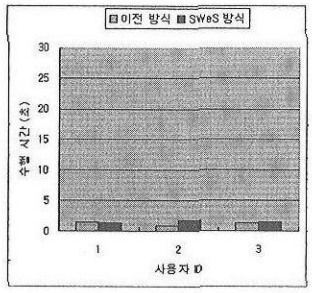


그림 7 성능 평가 1

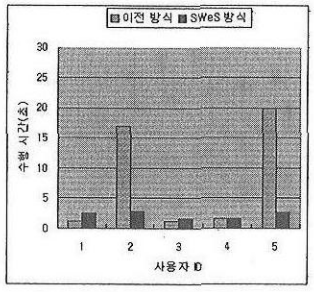


그림 8 성능 평가 2

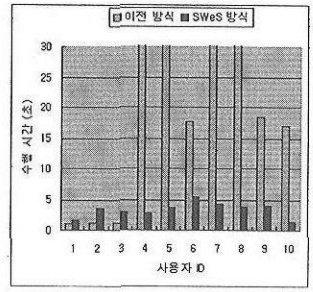


그림 9 성능 평가 3

5. 결론 및 향후 연구 계획

본 논문에서는 효율적인 웹용 데이타베이스 통로의 구현을 위해 현재 나와있는 여러 데이타베이스 통로들을 살펴보고, 웹용 데이타베이스 통로가 지원해 주어야 하는 고려사항 들을 제시하였다. 이를 토대로 하여 관계형 DBMS인 SRP와 웹을 연결해주는 데이타베이스 통로인 SWeS를 구현하였다. CGI 응용서버 방식으로 구현한 SWeS는 여러 서버가 사용자들의 정보를 공유하는 동적 서버 교체 방식을 사용한다. 이를 통해 서버의 가용성을 높여줄 수 있다. 또한 페이지 단위의 결과 전송을 지원하여, 동시 사용자 지원 및 부하 균형을 이루는 장점을 갖는다.

향후 SWeS에서는 여러 스크립트에 걸친 사용자의 상태 및 변수 값을 유지할 수 있도록 확장하고, 트랜잭션 지원에 관한 부분을 추가할 계획이다. 또한 사용자의 상태 해제에 대한 효율적인 해결책을 제시하며, 본 연구실에서 개발한 객체지향 DBMS인 SOP와의 연결 등을 지원하도록 확장할 계획이다.

감사의 글

본 연구를 수행하는데 많은 도움을 준 서울대학교 컴퓨터공학과 객체지향시스템 연구실의 SRP팀 여러분들께 감사의 말씀을 드린다.

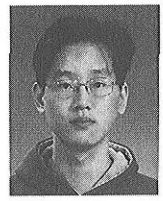
참 고 문 헌

[1] 김평철, 민영훈, "윌드와이드웹용 데이타베이스 통로의 성능 평가", 데이타베이스 연구회지, 제13권, 제2호, pp. 20-39, 1997.

[2] 김평철, "Uniweb 2.0 - 웹을 이용한 클라이언트-서버 데이타베이스 응용 개발 환경", 데이타베이스 저널, 제3권, 제2호, pp. 119-132, 1996.

[3] M. BJORN, "An Interactive Relational Database Gateway with Load Balancing", In AUUG95 & Assia Pacific WWW conference, AUUG95 & APWWW96, 1995.
[4] D.Eichmann, T.McGregor, and D.Danley, "Integrating Structured Databases Into the Web: The MORE System", In The First International Conference on the WWW, CERN, 1994.
[5] P.-C. Kim, "A Taxonomy on the Architecture of Database Gateways for the Web", In Proc. of the 13th ICAST and 2nd ICMIS, pp. 226-232, April 1997.
[6] D.Kristol and L.Montulli, "HTTP State Management Mechanism", Internet RFC 2109, February 1997.
[7] J.Ng, "GSQL: A Mosaic-SQL gateway", NCSA, University of Illinois at Urbana-Champaign, http://www.ncsa.uiuc.edu/SDG/People/jason/pub/gsql/back/starthere.html
[8] T.Nguyen and V.Srinivasan, "Accessing Relational Databases from the World Wide Web", Proc. of the ACM SIGMOD Conf. on Management of Data, 1996.
[9] D.Raggett, "HTML 3.2 Reference Specification", W3C Recommendation, January 1997.
[10] Rasmussen, B.F., and B.Pirenne, "WDB - A Web Interface to SQL Databases", European Southern Observatory, http://service.software.ibm.com/pbin-usa-demos/getobj.pl?/demos-pdocs/wwwdb2dnld.html.
[11] J.Rowe, "Webmaster's Building Internet Databases Servers with CGI", New Riders Publishing, 1996.
[12] SNU OOPSLA Lab, "SRP RDBMS PLATFORM 1.1 Manual", Seoul National University, 1996.
[13] H.F, T.Berners-Lee, R.Fielding, "Hypertext Transfer Protocol - HTTP/1.0", Internet RFC 1945, May 1996.
[14] C.Varela and C.Hayes, "Zelig: Schema-Based Generation of Soft WWW Database Applications", In The First International Conference on the WWW, CERN, 1994.
[15] http://gdbdoc.gdb.org/letovsky/genera/genera.html.
[16] http://www.o2tech.fr.
[17] http://www.kcom.co.kr/products/uni/uni.html.
[18] http://www.microsoft.com/iis/learnaboutiis/ActiveServer/default.asp.
[19] http://www.oracle.com/products/asd/asdhome.html.
[20] http://www.powersoft.com/products/jaguar/index.html.

최 일 환
1996년 서울대학교 컴퓨터공학과 학사. 1998년 서울대학교 컴퓨터공학과 석사. 1998년 현재 서울대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 객체관계형 데이타베이스, 웹, 트랜잭션, 사용자 인터페이스



이 상 철
1994년 서울대학교 수학과 학사. 1996년 서울대학교 컴퓨터공학과 석사. 1996년 ~ 현재 서울대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 질의어 최적화, 객체관계형 데이타 베이스

김 형 주
1982년 2월 서울대학교 컴퓨터고학과 졸업. 1985년 8월 Unv. of Texas at Austin, 전자계산학 석사. 1988년 5월 Univ. of Texas at Austin, 전자계산학 박사. 1988년 5월 ~ 9월 Univ. of Texas at Austin. Post-Doc. 1988년 9월 ~ 1990년 12월 Georgia Institute of Technology, 부교수. 1991년 1월 ~ 현재 서울대학교 컴퓨터공학과 부교수. 관심분야는 객체지향 시스템, 사용자 인터페이스, 데이타베이스