웹2.0에서 SOAP 처리와 성능 향상을 위한 모바일 웹 서버 프레임워크의 설계

김용태*, 정윤수**, 박길철***

A Design of Mobile Web Server Framework for SOAP Transaction and Performance Enhancement in Web2.0

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

본 연구는 지식경제부 지역혁신센터 사업인 민군겸용 보안공학 연구센터 지원으로 수행되었음

요 약

기존의 웹 서버는 과부하 상태인 경우 폐쇄 연결, 암호 핸드쉐이크 실행 증가와 서버 용량의 현저한 감소로 서버의 처리량에 문제가 발생하여 시스템의 전체적인 성능을 저하시킨다. 또한 쓰레드 블록으로 인하여 다수의 클라이언트 요청 처리가 원활하지 못하고, 쓰레드 활성화를 위하여 많은 시간과 자원을 요구하여, 클라이언트의 요청에 대해 접속 시간과 응답 시간이 증가하는 단점이 발생한다. 그러므로 본 논문에서는 웹서비스의 장점인 통합과 커뮤니케이션 지원과 시스템 성능 향상을 위해 서버의 과부하를 개선하고, 지연 처리를 위해 필요한 기술을 제공하는 확장된 웹서버를 제안한다. 그리고 기존 시스템(Tomcat 5.5에서 구현)과 제안한 모바일 웹 서버 아키텍처를 평가한다. 확장된 서버 아키텍처는 시스템 성능을 위해 우수한 교환 조건을 제공하고, 다중쓰레드와 쓰레드풀을 결합한 개선된 웹 서버 아키텍처를 평가한다. 본 논문에서 제안된 웹서비스 아키텍처는 오리지널 Tomcat 5.5의 평가 결과보다 개선된 성능 이익의 결과를 얻었다.

ABSTRACT

Existing web server lowers the whole capacity of system because of the problem on the processing load of server by closing connection, increasing code handshake operation, and remarkable decrease of server capacity if it is the state of overload. Also, there occurs disadvantages of increasing connection time about client's request and response time because handling of client's multi-requests is not smooth because of thread block and it requests a lot of time and resources for revitalization of thread. Therefore, this paper proposes the extended web server which provides the technique for delay handling and improves the overload of server for better system capacity, communication support, and the unification which is the advantage of web service. And it evaluates the existing system(implemented at Tomcat 5.5) and the proposed mobile web server architecture. The extended server architecture provides excellent exchange condition for system capacity and evaluates improved web server architecture which combines multi-thread with thread pool. The proposed web service architecture in this paper got the better result of improved capacity benefit than the evaluation result of original Tomcat 5.5.

키워드

Web Services, SOAP, Apache, WSDL, XML, Non-Blocking I/O, Thread Pool

* 한국해양대학교 공과대학 전파공학과

접수일자 2008. 08. 11

Ⅰ. 서 론

오늘날 웹은 모바일과 다양한 디바이스 환경에서 상호 통신하고 작용하는 동적인 서비스 기반 구조로 웹서비스를 이용하는 추세로 발전하면서 웹서비스에 대한 이용과 수요는 계속 증가하고 있다. 그리고 IT 분야의 발전으로 비즈니스 복잡성과 사용자 요구의 다양성에 유연하게 대처하는 시스템의 필요성이 증가하고 있다.

기존의 웹서버 아키텍처는 대부분 멀티쓰레드 패러다임 기반의 아파치와 톰캣(Tomcat) 기반으로 클라이언트에게 하나의 쓰레드를 할당하여 연결의 단절까지 모든 요구를 체크하는 부담이 있다. 특히 웹 서버가 과부하 상태인 경우는 서버의 용량을 현저하게 감소시켜 서버의 최대 처리량에 영향을 끼치므로 시스템의 전체적인 성능 저하 문제가 발생한다[1]. 그리고 기존의 서버와 클라이언트간의 통신은 입/출력 과정에서 쓰레드가 블록되는 경우가 발생하여, 다수의 클라이언트 요청 처리가 원활하지 못하고, 쓰레드 활성화를 위하여 많은 시간과 자원을 요구하여, 클라이언트의 요청에 대해 접속 시간과 응답 시간이 증가하는 단점이 발생한다[2].

그러므로 본 논문에서는 웹서비스의 장점인 통합과 커뮤니케이션 지원과 시스템 성능 향상을 위해 서버의 과부하를 개선하고, 지연 처리를 위해 필요한 기술을 제공하는 개선된 웹서버를 제안한다. 그리고 기존 시스템(Tomcat 5.5에서 구현)과 제안한 모바일 웹 서버 아키텍처를 평가한다.

