신뢰성 향상과 안전한 웹 서비스를 위한 웹 서버 아키텍처 환경의 설계

김용태* · 정윤수** · 박길철***

A Design of Web Server Architecture Environment for Reliability Enhancement and Secure Web Services

Yong-Tae Kim* · Yoon-Su Jeong ** · Gil-Cheol Park***

이 논문은 2009년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2009-0074117)

요 약

기존의 웹 서버 아키텍처 설계에서는 웹 서비스의 신뢰성, 견고성과 안전성을 유지하기 위하여 데이터 암호화(encryption) 기법을 사용하였다. 그러나 데이터 암호화 기법의 사용은 웹 서버의 처리량(throughput) 감소와 평균 응답 시간을 증가시키면서, CPU 작업을 낭비하기기 때문에 웹 어플리케이션 서버의 성능에 부정적인 영향을 나타낸다. 또한 최근의 웹 어플리케이션들은 안전한 인터넷 통신을 위하여 보안과 안전성을 요구하고 있다. 그러므로 본 논문에서는 안전한 웹 서비스를 위하여 기존의 웹 서버에 새로운 웹 서비스 모듈들을 추가하고 쓰레드 풀(Thread pool)과 Non-blocking I/O를 이용하는 개선된 웹 서버를 제안하고, 안전한 웹 서비스 성능을 나타내기 위하여 신뢰성과 안전성을 제공한다. 그리고 본 논문에서 제안한 웹 서버 시스템의 안전성과 성능을 평가하기 위하여 기존의 전형적인 톰캣 기반의 웹 서버와 제안 시스템에 대하여 실험을 통해 안전성과 성능을 비교 평가한다.

ABSTRACT

In the existing design of web server architecture, data encryption technique is used to keep the reliability, stability, and safety of web service. But the use of data encryption technique wastes the work of cpu while decreasing throughput of web server and increasing average response time so that it shows negative effect on the capacity of web application server. Also, the latest web applications require security and safety for the safe internet communication. Therefore, this paper suggests the improved web server which uses thread pool and Non-blocking I/O adding new web service modules to the existing web server for the safe web service, provides reliability and safety to show the safe web service capacity. And we compare and evaluate the safety and capacity through experiment on the existing traditional Tomcat based web server and the proposed system to evaluate the safety and capacity of the proposed web server system

키워드

웹 서버, SSL 프로토콜, 아파치 톰캣, HTTP/S, 논-블록킹 I/O, 쓰레드 풀

Key word

Web Server, SSL Protocol, Apache Tomcat, HTTP/S, Non-Blocking I/O, Thread Pool

* 한남대학교 멀티미디어학부 강의전담 교수
** 충북대학교 전자계산학과(교신저자)
*** 한남대학교 멀티미디어학부 교수

접수일자 : 2009. 08. 10
심사완료일자 : 2009. 08. 25

Ⅰ. 서 론

오늘날 웹 서비스의 중요성이 부각되면서 웹에 대한 의존도가 증가하고, 많은 비즈니스 어플리케이션들은 상업적 목적을 위하여 다양한 웹 서비스 들을 개발하고 있다. 그리고 웹 서비스를 이용하는 사용자도 폭넓게 확대되면서 웹 서비스의 성능과 보안 문제가 중요한 이슈가 되고 있다. 그러나 시간과 공간을 초월하는 웹 서비스의 많은 장점에도 불구하고 웹 서비스는 트랜잭션 관리와 보안에 취약하므로 신뢰성과 안전성을 보장하는 안전한 통신이 요구되고 있다.

최근의 웹 서비스 어플리케이션은 안전한 서비스 처리를 위해 클라이언트에게 요구된 보안과 프라이버시 수준에 해당하는 안전한 통신 채널을 제공해야 한다. 인터넷 환경에서 안전한 통신 표준은 HTTP에 보완적으로 사용하는 SSL(Secure Socket Layer) 프로토콜을 이용한다. SSL 프로토콜은 웹 서버와 클라이언트에서 수행하는 각각의 어플리케이션 사이에서 프라이버시(privacy)와 신뢰성을 제공하고, 공개키와 개인키를 이용하여 서버와 클라이언트 사이의 메시지를 암호화한다[1].

