웹 응용 서버를 위한 효율적인 스케줄링 알고리즘

(An Efficient Scheduling Algorithm for The Web Application Server)

이 형 동 † 이 병 준 † 김 형 주 ††

(Hyung-Dong Lee) (Byung-Joon Lee) (Hyoung-Joo Kim)

요 약 웹의 성장은 점차 복잡한 응용에 대한 요구를 가중시켰으며, 데이타베이스와 웹과의 효율적인 연동 역시 중요7한 문제가 되었다.

본 논문에서는 이와 같은 환경을 효율적으로 지원하기 위한 웹 응용 서버 WATS를 설계하고 구현하였다. WATS에서 응용 프로그램은 컴포넌트 단위로 작성된 후 동적으로 링크되고 요청을 처리할 응용 서버가 대기 상태로 존재하는 확장 API 응용 서버 방식으로 구현되었으며, 이러한 웹 서버와 응용 서버의 분리 구조는 대량의 요청을 처리하기에 적합하다. 또한 컴포넌트를 특성에 따라 분류한 후 각 특성에 적합한 특성 기반 스케줄링 알고리즘을 적용하여 프로세스 부하 균형을 이루었으며, 이 기법이 일반적인 라운드 로빈 스케줄링 알고리즘보다 좋은 성능을 나타냄을 성능 측정을 통해서 보인다.

Abstract The increasing popularity of the World-Wide-Web (WWW) has resulted in demand for more complex applications, and web gateways to database became core component in such applications. In this paper, we have designed and implemented WATS in order to support these environments. In WATS, application components are dynamically linked with application server processes. And it is implemented as extensible API application server architecture and is able to process a large amount of requests through separating web server from application server. Also we classify the components into various categories according to its own properties and devise process load balancing algorithm by using property-based scheduling. We show WATS using this algorithm performs better than those using general round-robin algorithm.

1. 서 론

1.1 연구 배경

월드와이드웹(World-Wide Web, 이하 웹이라 함)은 스위스의 CERN에서 개발한 하이퍼미디어 방식의 대규모 정보 서비스 시스템으로서, 1992년 발표된 이후 인터넷을 중심으로 급속히 성장해왔다[1][2]. 웹은 쉬운 사용자 인터페이스와 광범위한 접속 때문에 그 사용자가 폭발적으로 증가하고 있고, 현재는 사용의 폭을 넓혀 상업적인 응용에도 많이 이용되고 있다. 하지만 이러한 사용자의 급증 및 다양한 분야(예를 들어, 가상 금융, 전자 상거래 등)로의 적용은 점차 복잡한 형태의 웹 응용 프로그램에 대한 요구를 가중시켰고, 과거의 정적인 서비스는 동적인 서비스로 바뀌게 되었다. 이에 대한 해결 방법으로 초창기에는 CGI(Common gateway interface)[3]가 대두되었는데, 이는 웹 서버로부터 여러 가지 입력 인자를 환경 변수로 받아, 프로그램을 수행한 후 그 결과를 HTML 페이지 형태로 가공하여 브라우저로 보내는 방식이다. 하지만 CGI 방식은 요청마다 새로운 프로세스를 생성해야 하는 성능 상의 문제점으로 인해, 위와 같은 다양한 요구들을 충족시키기에는 한계가 있었다. 또한 응용 개발자는 복잡하고 거대한 응용 프로

· 본 논문은 핵심 소프트웨어 기술개발 사업 객체지향기술을 이용한 인터넷상의 트랜잭션 처리기술 개발(100-115, 1997.10.1 1999 9 31)에 의해 지원 받았음

† 학생회원 : 서울대학교 컴퓨터공학과
hdlee@oopsla.snu.ac.kr
bjlee@papaya snu ac kr

†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@oopsla snu.ac.kr

논문접수 : 1998년 11월 3일
심사완료 : 1999년 5월 11일

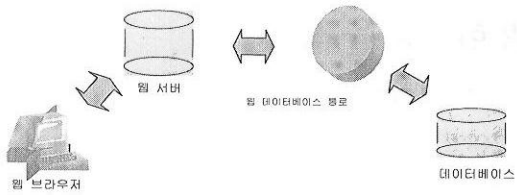

그림 1 웹 데이타베이스 통로

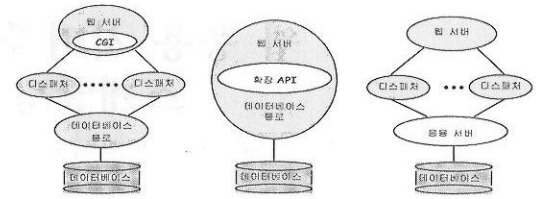

그림 2 웹 응용 서버 구조

그램을 개발할 때마다 처음부터 다시 작성해야 하는 비효율성도 가진다.