본 연구의 성능 평가는 비교 대상의 웹서비스보다 매우 적은 클라이언트 지연과 더 높은 처리량을 얻었다. 기존의 웹서비스 기반 엔진은 처리량을 감소시키고 응답 시간을 예측하기 어렵다. 본 논문에서는 결과를 검증하기 위해 웹서비스에 대해서 CPU와 네트워크에 대한 폭넓고 다양한 유용성과 시나리오에 대해 강도 높은 검사를 실시한다.

본 논문은 다음과 같이 구성한다. 2장은 관련 연구로 웹서비스의 개요와 기존 웹서비스 시스템의 문제점을 기술하고, 3장에서는 본 논문에서 제안하는 웹서비스 시스템의 프레임워크 설계에 대하여 기술한다. 4장은 제안 시스템과 Apache-Axis를 사용하는 웹서비스의 시험 결과를 나타낸다. 마지막으로 5장에서 결론과 미래 작업의 내용을 기술한다.

Ⅱ. 관련 연구

2.1 웹 서비스의 개요

최근 XML 기반의 웹서비스 기술이 등장하면서 HTTP, SOAP(Simple Object Access Protocol), WSDL (Web Services Discription Language), UDDI(Universal Description, Discovery and Integration)등과 같은 공개 표준을 이용한다. 웹서비스는 특정 플랫폼에 독립적이며 서로 다른 벤더 간의 상호운용성 보장이 용이하여 최상의 SOA(Service Oriented Architecture)의 구현 기술로 인식되고 있다[3]. 따라서 기존 HTML 기반의 웹서비스와의 차이점은 XML 기반의 표준화된 언어와 프로토콜의 사용으로 사용자가 시간, 장소, 단말기에 무관하게 원하는 기능 및 서비스를 이용하는 유비쿼터스 환경의 주요 핵심 기술로 발전하고 있다.

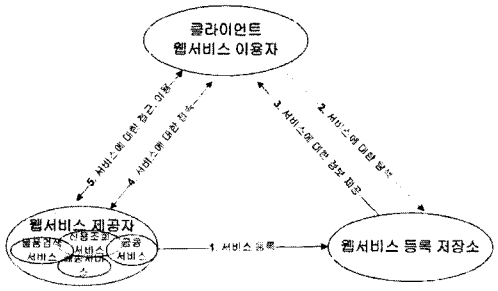

그림 1. 웹 서비스의 제어 구조
Fig. 1 Control Architecture of Web Services

XML은 웹서비스를 위한 가장 중요한 기술이고, 데이터의 정의, 공유, 교환에 사용하며, 웹서비스의 호출과 서비스 결과 등이 모두 XML 문서를 사용하여 전송한다[4]. SOAP은 XML 기반 프로토콜로 서로 다른 소프트웨어 컴포넌트들을 연결하는 프로토콜이다. XML과 HTTP에 의해 플랫폼과 관계없는 서버와 서비스의 접근이 가능하며, 웹서비스의 요청과 응답 메시지 형식을 정의한다[5]. WSDL은 특정 비즈니스가 제공하는 서비스를 기술하고, 웹서비스 이용에 필요한 인터페이스와 입/출력 메시지 형식을 기술하여 서비스에 대한 전자적인 접근 방법을 제공하는 XML 기반의 언어이다. UDDI는 웹서비스에 대한 디렉토리 서비스를 지원하는 분산 레지스트리 표준으로 웹서비스의 등록과 검색/바인딩을 위한 메커니즘을 제공하여 실시간으로 서비스의 조회

가 가능한 공용 디렉토리이다[6].

현재의 웹서비스는 이러한 세가지 기술을 기반으로 구현되어 있으며, 서비스를 위해 교환하는 메시지는 SOAP을 사용하고, 서비스의 바인딩과 호출을 위해 필요한 인터페이스 명세는 WSDL을 이용하고 모든 표준은 XML로 작성하며, 기존의 인터넷 환경에서 HTTP 프로토콜로 웹서비스에서 사용되는 XML 메시지를 전달하므로 적은 이식 비용과 플랫폼에 독립적이라는 특징을 가진다.

2.2 아파치(Apache) 웹 서비스

아파치 웹서버는 NCSA(National Center for Supercomputing Applications) httpd 1.3을 기반으로 개발된 웹서버다. httpd 서버는 트래픽 증가에 효과적이지 않고, 여러 웹 사이트 관리 문제가 발생한다. 아파치 1.3 버전까지 개발 목표는 안정성과 풍부한 기능성 제공이지만, 아파치 2.0 버전은 빠른 속도, 다양한 플랫폼 지원, 프로그래밍 인터페이스를 통한 견고한 모듈 개발을 목표로 한다[7].