그러나 웹 서비스에서 보안 메커니즘의 도입은 추가적인 비용 부담이 발생한다. 공개키 암호 메커니즘은 클라이언트의 서비스 요구를 서버가 수신하면 암호화 키 교환을 위해 초기 핸드쉐이크(handshake) 통신을 수행한다. 그러나 초기 핸드쉐이크 통신은 서버의 성능을 감소시키고 새로운 연결을 제한한다[2]. 그리고 대부분의 웹 서버 환경은 다중쓰레드 패러다임에 의해 설계되어 쓰레드가 클라이언트의 모든 요구를 체크하는 부담을 가지며, 서버와 연결 가능한 동시 클라이언트의 최대 수가 생성된 쓰레드의 수에 제한된다. 그리고 서버가 과부하 상태일 때 폐쇄 연결, 암호 핸드쉐이크(handshake) 수행의 수를 증가시키고, 서버의 용량을 현저하게 감소시켜 결과적으로 서버의 최대 처리량에 대한 중요한 영향을 준다. 따라서 본 논문에서는 웹 서버의 성능 향상과 신뢰성을 증가시키기 위하여 안전한 통신 프로토콜을 이용하여 안전한 웹 서버 아키텍처를 설계한다.

본 논문은 다음과 같이 구성한다. 2장은 웹 서비스, HTTP/S 프로토콜 그리고 웹 서비스 보안에 대하여 기술한다. 3장에서는 본 논문에서 제안한 안전성을 위한 웹 서버 아키텍처 환경의 설계에 대하여 기술하고, 4장은 제안한 웹 서버 시스템의 실행 환경과 실험 결과를 기존 시스템과 비교 평가한다. 마지막으로 5장에서 결론과 본 논문과 연관된 향후 연구에 대하여 기술한다.

Ⅱ. 관련 연구

2.1 SSL 프로토콜과 HTTP/S의 개요

HTTP/S(Secure HTTP)는 HTTP의 확장판으로 웹 환경에서 안전한 파일을 교환을 위해 각각의 HTTP/S 파일은 암호화되며, 전자서명을 포함하고, 또 다른 보안 프로토콜인 SSL(Secure Socket Layer)의 대안으로 사용한다[3]. HTTP/S와 SSL의 차이점은, HTTP/S는 사용자 증명을 위한 인증서를 클라이언트에서 전송이 가능하지만, SSL에서는 오직 서버만이 인증이 가능하다. HTTP/S는 단일 암호화 시스템을 사용하지 않지만, RSA 공개키/개인키 암호화 시스템은 지원한다. SSL은 TCP 계층보다 더 상위의 프로그램 계층에서 동작한다. HTTP/S는 HTTP 응용의 상위 계층에서 동작한다[4].

HTTP/S는 HTTP 어플리케이션 계층 하단에서 SSL을 사용한다. SSL 프로토콜은 인터넷 환경에서 프라이버시 통신을 제공하기 위하여 클라이언트/서버 어플리케이션이 메시지 위조, 변경 그리고 도청을 막기 위해 공개키의 결합과 개인키 암호 작성법 알고리즘과 디지털의 증명서(X.509)를 사용하여 설계되었다.

SSL 프로토콜은 안전한 통신을 위한 암호문 사용으로 연결에 필요한 컴퓨팅 시간을 증가시켜 서버 성능에 중요한 영향을 준다[5]. SSL 프로토콜은 SSL 핸드쉐이크와 SSL 레코드 프로토콜의 두 단계로 구분된다. SSL 핸드쉐이크는 SSL 수행 시간 대부분을 사용하면서 서버가 RSA처럼 공개키에 의해 클라이언트 인증을 허용하고, 클라이언트 및 서버가 빠른 암호화, 복호화를 위해 사용된 대칭적인 키의 생성에 협력하고, 클라이언트가 서버에 자신을 인증하는 것을 허용한다[6].

SSL 핸드쉐이크는 full SSL 핸드쉐이크와 재개된 SSL 핸드쉐이크로 구분된다. 종단간에 새로운 SSL 연결을

개설할 때는 완전한 교섭을 요구하는 새로운 SSL 세션과 full SSL 핸드쉐이크를 수행하므로 많은 계산 시간을 소비한다. 재개된 SSL 핸드쉐이크는 새로운 HTTP 연결 없이 기존의 SSL 연결을 사용한다. SSL 세션 ID의 재사용으로 재개된 SSL 핸드쉐이크의 수행 시간을 상당히 감소시킨다.