다른 한편 이와 같은 형태의 응용들로 인해 파일 시스템을 이용하던 초창기와 달리 대량의 자료에 대한 관리와 웹을 통한 이의 접근에 대해 관심을 갖게 되었고, 이는 웹과 데이타베이스와의 연동이라는 새로운 문제를 제기 시켰으며, 이에 대한 연구가 계속 되어왔다. 이러한 웹과 데이타베이스와의 연동에서 가장 중요한 역할을 하는 것은 <그림 1>이 나타내는 데이타베이스 통로(database gateway)[1]다. 데이타베이스 통로는 웹과 데이타베이스의 통합에서 핵심 요소로, 웹으로 표현된 사용자의 요구를 데이타베이스와 연동하여 처리한다.

이러한 배경에서 다양한 요구들을 충족시키기 위한 것이 바로 웹 응용 서버이며, 이는 다양하고 복잡한 응용들을 손쉽게 개발 할 수 있고, 이러한 응용들을 웹 환경 하에서 효율적으로 처리할 수 있어야 한다. 이와 같은 웹 응용 서버를 구현하는 방법에는 여러 가지가 존재하며, 가장 기초적인 방법으로는 이미 언급했던 CGI 방식, 이의 성능 문제를 개선한 CGI 응용 서버 방식, 그리고 웹 서버의 API(application programmer's interface)를 이용하는 확장 API 방식 등이 있다[1].

본 논문에서는 확장 API 응용 서버 방식을 이용하여 WATS(web application and transaction server)를 구현하였다. WATS는 웹에서의 트랜잭션 처리와 같은 새로운 형태의 응용들이 대량으로 요청될 때 이를 효율적으로 처리함을 목적으로 한다. WATS는 소프트웨어 재사용성을 위해 컴포넌트 단위로 응용 프로그램을 개발하므로 사용자는 웹 응용 개발이 용이하고 기존의 컴포넌트를 재사용하여 새로운 응용을 개발할 수도 있다. 또한 효율적인 스케줄링으로 대량의 요청에 대처할 수 있다.

1.2 논문의 구성

2절에서는 관련 연구로 현재 상용화 되어있는 웹 응용 서버들의 특징을 간단히 살펴보고, 웹에서 사용할 수 있는 성능 평가 도구들에 대해 알아본다. 3절에서는 우리가 설계 및 구현한 WATS(web application and transaction server)의 구조에 대해 살펴보고, 4절에서는 WATS의 부하 균형 알고리즘인 특성 기반 스케줄링에 대해 알아본다. 5절에서는 일반적인 라운드 로빈 방식의 스케줄링과의 성능 비교 결과 및 분석을 기술하며, 마지막으로 6절에서는 결론 및 향후 연구 계획에 대해 살펴보기로 한다.

2. 관련연구

2.1 웹 응용 서버

이번 절에서는 웹 응용 서버의 여러 구조들에 대해 간략히 알아본다. 이러한 웹 응용 서버에는 요청 도착 시 응용이 수행되는 방식에 따라 여러 형태를 갖는데, <그림 2>가 이러한 구조들을 보이고 있다. <그림 2>의 (A)는 CGI 응용 서버 방식을 나타내고 있다. 이는 기존의 CGI 실행 파일 구조에서 데이타베이스를 접근하는 등 실제 연산을 수행하는 프로세스를 데몬(daemon) 방식으로 바꾸고, 요청 시에는 작은 규모의 디스패처가 생성되어 처리되는 방식이다. <그림 2>의 (B)는 확장 API 방식으로, 웹 서버가 제공하는 확장 API[1]를 이용하여, 웹 응용을 작성하면, 수행 시 해당 응용이 동적으로 링크되어 하나의 프로세스로 수행된다. 이 방식은 초기에 웹 서버 업체에서 서버의 기능을 사용자가 확장시켜 사용할 수 있게 제공한 것으로 , 프로세스 관리 비용을 줄여 성능이 뛰어나 많이 사용되는 방식이다. 하지만 이 방식도 응용이 웹 서버의 프로세스로 수행되므로 서버의 부담으로 인해 대량의 요청을 처리하기에는 부적절하다. 이러한 문제점을 해결하기 위한 방식으로 응용 서버를 대기시키고, 디스패처가 이들을 연결하는 방식이 <그림 2>의 (C)에서 보여주는 확장 API 응용 서버 방식이다. 이 구조의 장점은 확장 API 방식의 장점

1) Netscape사의 NSAPI, Microsoft 사의 ISAPI가 대표적이다.