초기의 아파치 1.3 버전까지는 멀티프로세스 아키텍처 기반의 프리포크 기법의 사용으로 안정적이지만, 프로세스를 생성과 스케줄링에 많은 오버헤드를 가지며 응답이 느리다. 아파치 2.0 버전은 멀티쓰레딩 기능의 추가로 기존의 한계인 확장성과 유연함을 갖는 하이브리드 웹서버를 표방한다. 또한 쓰레드 지원으로 웹서버의 확장성을 증가시키고, 프로세스와 쓰레드의 혼합 실행이 가능하다[8].

2.3 기존 웹서비스 시스템의 문제점

톰캣 AXIS에 의해 웹서비스를 쉽고 빠르게 구현하는 장점을 갖지만, 웹 서버 외에 톰캣을 설치하고 사용하는데 톰캣 서블릿 엔진을 사용하면, 웹서버 설치와 추가적인 인터넷 포트가 필요하고 처리 시간을 요구한다. 따라서 서버의 보안적인 측면과 관리적인 측면에서 부가적인 추가 요소가 발생한다. 이러한 단점을 보안하기 위해 웹 서버에 부가적으로 웹서비스 모듈을 직접 구현하여 추가하는 작업이 필요하다.

사용자의 요청 처리를 위한 쓰레드의 실시간 생성, 삭제 작업으로 인하여 발생하는 과부하는 작업 처리를 지연시키며 또 다른 과부하를 발생시킨다. 따라서 요청에 대한 적은 지연과, [3]에서와 같이 쓰레드의 과부하를 위한 다양한 해결 방법 중에서 쓰레드풀이 가장 대표적이다. 하지만 동적인 쓰레드 생성 방식이 아니고 설정된 양만큼씩 생성하여 한계치를 초과하는 경우는 쓰래싱(thrashing)이 발생하여 성능이 감소한다. 그리고 쓰레드풀의 크기를 설정하면 쓰레드풀의 쓰레드가 시스템의 자원을 항상 점유하여 다른 프로그램들이 자원 사용을 못하는 단점이 존재하여 다수의 클라이언트가 동시에 접속하는 경우는 정상적인 동작이 불가능하다.

또한, 기존의 클라이언트와 서버간의 통신 방법은 블록킹 I/O를 사용하여, 서버와 클라이언트의 입/출력 과정에서 쓰레드가 블록되는 경우가 발생하여, 다수의 클라이언트의 요청을 원활히 처리하지 못한다. 블록킹되어 제외된 쓰레드는 클라이언트 요청으로 다시 활성화되어 대기 상태로 변경된다. 쓰레드 활성화(메모리에서 로딩하여 대기 상태로 변환) 단계는, 많은 시간과 자원을 요구하여, 클라이언트의 요청에 대해 접속 시간과 응답 시간이 길어지는 단점이 발생한다[9].

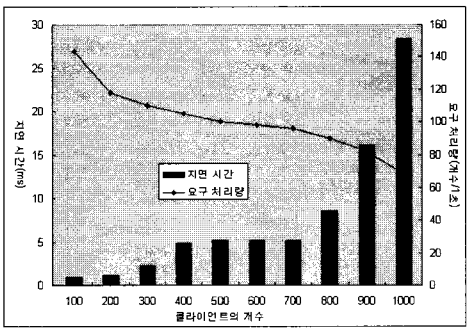

그림 2. 클라이언트의 평균 지연과 Axis 서버의 평균 처리량

Fig. 2 The average delay of the client and average throughput of Axis server

[10]에서 나타나는 것처럼 기존의 다양한 연구 결과는 아파치 웹서비스의 성능 결과가 다른 웹서비스보다 하락함을 나타낸다. 그림 2는 기존의 웹서비스 시스템에서 클라이언트의 평균 지연 시간과 Axis 서버의 평균 처리량을 나타낸다. 또 다른 연구 결과에서 나타난 것으로 클라이언트의 개수에 대한 평균 지연 시간과 Axis 서버의 평균 처리량을 나타낸다. Vinay Bansal and Piyush Shivam의 "SWSA: A Scalable Web Services Architecture"에서도 그림 2에서 나타난 것과 같이 아파치의 Axis 서버의 처리량은 부하를 증가시키면 떨어지기 시작한다. 이는 더 높은 부하에서 쓰레드 생성과 쓰레드 문맥 교환

비용을 지배하는 것에 기인한다[11].