2.2 웹 서버의 구성 환경

일반적으로 웹 서버 설계는 다중 쓰레드 아키텍처를 기반으로 병행 처리 프로그래밍 모델을 사용한다. 초기의 아파치 웹 서버는 멀티프로세스 아키텍처를 기반으로 안정적이지만 프로세스 생성의 오버헤드로 클라이언트의 요구 처리가 느려진다.

그러므로 대안적으로 확장성과 유연함을 위하여 다중쓰레드 모델과 BIO(Blocking I/O) 모델의 웹 서버를 사용한다. 다중쓰레드 모델과 BIO 모델은 클라이언트의 요청 처리를 위하여 하나의 워커(worker) 쓰레드에 할당된다. 다중 쓰레드 모델은 쓰레드와 연결이 종료될 때까지 클라이언트와 연결이 유지되며 다수의 요청 증가에 유연하게 대처하는 확장성이 요구되는 사이트, 안정성과 오래된 소프트웨어와의 호환성이 필요한 사이트에 사용한다[7].

다중쓰레드 모델은 웹 서버 프로그래밍이 쉽고 쓰레드는 클라이언트 요구에 의해 연결을 담당하고 작업이 종료되면 연결이 종료된다. 다중쓰레드 모델은 실행 시간이 짧은 클라이언트 요청 연결에 적절하고 낮은 비활동 기간을 가진다. 다중쓰레드 웹 서버 아키텍처는 워커 쓰레드 풀(thread pool)과 쓰레드 수신자(acceptor)에 의해 구성되며, 쓰레드 수신자는 새로운 클라이언트의 연결을 수신하고, 연결 개설은 워커 쓰레드 풀에 존재하는 하나의 쓰레드에 할당하고 웹 클라이언트에 대응하는 모든 요구를 처리하는 책임이 있다. 또한 쓰레드를 지원하여 웹 서버의 확장성을 증가시키고, 여러 프로세스와 여러 쓰레드를 혼합하여 실행이 가능하다[8].

2.3 Non-blocking I/O와 쓰레드 풀 모델

기존의 웹 서버는 다수의 동시 접속이 발생하는 경우 쓰레드의 생성 시간과 쓰레드 개수의 증가로 인한 문맥 교환(context switching)의 오버헤드가 발생하므로 일정한 쓰레드 개수의 유지가 필요하다. 또한 자원의 비효율적인 사용과 연결 오류가 발생한다.

쓰레드 풀 모델은 자원 낭비와 연결 오류를 방지한다. 즉, 연결에 많은 비용이 집중되기 때문에 한번 접속으로 계속 사용한다. 쓰레드 풀 모델은 최적화된 쓰레드 개수에 의하여 많은 사용자들의 요구에 빠르게 응답하지만 항상 고정된 시스템 자원을 점유하여 사용되지 않는 자원을 다른 프로그램들이 사용할 수 없다는 단점이 존재한다[9]. 또한 미리 할당된 쓰레드가 많게 되면 자원 낭비가 발생하고, 반대로 적으면 서버의 병렬성이 저하, 응답 속도 감소 그리고 잦은 접속 실패가 나타난다.

또한 동시에 다수의 클라이언트들의 요구들에 의해 발생되는 데이터 트래픽의 처리를 위하여 Non-blocking I/O가 필요하다. Blocking I/O는 성능 및 확장성에 제한이 있다. 이와 대조적으로 Non-blocking I/O는 호출된 쓰레드의 지속성에 의해 미들웨어 솔루션들은 동시 사용자를 위해 다중 쓰레드를 수행하여 성능 및 확장성을 확보한다.

2.4 웹 서버의 구성을 위한 톰캣

톰캣(Tomcat)은 Apache에서 open-source로 개발된 서블릿(servlet) 컨테이너이며 JSP를 지원하는 웹 서버이다. 톰캣 Jasper 컴파일러는 JSP를 서블릿으로 컴파일 한다. 톰캣의 주요한 목표는 Sun 서블릿과 JSP 명세 그리고 서블릿 컨테이너 구현을 위한 것이다. 톰캣은 독립된 웹 서버 또는, 웹 서버 조력자로서 작업이 가능하다. 톰캣 서블릿 엔진은 Apache HTTP 서버나 다른 웹 서버와 함께 결합하여 사용된다. 초기에 개발한 톰캣은 속도와 트랜잭션 조작에 대한 최소의 요구들을 가지고 적당한 개발 환경에 대해 독립적으로 존재되는 것으로 인식했다. 톰캣은 고 소통량과 고유용성 환경에 독립적인 웹 서버로 사용된다[10].