을 살리면서 웹 서버와 응용 서버를 분리함으로써 분산 처리가 가능하여 대량의 요청을 처리하는데 적합한 구조로 최근의 많은 웹 응용 서버들이 이 구조를 따르고 있다. 이러한 웹 응용 서버들의 실제 구현 예로 이번 절에서는 CGI 응용 서버 방식을 취하는 UniWeb[1]을 비롯하여, 확장 API 응용 서버 방식의 제품들을 살펴본다. 이에는 Oracle의 OAS(oracle application server)[4], Sybase의 Jaguar CTS[5]등 기존의 데이타베이스 업체들에서 자사의 데이타베이스와 웹과의 연동을 위해 내놓은 제품들과, Microsoft의 MTS(microsoft transaction server)[6] 등이 있다. 이와 같은 제품들은 단순히 웹 상에서의 복잡한 응용 개발에만 활용되는 것이 아니라, 웹 상에서의 트랜잭션 처리를 지원하기 위한 수단도 제공한다.

2.2 성능 평가 도구

현재 웹 응용 서버에 대한 성능 평가 도구는 없으며, 다만 웹 서버의 성능 평가 도구만이 존재한다. 웹 서버의 성능 측정 대상은 사용자가 요구한 문서에 대한 응답시간으로 볼 수 있으며, 이를 웹 응용 서버 성능 측정에 그대로 적용하기에는 무리가 따른다. 즉, 웹 서버 자체의 성능 및 데이타베이스의 성능은 데이타베이스 통로의 성능과는 무관하다. 따라서 웹 응용 서버의 성능을 측정하기 위해서는 웹 서버의 성능 평가에 질의 종류, 데이타베이스 연동 환경 등의 추가적인 요소가 필요하다[7][8]. 현재 웹 서버의 성능 평가 도구로는 WebSTONE[9], SPECweb96[10], Webpest[11] 등이 있다.

3. WATS의 설계 및 구현

3.1 구조

WATS는 기본적인 구조로 앞서 언급했던 확장 API 응용 서버 방식을 택하고 있다. 기본 요소는 대기자, 전달자, 패키지 프로세스(package process), 그리고 실제 응용 프로그램인 컴포넌트로 이루어진다.

● 대기자

대기자는 HTTP를 통한 URL 요청을 받아 WATS로의 요청 메시지를 분류하여 전달자에게 보내는 역할을 수행한다. 요청이 일반적인 정적인 HTML 파일에 대한 URL인 경우는 기존의 웹 서버와 동일한 작업[2]을 수행하며 URL에 특정 키워드(예를 들어, "http://coconut.snu.ac.kr/wts/hello/", hello 컴포넌트에 대한 호출)를 가지는 요청 메시지 메시지인 경우는 해당 메시지를

2) 정적인 HTML 문서에 대한 참조인 경우 해당 파일을 찾아 브라우저로 전송한다.

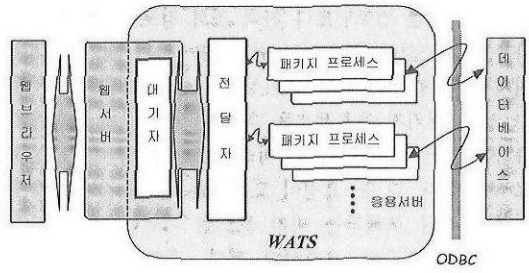

그림 3 WATS의 전체적인 구조

전달자에게 보내게 된다. 이러한 대기자의 응용 서버와의 분리를 통해 얻을 수 있는 장점은 웹 서버와 응용 서버를 분리시킴으로써 많은 요구에 대해 부하를 여러 응용 서버로 분배할 수 있고 기존의 어떠한 웹 서버를 사용하여도 무방하다. 즉, 대기자는 CGI와 유사하게 특정 URL에 대한 메시지들만을 선별하여 전달자에게 보내는 작업만 하므로 기존의 웹 서버에 확장 모듈을 추가하여 사용할 수 있다. 아파치 그룹의 Apache[12]나 Microsoft사의 IIS(internet information server)[6], Netscape사의 Livewire[13] 등의 웹 서버를 사용할 수 있으며, 본 시스템은 아파치(apache)의 확장 모듈 기능을 이용하여 구현하였다.