Ⅲ. 웹서비스의 프레임워크 설계

3.1 제안 시스템의 설계

본 논문에서는 톰캣 AXIS를 사용하는 웹서비스의 클라이언트 요청에 대한 접속 시간과 응답 시간의 지연으로 인한 처리 지연과 요청 증가에 대한 처리량 감소의 단점을 보완하기 위해서 톰캣 서블릿 엔진을 제거하고 WSDL과 관련된 부분과 직접 SOAP 요구와 응답 메시지를 처리하는 모듈을 설계하고 쓰레드풀을 이용하는 논-블록킹 I/O 부분과 서버로 요청을 받아들이는 수신 모듈을 설계하고 구현한다.

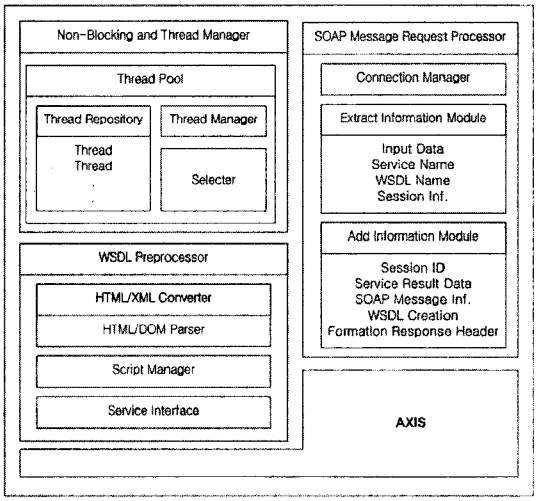

그림 3. 개선된 모바일 웹 서비스 시스템의 구조
Fig. 3 The improved structure of mobile web service system

본 논문에서 제안한 웹서비스 시스템의 프레임워크는 그림 3과 같다. 표준 시스템과 구현 시스템의 가장 중요한 차이는 구현 시스템에서는 톰캣 서블릿 엔진을 제거하고 WSDL 파일은 WSDL 전처리기에 의해 직접 생성하고 SOAP 메시지는 구현 시스템에서 처리한다. 추가되는 웹서비스 모듈은 서버에 구현하고, 메시지 조작, 메시지 체인, WSDL 처리에 대한 기능, 그리고 수신된 SOAP 요구 처리는 서버에서 처리하기 위해서 구현 시스템의 구성 요소에 다음과 같은 모듈을 구현한다.

- WSDL 전처리기(WSDL Preprocessor)
- SOAP 메시지 요구 처리기(SOAP Message Request Processor)
- NBTM 관리자(Non-Blocking and Thread Manager)

구현 시스템이 웹서비스를 처리하기 위한 동작 절차는 다음과 같다.

- 클라이언트는 지정된 URL로 WSDL을 얻는다.
- 클라이언트가 WSDL에 맞는 SOAP 메시지를 구현 시스템으로 전달한다.

구현 시스템에서는 클라이언트가 전송한 SOAP 메시지를 접수하여, SOAP 메시지에서 클라이언트가 원하는 WSDL명, 서비스명, 입력값을 추출한다.

- 추출한 값으로 서비스를 수행한다.

서비스 결과를 WSDL에 알맞게 SOAP 메시지로 변환하여 클라이언트로 전달한다.

3.2 3.1 WSDL 전처리기 설계

무선 웹서비스 통신을 위해서는 WSDL 언어가 필요하고, 기존의 웹에 존재하는 HTML 데이터를 무선 통신 데이터 형식인 WSDL 데이터 형식으로 변환하기 위하여 WSDL 전처리기를 설계한다. WSDL 전처리기는 웹서비스에 대한 요구를 WSDL 파일로 생성시키고 클라이언트의 요청에 의해 활성화한다.