톰캣은 다중쓰레드 아키텍처를 가지고 있고 연결 서비스 스키마를 따르고 주어진 시간에 1개의 쓰레드(HttpProcessor)는 포트를 청취하고 소켓 구조를 할당하는 서버로 새로 들어오는 연결을 접수하는 책임이 있다. 이 포인트로부터 클라이언트와 함께 지속적인 연결 개설을 통해 받아들여진 요구를 제공하는 HttpProcessor 쓰레드는 다른 HttpProcessor가 새로운 연결을 계속해서 접수하는 동안 책임을 진다. HttpProcessors는 쓰레드 생성 오버헤드를 회피하기 위해서 쓰레드 풀로부터 일반적

으로 선택된다.

Ⅲ. 웹 서버 아키텍처 설계

본 논문에서 제안하는 안전성을 위한 웹 서버 아키텍처는 다중쓰레드 모델과 NIO(Non- blocking I/O) 모델의 장점을 갖는다. 웹 서버 아키텍처의 오퍼레이션 모형은 워커 쓰레드에 클라이언트 요구의 배정과 다중쓰레드 모형은 프로세싱과 클라이언트 요구의 서비스에 사용된다.

3.1 제안 시스템의 설계

본 논문에서는 다수의 클라이언트를 통하여 동시에 웹 서비스를 요구하는 경우 각각의 서비스 요구의 해결을 위해 소요되는 쓰레드 생성 시간과 점유 자원의 증가 문제를 해결하기 위해 쓰레드 풀 모델을 이용한다. 그리고 서버에서 쓰레드의 재활성화로 인하여 접속 시간과 응답 시간이 증가하는 문제점 개선을 위하여 Non-blocking I/O를 이용하여 클라이언트에서 데이터 송수신시 쓰레드의 재활성화로 인한 시간과 자원 낭비 문제를 해결하여 서버 접속 속도와 응답 속도를 개선한다.

본 논문에서 제안한 웹 서비스 시스템 아키텍처의 구성 환경은 그림 1과 같다.

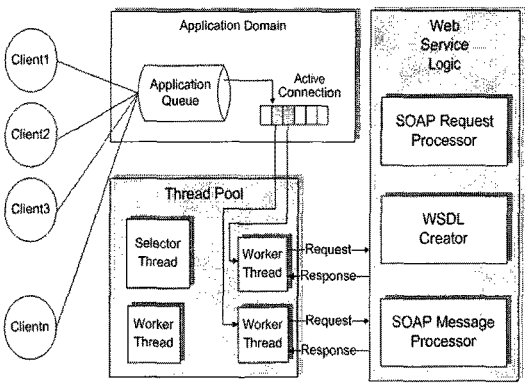

그림 1. 웹 서비스를 위한 시스템 구성 환경
Fig. 1 System configuration environment for Web service

본 논문의 제안 시스템에 추가되는 웹 서비스 모듈은 서버에 구현한다. 제안 시스템은 쓰레드 풀과 Non-blocking I/O 그리고 클라이언트의 웹 서비스 요구를 처리하는 웹 서비스 로직으로 구성한다.

3.2 웹 서비스를 위한 쓰레드 풀의 역할

제안시스템의 아키텍처 구현은 I/O 모델과 오리지널 쓰레드를 변화시켜 하나의 쓰레드는 NIO 셀렉터를 통해 클라이언트의 연결을 수신하고 등록하는 역할을 한다. 등록된 연결이 활동적인 경우는 데이터 읽기 작업을 위해 사용할 수 있기 때문에 소켓에서 차단되지 않고 HttpProcessor 쓰레드는 쓰레드 풀로 셀렉터 쓰레드에 의해 발송된다. 각각의 HttpProcessor는 자신을 위한 요구가 활동적인 경우 연결 할당을 서비스한다. 쓰레드는 블록킹(blocking)없이 요구를 처리하며 요구가 종료되면 연결은 셀렉터에 재등록되고 쓰레드는 새로운 연결이 할당될 때까지 쓰레드 풀에서 대기한다. 또한 모든 HttpProcessors가 수행중인 경우는 수락자 쓰레드가 새로운 연결을 받아들이지 않게 한다.