● 전달자

전달자는 대기자로부터 전해지는 요청을 해당 패키지 프로세스에게 전달하는 일을 하며, 적절한 패키지 프로세스를 선택하는 작업을 한다. 즉, 가장 적은 부하를 갖는 패키지 프로세스를 선택하는데, 이러한 프로세스 부하 균형에 대해서는 다음 절에서 자세히 다룬다.

● 패키지 프로세스

패키지 프로세스는 유사한 작업을 수행하는 컴포넌트들의 집합이다. 컴포넌트는 일반적인 동적 링크 라이브러리(DLL, dynamic link library)이며, 미리 지정된 인터페이스를 갖는다. <그림 4>와 같은 은행 업무 패키지

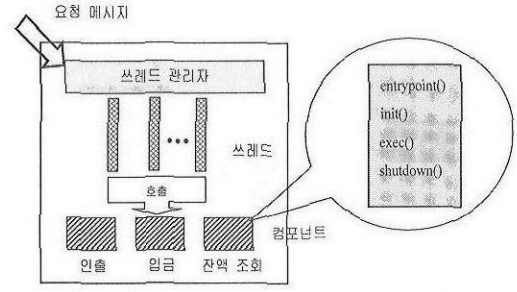

그림 4 은행 업무 패키지와 컴포넌트

인 경우, 인출 컴포넌트나 잔액 조회 컴포넌트처럼 같이 사용될 가능성이 높은 컴포넌트들을 같은 패키지 내에 위치시킴으로써 관리의 용이성 및 성능 향상을 가져올 수 있다. 패키지 프로세스는 초기화 작업 때에 자신에게 속하는 모든 컴포넌트들을 미리 로드시킨 후 대기 상태로 존재한다. 그 후 요청이 도착하면 새로운 쓰레드를생성하고 해당 쓰레드는 미리 로드된 적절한 컴포넌트를 찾아 해당 함수를 호출하게 된다. 패키지 프로세스는 WATS 운영자가 초기에 정해준 숫자만큼 시스템 시작 시 생성되어 대기 상태로 존재하며, 부하가 많을 경우 역시 운영자가 최대로 정해 놓은 개수 내에서 동적으로 새로운 패키지 프로세스를 생성하게 된다. 사용자로부터 요청 수행 시 이미 생성되어 대기상태로 있는 프로세스가 수행하게 되므로 CGI와 같이 요청마다 프로세스가 생성되어야 하는 부담이 없게 된다.

● 컴포넌트(component)

WATS에서 컴포넌트는 앞서 설명했듯이 일반적인 동적 링크 라이브러리며, 응용 프로그램 개발의 기본 단위가 된다. 응용 개발자는 미리 정해져 있는 세 가지 종류의 인터페이스를 구현하여 동적 링크 라이브러리의 형태로 등록시켜주면 된다(<그림 4> 참조, entrypoint()는 세가지 인터페이스의 포인터를 얻어 오는데 사용된다). 세 가지 인터페이스는 init(), exec(), shutdown() 이며, 각각은 초기화 작업, 실제 작업, 마무리 작업 등을 행한다. 하나의 응용은 여러 개의 컴포넌트들로 이루어질 수 있으며, 하나의 컴포넌트는 유사한 여러 응용에서 사용될 수 있으므로 재사용성을 높일 수 있다.

3.2 WATS의 동작 과정

3.2.1 전체적인 과정

사용자가 웹 브라우저를 통해 WATS 컴포넌트에 대한 호출을 요청하면, 대기자는 URL을 분석하여, 해당 요청이 WATS 컴포넌트에 대한 요청임을 인지하고 전달자에게 메시지를 보낸다. 전달자는 자신이 유지하고 있는 정보를 이용하여 해당 컴포넌트가 속하는 패키지를 알아낸 후 선택되어진 패키지 프로세스들에 대해 현재 상태 정보를 얻어낸다. 그 다음 해당 패키지 프로세스들 중에서 가장 부하가 적은 것을 선택하여 메시지를 전달한다. 메시지를 전달받은 패키지 프로세스는 새로운 쓰레드를 생성한 후 해당 컴포넌트를 수행시킨다. 컴포넌트의 수행 결과는 다시 전달자를 거쳐 대기자에게 보내어 지며, 대기자가 다시 사용자의 브라우저로 보내게 된다. 이때 각각의 모듈간 통신은 TCP 소켓을 이용한다.