본 논문에서 제안한 WSDL 전처리기는 HTML/XML 파서를 구현하여 WSDL 생성을 지원하고, HTML/XML 파서를 통하여 클라이언트로부터 요청된 메시지를 WSDL로 간단하게 변환한다. WSDL 전처리기는 Java 1.4의 org.w3c.dom 라이브러리를 이용하여 AXIS에 독립적으로 실행하며, Dom의 Document 객체를 사용하고, 모든 태그는 Element 객체를 사용한다. 그림 4는 클라이언트가 서버에 WSDL을 요청하는 과정을 나타낸다. 클라이언트가 HTTP를 통하여 서버에게 WSDL 형식을 http://host주소:port/webservice/ WSDL과 같은 URL 형태로 요청하면, 서버의 접속부에서 먼저 접속을 수락하고, 클라이언트가 전송한 요청의 종류를 분석하여 클라이언트의 요청 종류의 형태가 WSDL 생성에 대한 요청인 경우는 WSDL 전처리기에서 WSDL을 생성한다.

```
Procedure Client()
{
  call Server(SOAP_Message);
  Wait for return of transaction result from server ;
  the receipt of a transmission result;
  transmission WSDL request;
}
Procedure Server()
{ Procedure Connection()
   {  connection acceptance
      analysis of request a kind ;
      If (kind of request = request WSDL) then
      {  call WSDL_Creation();
         Wait for return result from WSDL_Creation;
         the receipt of a result;
         transmission to Client transaction result;  }
   }
   Procedure WSDL_Creation()
   { analysis request;
      an extract of WSDL name ;
      creation of response header information
      an extract of WSDL information ;
      creation of a XML document information
      addition of WSDL information
      addition of ComplexType;
      creation of a WSDL;
      creation of a result;
      call Connection()
      return of result to Connection   }
}
```

그림 4. 클라이언트의 WSDL 요청 처리 과정
Fig. 4 A WSDL request process of a client

WSDL 전처리기에서는 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 클라이언트가결과를 수신하면 WSDL 요청을 완료한다.

3.3 SOAP 메시지 요구 처리기 설계

SOAP 메시지 처리기는 클라이언트의 SOAP 메시지 형태의 요구를 분석하는 모듈과 SMWSS의 수신 결과를 규격화된 SOAP 메시지로 구현하는 모듈이 필요하다. 따라서 SOAP 메시지 처리기는 웹 서비스 요청을 수신하는 접속부와 SOAP 메시지 분석기 그리고 SOAP 메시지 생성기로 구성한다. 그림 5는 클라이언트가 서버에 요청하는 경우의 처리 과정을 나타낸다.

SOAP 메시지 분석기는 요청 서비스명, 입력값, 세션 아이디가 포함된 클라이언트의 SOAP 형태의 XML 문서인 SOAP 요청 메시지를 접속부가 SOAP 메시지 분석기로 전송하면 클라이언트의 요청 형태를 분석한다.

```
Procedure Server()
{ Procedure Connection()
   {  connection_acceptance
      analysis of request a kind;
      If (kind of request = request web service) then
      {call SOAP_Message_Analysis;
        Wait for result return SOAP message;
        transmission transaction result to Client;  }
   }
   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()
    transmissiontransaction result to Connection  }
}
```

그림 5. 클라이언트의 웹 서비스 요청 처리 과정
Fig. 5 A Web service request process of a client

요청 서비스의 WSDL명 추출 후 수신 SOAP 메시지의 분석 정보에서 서비스명과 입력값의 추출과 정보를 생성하여 서비스 처리부에 서비스 처리를 요청한다.

서비스 처리부의 처리 결과를 SOAP 메시지 생성부로 전송하면 응답 SOAP 메시지 생성을 위하여 응답 헤더 정보와 응답 SOAP 메시지의 정보를 생성하고, 정상적인 서비스 처리 결과는 서비스 결과값과 세션 아이디를 추가하고, 비정상인 경우는 실패 정보를 추가하여 SOAP 메시지를 생성하고 접속부로 전송하면, 접속부는 다시 클라이언트로 전송한다.

3.4 NBTM 관리자 설계

이 논문에서 NBTM 관리자는 다수의 클라이언트의 동시 접속을 원활하게 처리하기 위하여 사용자의 요구에 따라 쓰레드풀의 쓰레드를 동적으로 생성/삭제하는 쓰레드 풀 모델을 설계하고 구현한다. 동적인 쓰레드 생성/삭제를 위한 고려 사항은 다음과 같다.

- 쓰레드의 생성, 삭제로 인해 요청에 대한 응답 시간의 최소화한다.
- 쓰레드 풀의 크기를 자동으로 조절한다.
- 생성, 삭제에 관련된 과부하를 줄이고 응답 시간을 최소화한다.

NBTM 관리자는 일정수의 쓰레드를 미리 만들어 두고 필요한 상황에서 하나씩 사용하고 쓰레드를 제거하지 않고 대기 상태로 만들어 나중에 다시 사용하기 위하여 쓰레드풀에 다시 넣는다. 즉 쓰레드 생성/제거 오버헤드를 줄인다. 본 논문에서 사용된 쓰레드풀은 다음과 같은 동작한다.