Non-blocking I/O는 쓰레드에서 데이터를 송수신하는데 쓰레드가 블록킹 되지 않고, 즉시 처리 가능한 작업(데이터 전송 또는 수신)만을 수행하고 나머지 작업을 할 수 있을 때까지 wait 상태로 기다린다. 작업 수행 가능 상태가 되면 메시지를 통해서 다시 작업을 수행하게 되므로 블록킹 되어 낭비되는 자원 없이 많은 네트워크 I/O의 처리가 가능하다. 또한 클라이언트의 요청에 대해 접속 시간과 응답 시간이 감소하여 다수의 클라이언트 요청에 원활한 처리가 가능하다.

Non-blocking I/O는 통신 유휴시간에 쓰레드가 사용되는 것을 방지하기 위한 것으로서 쓰레드는 읽기나 쓰기 때 blocking 되지 않는다. 또한 selector의 도입으로 클라이언트의 많은 접속을 처리할 서버의 부하가 상당히 줄어든다. 즉, 클라이언트가 서버로 보내는 데이터는 모두 non-blocking I/O를 사용하여 한 쓰레드에서 수신하기 때문에, 기존의 한 클라이언트 연결에 대해 하나의 쓰레드가 계속 사용되었던 방법 대신, 클라이언트가 요청한 하나의 작업에 대해서 쓰레드를 쓰레드 풀에서 가져와 사용하고 다시 쓰레드 풀로 반환하는 방법을 사용하였다. 클라이언트가 서버로 보낸 데이터를 받아오는 시간동안 쓰레드가 run 상태가 아니라 wait 상태일 수 있기 때문에 non-blocking I/O를 사용하지 않는 경

우보다 접속을 신속하게 할 수 있고 요구 대응성이 높아진다.

3.3 웹 서비스 로직 설계

본 논문에서 웹 서비스를 위한 웹 서비스 로직은 SOAP 요구 처리기, WSDL 생성기, SOAP 메시지 처리기로 구성한다. 본 논문의 제안 시스템이 웹서비스를 처리하기 위한 동작 절차는 다음과 같다.

1) 클라이언트는 웹 서비스를 위해 먼저 정의된 URL을 제안 시스템으로 전송하여 WSDL을 얻는다.
2) 클라이언트는 수신한 WSDL을 이용하여 SOAP 메시지를 생성하여 제안 시스템에 서비스 요구를 전송한다.
3) 제안 시스템은 클라이언트의 SOAP 메시지를 접수하고 수신한 SOAP 메시지에서 웹 서비스를 위해 필요한 WSDL명, 서비스명, 입력값들을 추출한다.
4) 추출한 값으로 웹 서비스를 수행한다.
5) 제안 시스템에서 수행한 웹 서비스 수행 결과를 WSDL 기반의 SOAP 메시지로 변환하여 클라이언트로 전송한다.

SOAP 요구 처리기(SOAP Request Processor)는 SOAP 메시지 분석기와 생성기로 구성한다. SOAP 메시지 분석기는 클라이언트의 요청 메시지 또는 웹 서버의 수신 메시지 여부를 분석하여 웹 서비스 요청 메시지인 경우는 클라이언트의 요청을 분석하여 클라이언트가 요청한 서비스의 WSDL명을 추출한 후 전달된 SOAP 메시지를 분석하여 정보를 생성한다. 분석한 정보에서 서비스명과 입력값들을 추출한 후 서비스 처리부에 서비스 처리를 요청한다. 그리고 웹 서버에서 서비스를 처리한 결과인 경우는 SOAP 메시지 처리기로 전달되어 응답 SOAP 메시지를 생성한다. 다음 그림 2는 웹 서비스 요청 처리 과정을 나타낸다.

WSDL 생성기(WSDL Creator)는 모바일 통신을 위해 HTML 데이터를 WSDL 데이터 형식으로 변환하여 WSDL 파일을 생성시킨다. WSDL 생성기는 XML 파서를 구현하여 WSDL 생성을 지원한다. 클라이언트가 HTTP를 통해 SOAP 메시지를 보내고 웹 서비스를 요구하면, 서버는 클라이언트 연결을 허락한다.

```
Procedure SOAP_Message_Aalysis()
{
 request analysis;
 an extract of WSDL name;
 creation of SOAP message information
 an extract of service name;
 an extract of input value
 request of service transaction to
            SOAP_Message_Creation;
 call SOAP_Message_Creation();
}

Procedure SOAP_Message_Creation()
{
 creation of response header information
 creation of SOAP message information

 If (transaction result = normal) then
  {
   an addition of service result value
   an addition of session ID;
  }
 else
   an addition of a failure information
 creation of a result SOAP message;
 call Connection()
 transmission transaction result to Connection
}
```

그림 2. 웹 서비스의 요청 처리 과정
Fig. 2 A Request Process of Web service

WSDL 생성기는 먼저 요청 내용을 분석하여 클라이언트가 원하는 WSDL명을 추출한 다음, HTTP를 사용하므로 헤더 정보가 필요하기 때문에 응답 헤더 정보를 생성하고 추출한 WSDL명에 대한 WSDL 정보를 획득한다. SOAP 메시지로 사용할 XML 문서 정보를 생성하고, 추출한 WSDL의 정보 즉, 각 서비스와 바인딩에 대한 정보를 WSDL에 추가하고, 출력형에 대한 complexType(출력형인 문자열, 문자열 배열, 문자열 2차원 배열에 대한 정보)도 추가한다. WSDL을 생성하고 헤더 정보를 추가하여 결과를 생성하고 접속부에 전달하면 접속부에서는 결과를 클라이언트로 전달하고 클라이언트가 결과를 수신하면 WSDL 요청을 완료한다.

SOAP 처리 과정은 웹서비스에게 요구를 보내기 전에 웹서비스 요구를 처리한다. 또한 클라이언트에게 보내기 위하여 SOAP 형식에서 응답을 변형시킨다.

SOAP 메시지 처리기(SOAP Message processor)는 SOAP 분석기, 서비스 프로세서와 SOAP 생성기로 구성된다. 클라이언트들의 요구로부터 서비스 이름과 입력 데이터를 SOAP 분석기가 추출하여 서비스 프로세서 모듈에 서비스 처리를 요구한다. 그 서비스 처리 모듈에서 돌아오는 결과는 SOAP 생성기 모듈로 보내어진다. 응답 헤더와 응답 메시지 부분을 새로 만든 뒤에, SOAP 생성기 모듈은 서비스 처리 결과가 성공적인지를 검사한다. 그 결과가 성공적이라면, 그 결과와 세션 ID가 더해진다. 아니면 정보는 붙여진다. 서비스 연결자는 클라이언트에게 응답 SOAP 메시지를 보냈다. 클라이언트가 SOAP 응답 메시지를 받으면, 웹 서비스 요구는 완료된다.

Non-blocking thread Manager는 통신 유휴시간에 쓰레드가 사용되는 것을 방지하기 위한 것으로 쓰레드는 읽기나 쓰기를 할 때 블록킹 되지 않는다. NBTM은 네트워크상에서 서버와 클라이언트의 연결 방법에 관한 것으로 쓰레드 서버 환경에서 클라이언트의 요청(접속, 실행)에 대해 사용된 쓰레드가 클라이언트의 요청에 대해 바로 실행할 수 있는 대기상태(wait)로 전환하여 클라이언트의 요청에 대한 서버의 접속 시간과 응답 시간을 향상하는 방법에 관한 것이다.

IV. 실험 환경 및 성능 평가

4.1 실험 환경 및 방법

본 논문의 실험을 위해서 기존의 어플리케이션 서버는 AIX와 톰캣을 사용하고 제안 시스템의 서버는 AIX와 추가된 웹 서비스 모듈을 사용한다. 본 논문에서는 지속적인 연결 타임아웃으로 3초 그리고 HttpProcessors의 최대 수를 100으로 설정하여 톰캣을 구성했다. 그리고 제안 시스템에서는 1개의 수락자/셀렉터 쓰레드와 10개의 HttpProcessors 쓰레드로 구성한다.