- 서버가 구동할 때 지정한 개수만큼 쓰레드를 생성하며, 생성한 쓰레드는 관리를 위하여 쓰레드풀에 추가하고, 쓰레드는 무한 반복하며 수행 세팅된 명령을 수행하고, 자신을 풀에 다시 추가하고, 다음 명령이 세팅될 때까지 기다린다.
- 새로운 명령의 처리 요청은 사용가능한 쓰레드가 쓰레드풀에 존재하는 경우는 쓰레드를 사용하고, 존재하지 않으면 제한된 범위에서 새로운 쓰레드를 생성하여, 쓰레드에서 명령을 처리한다.
- 오랜 시간동안 사용치 않은 쓰레드는 지정한 타임아웃이 초과하면 제거하여 쓰레드를 관리한다.

클라이언트가 서버로 전송하는 데이터는 모두 논-블록킹 I/O를 사용한다. 논-블록킹 I/O는 쓰레드가 클라이언트에서 데이터를 수신하거나 클라이언트로 데이터를 전송할 때 블록되지 않고, 즉시 처리 가능한 작업(데이터를 보내거나 받는 작업)만을 하고 나머지 작업이 가능할 때까지 대기 상태로 한다. 작업이 가능한 상태에서 메시지를 통해 다시 작업을 수행하므로 블록되어 통신 유휴 시간에 쓰레드 사용을 방지하므로 자원을 낭비하지 않고 많은 네트워크 I/O를 처리한다.

기존의 하나의 클라이언트 연결에 대해 하나의 쓰레드가 계속 사용되었던 방법 대신, 클라이언트가 요청한 하나의 작업에 대해서 쓰레드풀에서 쓰레드를 가져와 사용하고 다시 쓰레드풀로 반환하는 방법을 사용한다. 클라이언트가 서버로 보낸 데이터를 수신하는 동안 쓰레드가 동작 상태가 아니라 대기 상태이므로 기존의 방법보다 더 효율적이다.

다음의 그림 6은 쓰레드풀에 전달한 클라이언트 요청을 처리하는 과정을 나타낸다. 쓰레드풀은 클라이언트의 명령 수행 요청을 수신하고 새로운 클라이언트 명령 수행이 가능한 쓰레드의 존재 여부를 확인하여 사용 가능한 쓰레드가 존재하는지 체크한다.

사용 가능한 쓰레드가 존재하지 않으면, 현재 쓰레드의 개수와 허용된 쓰레드 개수를 비교하여, 쓰레드의 추가 생성이 가능하면 허용된 쓰레드 개수를 초과하지 않는 일정한(delta) 값만큼 쓰레드를 새로 생성하고 쓰레드를 시작하여 쓰레드 저장소에 저장한다. 쓰레드가 부족할 때마다 하나씩 생성하면 자주 쓰레드를 생성하므로 한번에 일정한 값만큼씩 생성한다.

```
Procedure thread()
{  if( thread=termination)) then operation terminate
   else {  Wait for wake-up message
             instruction transaction;
             transmission of transaction result;
             restoration transaction thread to thread_pool;
             checking transaction time;
             call thread();  }
}
Procedure Server()
{  the receipt of request instruction transaction;
   if(useable thread=exit) then
         {  request thread to thread_pool;
            transmission of instruction to thread;
            transmission of thread start message;  }
   else if(current thread number < thread_pool size) then
         { while ((current thread number < thread_pool size) and
(              current thread number<max thread number+delta))
           {  creation of thread;
              start thread;
              store thread to thread_pool;  }
      request thread;
      transmission of instruction to thread;
      transmission of thread start message ;  }
   }
   else {  creation of transaction impossible message;
         transmission of transaction impossible message ;  }
}
```

그림 6. 쓰레드풀에서 새로운 명령 처리 과정
Fig. 6 New operation process in the thread pool

쓰레드의 추가 생성이 불가능하면 클라이언트의 요청 처리 불가능 메시지를 생성하고 클라이언트로 처리 불가능 메시지를 전달한다. 사용 가능한 쓰레드가 존재하면 쓰레드 저장소에 쓰레드를 요청하여 할당받은 쓰레드에 처리할 명령을 전달하고 해당 쓰레드에 신호 메시지를 전달하고 종료한다.

Ⅳ. 실험 환경 및 성능 평가

4.1 실험 환경 및 방법

본 논문의 실험 시스템의 성능 평가를 위해서 실험은 기존 방식을 적용하는 웹서비스 시스템과 본 논문의 구현 시스템을 각각 준비하였다. 또한 본 논문의 구현 시스템에서 사용하는 논-블록킹 I/O와 쓰레드풀을 함께 사용하는 경우에 대하여 비교 평가하기 위하여 테스트는 다양한 시스템 형태를 가지고 실시하였다. 여러 방법으로 서버를 구성한 뒤 200개의 클라이언트(20쓰레드*10프로세서)를 접속시켜 연결-리퀘스트-리브-연결-리퀘스트-…의 순서로 서버를 사용하였다. 각각의 버전을 위하여 구성한 서버의 형태는 다음의 [표 1]과 같다.