4.2 성능 평가

본 논문의 실험은 서버 성능의 테스트를 위하여 다수의 문자를 전송한다. 다수의 문자 전송은 초기 TCP, HTTP 연결을 설정하고 SOAP 메시지로 문자를 전송한다. 그림 3은 쓰레드 풀과 Non-blocking I/O를 사용하는 경우 클라이언트의 증가에 대한 응답 처리 개수를 측정한 결과이다. 제안 시스템의 처리량이 톰캣을 이용한 시스템보다 향상됨을 나타내며 부하가 증가할수록 실행 결과의 차이는 더욱 커진다. 톰캣 서버는 클라이언트의 증가에 의해 포화점에 도달하지만 제안 시스템은 56% 정도 처리율을 나타낸다.

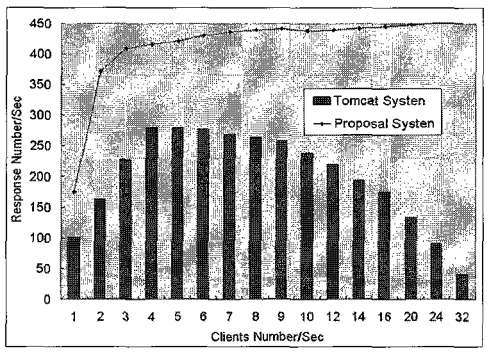

그림 3. Tomcat Systen과 Proposal Systen의 응답 개수 비교

Fig. 3 Response number comparison of Tomcat Systen and Proposal Systen

그림 4에서는 클라이언트 증가에 대한 응답 처리 시간을 나타낸다. 쓰레드를 사용하는 경우는 쓰레드 생성에 대한 오버헤드 때문에 쓰레드 풀과 Non-blocking I/O를 사용하는 경우가 전체적인 지연 시간이 단축됨을 나타낸다. 쓰레드 구현은 시스템이 과부하 상태일 때 다수의 동시 연결의 빠르게 증가하는 경우 새로운 연결을 수락자 쓰레드가 허락하기 때문에 응답 시간이 저하되는 중요한 결점을 나타낸다.

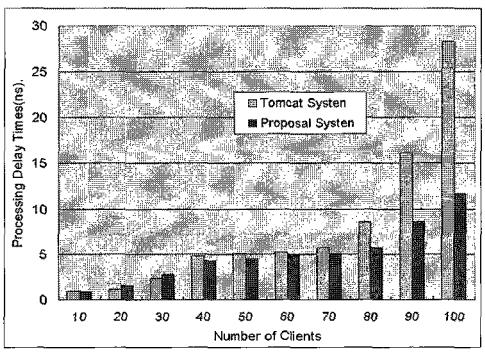

그림 4. Tomcat Systen과 Proposal Systen의 응답 처리 시간 비교

Fig. 4 Response Processing Time comparison of Tomcat Systen and Proposal Systen

그림 5에서는 톰캣을 사용한 서버와 제안 시스템의 세션 처리율을 각각 나타낸다. 두 가지 서버에서 각각 성공적으로 완료된 사용자 세션의 처리량을 측정한 결과 톰캣을 사용한 서버는 웹 서비스에 안전성을 도입하면 처리량이 급격히 감소되지만 제안 시스템의 경우는 클라이언트의 증가에 대한 세션 처리량 변화가 거의 없음을 나타낸다.

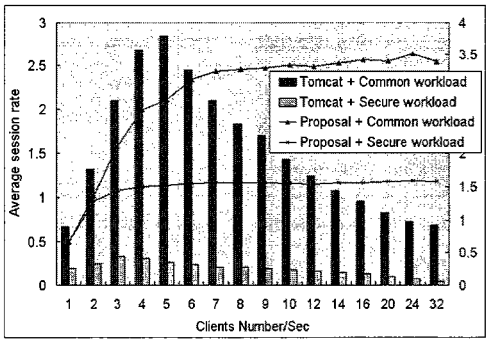

그림 5. 톰캣 서버와 제안 서버의 세션 처리율
Fig. 5 Session Throughput of Tomcat Server and Proposal Server

V. 결 론

본 논문에서는 새롭게 새로운 모바일 웹 서버 아키텍처를 제안하였다. 새롭게 제안한 웹 서버는 부가적인 모바일 웹서비스 모듈을 직접 구현하여 추가하였다. 추가되는 모듈들은 SOAP 요구 처리기, WSDL 생성기, SOAP 메시지 처리기 등이며, Non-blocking I/O와 쓰레드 풀, 그리고 세션에 근거한 표준 부하에서 그리고 암호화 기술을 사용할 때에도 웹 서버의 성능 향상이 가능하다.