표 1. 실험을 위한 서버 구성 형태
Table. 1 A server configuration form for experiment

Apache 버전	클라이언트의 접속 처리 방법
Only 1.52.48	기존의 소스 적용
2.0 Only	ActiveQueue
2.0(ActiveQueue+쓰레드풀)	쓰레드풀에 의한 처리
2.0(ActiveQueue+NIO)	논-블록킹 I/O(NIO)에 의한 처리
2.0(Only 쓰레드풀)	쓰레드풀에 의한 처리
2.0(NIO+쓰레드풀)	NIO와 쓰레드풀에 의한 처리

또한 중계 프로그램을 사용한 경우에는 서버는 4개를 실행시켜 사용하였다. 각각의 버전에 대한 테스트 결과는 다음 [표 2]와 같다. [표 2]의 테스트 결과에서 나타나는 것처럼 중계 프로그램을 사용하는 것 보다 서버에 NIO와 쓰레드풀을 사용하였을 때의 결과가 연결 오류 횟수는 비교 대상 모두 0%이며 리퀘스트 오류 개수는 0.024%로 더 좋게 나왔다.

본 논문의 실험은 서버 성능의 테스트를 위하여 다수의 문자를 전송한다. 다수의 문자 전송은 초기 TCP, HTTP 연결을 설정하고 SOAP 메시지로 문자를 전송한다. 본 논문의 실험은 모바일 웹 서비스 시스템의 실행 평가에 초점을 맞췄다.

표 2. 각각의 버전에 대한 테스트 결과
(백분율=횟수/전체연결시도횟수)
Table. 2 The test results regarding each version

	1.52.48	2.0(AQ)	2.0(AQ+TP)
전체 시간	1시간 2분	1시간 9분	1시간 20분
연결 오류 횟수	0/25883/0/55968 (0%/46.25%/0%)	0/2924/0/38228 (0%/7.65%/0%)	40/2940/2452/38326 (0.1%/7.67%/6.40%)
리퀘스트 오류 횟수	5523/36149 (15.278%)	11032/37162 (29.686%)	9001/40480 (22.236%)
	2.0(AQ+NIO)	2.0(TP)	2.0(NIO+TP)
전체시간	1시간 12분	1시간	1시간 20분
연결 오류 횟수	13/11932/7051/52058 (0.02%/22.92%/13.54%)	0/2831/0/34752 (0%/8.15%/0%)	0/0/0/41138 (0%/0%/0%)
리퀘스트 오류 개수	5787/41539 (13.931%)	8722/35660 (24.459%)	10/40948 (0.024%)

* 연결오류: 타임아웃 횟수/refused횟수/핸드쉐이크 에러 횟수/전체 연결 시도 횟수
* 리퀘스트오류: 타임아웃 횟수/전체요청 횟수
* 타임아웃 횟수: 서버에 연결하였으나 서버가 일정시간 동안 응답이 없는 경우
* refused 횟수: 서버에 연결할 수 없는 경우
* 핸드쉐이크 에러 횟수: 서버에서 적절한 응답을 하지않은 경우

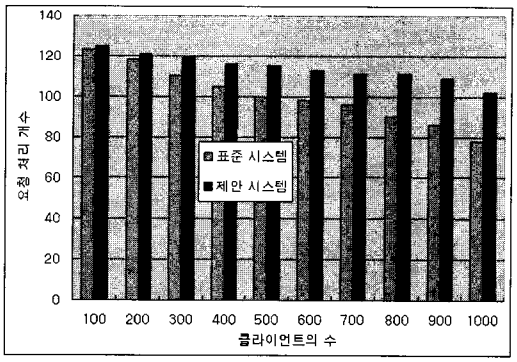

그림 7. 클라이언트의 초당 요청처리 개수
Fig. 7 Requests processed number of client per second

그림 7은 클라이언트의 초당 요청처리 개수를 나타낸다. 각각의 시스템의 요청 처리는 개선 시스템과 제안 시스템은 일정한 수준을 유지하지만 표준 시스템에서는 증가하는 클라이언트에 대하여 성능은 반대로 감소한다. 이것은 표 2에서 발생하는 처리 시간의 최종 결과에 기인함을 나타낸다.

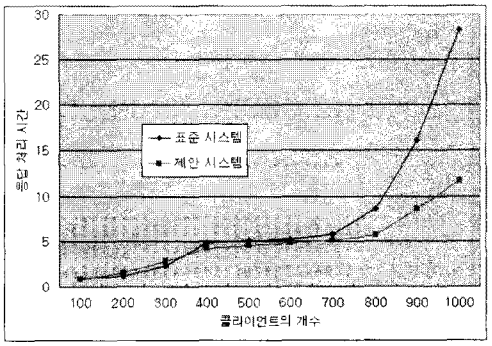

그림 8. 클라이언트의 개수에 대한 실험 시스템의 응답 지연 시간
Fig. 8 the delayed response time of the experimental system for the number of clients

그림 8은 클라이언트의 개수에 대한 실험 시스템의 응답 지연을 측정했다. 클라이언트의 개수를 증가시키면서 개선 시스템과 제안 시스템은 응답 시간에 대한 선형적인 증가를 나타낸다. 이것은 문맥 전환 비용의 감소에 의한 결과를 의미한다.

V. 결 론

본 논문은 웹 서비스의 성능 향상 문제와 지연 처리 해결이 중요한 이슈가 되고 있는 상황에서 모바일 웹 서비스의 성능 향상과 지연 처리 해결을 위하여 새로운 모바일 웹 서비스 아키텍처를 제안하였다.

본 논문에서는 논-블록킹 I/O와 쓰레드풀에 의한 웹 서버 아키텍처를 사용하여 시험했다. 특히 세션에 근거한 표준 workload 하에서 서버의 성능 향상을 가져왔다. 개선된 웹 서버는 모든 성능 파라미터에서 다중쓰레드를 사용하는 오리지널 톰캣보다 뛰어나고 웹 어플리케이션에 안전한 연결을 포함하는 오버헤드를 최소로 한다. 본 논문의 제안 시스템은 매우 작은 연결 오류를 가지고 있으므로 오리지널 웹 서비스 구현보다 효율적인 서비스 요구를 처리할 수 있다. 추후 작업은 웹서비스 아키텍처를 자율적인 서버의 구현에 맞추어 중요한 단계인 쓰레드풀의 크기, 서버 용량과 서버 타임아웃 등 웹 컨테이너의 세부 조정이 필요하다.

참고문헌

[1] 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, Denver, Colorado(USA). April 4-8, 2005, 2005.

[2] 전홍석, 이승원, 강현규, "아파치 웹 서버에서의 다중 쓰레드 풀 활용 기법 분석", 정보과학회논문지: 시스템 및 이론 제 32 권 제 1 호, pp.21-28, 2005.2.

[3] D. Schmidt , I. Pyarali, M. Spivak, and R. Cytron, "Evaluating and Optimizing Thread Pool Strategies for Real-T ime CORBA," A CM SIGP LANN otices, Vol 36, N o 8, pp. 214-222, August 2001.

[4] Eric Newcomer and Greg Lomow, Under standing SOA with Web Services, Sddison-Wesey pp. 51-197, 2005.

[5] Douglas B. Terry, Venugopalan Ramasubramanian, Caching XML Web services for mobility. ACM Queue −Tomorrow''s Computing Today, 1(3) 70 - 78, 2003.

[6] Robert Steele, "A Web Services-based System for Ad-hoc Mobile Application Integration", Proc. Of IEEE International Conference on Information Technology: Computers and Communications, pp. 248-252, 2003.

[7] The Apach Software Foundation, http://tomcat.apache.org/

[8] http://www.superuser.co.kr/apache/apache2_manual/new_features_2_0.html

[9] Annie P. Foong, Thomas R. Huff, Hervert H. Hum, Jaidev P. Patwardhan, and Greg J. Regnier. "TCP Performance Re-Visited", In Proc. Of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2003), pp. 70-79, Austin, Texas, March 2003.

[10] Dan Davis, Manish Parasher, "Latency Performance of SOAP Implementations", Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 407-412, 2002.

[11] Vinay Bansal and Piyush Shivam, "SWSA: A Scalable Web Services Architecture", Technical Report. Dept. of Computer Science, Duke University

저자소개

김 용태(Yong-Tae Kim)

1988년 숭실대학교 전자계산학과 석사.
2008년 충북대학교 전산학과 이학 박사
2002년-2006년 가림정보기술 이사
2006.3 - 현재 한남대학교 멀티미디어 학부 강의전담 교수
※ 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, 정보보호, Network Security, 이동통신보안

박 길 철(Gil-Cheol Park)

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