또한 본 논문에서 제안한 웹 서버 아키텍처는 서버가 과부하 상태일 때 사용자 세션 처리와 서버의 서버의 처리량이 다중쓰레드를 사용하는 전형적인 톰캣 서버보다 향상되고 안전한 웹 어플리케이션 연결을 제공한다.

본 연구의 향후 작업 과정은 웹 서버 구현에 필수 요소인 워커 쓰레드 풀의 크기와 서버 용량과 관계있는 서버 타임아웃 그리고 서버에 연결되는 클라이언트와 같은 웹 컨테이너의 세부 조정에 대한 연구가 필요하다.

참고문헌

[1] Vicenç Beltran, David Carrera, Jordi Guitart, Jordi Torres, and Eduard Ayguadé, "A Hybrid Web Server Architecture for Secure e-Business Web Applications", LNCS, Springer, pp.366-377, 2005.10.

[2] Guitart, J.; Carrera, D.; Beltran, V.; Torres, J.; Ayguade, E, "Session-based adaptive overload control for secure dynamic Web applications", ICPP 2005. International Conference, pp. :341-349, 2005 .

[3] J. Guitart, V. Beltran, D. Carrera, J. Torres, and E. Ayguad´e. "Characterizing secure dynamic web applications scalability", In 19th International Parallel and Distributed Processing Symposium 2005, Denver, Colorado(USA), pp.4-8, 2005.

[4] Meadors, K, "Secure electronic data interchange over the Internet", Internet Computing, IEEE, Vol 9, pp. 82 - 89, 2005.

[5] 오현목, "유무선 통합 메시지 모델 연구", 최종 결과 보고서-NCA IV-RER-05013, 한국전산원, 2005. 12.

[6] Berbecaru, D, "On Measuring SSL-based Secure Data Transfer with Handheld Devices", Wireless Communication Systems, 2005. 2nd International Symposium 2005, pp. 409-413, 2005.

[7] Beltran, V.; Carrera, D.; Torres, J.; Ayguade, E, "Evaluating the scalability of Java event-driven Web servers", ICPP 2004. International Conference, vol. 1, pp. 134-142, 2004.

[8] Te-Kai Liu; Hui Shen; Kumaran, S, "A capacity sizing tool for a business process integration middleware", CEC 2004. Proceedings, pp. 195-202, 2004.

[9] DongHyun Kang; Saeyoung Han; SeoHee Yoo; Sungyong Park, "Prediction-Based Dynamic Thread Pool Scheme for Efficient Resource Usage", CIT Workshops 2008, pp. 159-164, 2008

[10] Ryan B Bloom, Apache Server 2.0: The Complete Reference, McGraw-Hill Osborne Media, 1st edition, 2002.

저자소개

김 용태(Yong-Tae Kim)

1984년 한남대학교 계산통계학과 학사
1988년 숭실대학교 전자계산학과 석사
2008년 충북대학교 전산학과 이학박사
2002년 ~ 2006년 가림정보기술 이사
2006년 3월 ~ 현재 한남대학교 멀티미디어 학부 강의 전담교수
※관심분야: 모바일 웹서비스, 정보보안, 센서 웹, 모바일 통신보안, 멀티미디어

정 윤수(Yoon-Su Jeong)

2000년 2월 : 충북대학교 대학원 전자계산학 이학석사
2008년 2월 : 충북대학교 대학원 전자계산학 이학박사
※관심분야: 센서 보안, 암호이론, 정보 보호, Network Security, 이동통신보안

박길철(Gil-Cheol Park)

1983년 한남대학교 계산통계학과 학사
1986년 숭실대학교 전자계산학과 석사
1998년 성균관대학교 정보공학과 박사.
2006년 UTAS, Australia 교환교수
1998년 8월 ~ 현재 한남대학교 멀티미디어학부 교수
※관심분야 : multimedia and mobile communication, network security