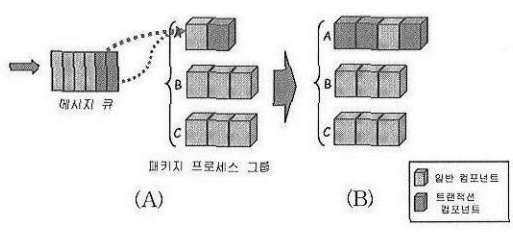

그림 6 트랜잭션 컴포넌트로 인한 부하 불균형

3.2.2 전달자 동작 과정

전달자는 컴포넌트 호출에 대한 메시지를 받아 적절한 패키지 프로세스에게 전달하는 일을 한다. 이러한 작업을 위해 전달자는 두 가지 종류의 정보를 유지하는데, 하나는 패키지와 각 패키지에 속하는 컴포넌트들에 대한 정보로 패키지 인스턴스의 초기 개수 및 최대 생성 개수, 각 컴포넌트의 타입, 동적 링크 라이브러리의 물리적 위치 등 정적인 정보들로 이루어진다. 다른 하나는 현재 동작 중인 패키지 프로세스들에 대한 동적인 상태 정보로, 현재 수행 중인 패키지들의 쓰레드의 개수 및 부하 정도 등이 있다. 전달자는 초기 작업 시 환경 파일을 통해 이러한 정보를 읽어 들여 각 패키지 별로 초기 개수만큼의 프로세스를 띄우게 된다. 이 프로세스들은 메시지 대기 상태로 존재하며, 요청이 도착하게 되면 바로 작업을 수행하므로 추가적인 프로세스 생성에 따른 비용이 없게 된다. 또한 각 패키지 프로세스는 최대로 실행시킬 수 있는 쓰레드의 수가 제한되어 있으며, 만일 가용한 패키지 프로세스가 더 이상 없는 경우(현재 대기 상태로 있는 모든 패키지 프로세스가 자신의 최대 개수 만큼의 쓰레드를 수행하고 있는 경우) 최대 인스턴스 개수 범위 내에서 새로운 프로세스를 생성하고, 새로이 생성된 패키지 프로세스에게 요청 메시지를 전달한다. 패키지 프로세스에 의해 요청이 처리가 되면 HTML 형태의 결과값이 전달자로 오게 되며, 전달자는 그 수행 결과에 따라 적절히 프로세스 상태 정보를 갱신한 후 대기자를 통해 웹 브라우저로 결과값을 보낸다.

3.3 프로세스 부하 균형

웹 브라우저로부터의 메시지를 처리하는 응용 서버인 패키지 프로세스는 서버 그룹을 이룬다. 즉, 각 패키지 프로세스는 유사한 작업을 수행하는 컴포넌트들을 포함하여 몇 가지의 서비스를 처리할 수 있으며, 여러 개의 동일 서버(동일한 작업을 수행하는 패키지 프로세스)가 하나의 서버 그룹을 이루게 된다. 이 구조에서 우리는 프로세스 간의 부하 균형을 통해 성능 향상을 가져올 수 있다. 또한 고장 허용(fault tolerance)을 위하여 동일 서버 그룹을 여러 사이트에 중복 존재시키는 서버

클래스[14][15]로 확장하여 생각할 수 있다.

프로세스 부하 균형의 기본 개념은 동일 작업을 수행하는 프로세스들이 동일한 부하를 가지도록 하는 것이며, 이를 통해 얻을 수 있는 장점은 다음과 같다.

첫째, 한 프로세스에 부하가 편중되면, 그 프로세스에는 많은 수의 쓰레드가 생성되며, 서로의 간섭으로 인해 적은 수의 쓰레드를 여러 프로세스가 수행시키는 것에 비해 그 수행 시간이 길어지게 된다.

둘째, 각 프로세스는 제한된 개수의 쓰레드를 생성하도록 되어있기 때문에, 하나의 프로세스로 요청 메시지가 편중되면, 다른 프로세스는 허용 가능 상태임에도 불구하고, 처리 할 수 없게 된다.

이러한 프로세스 부하 균형을 위해서는 들어오는 요청 메시지들을 각 프로세스가 동일한 부하를 가지도록 분배해야 한다. 만일 모든 컴포넌트가 동일한 부하를 가진다면 요청 메시지들을 라운드 로빈(round-robin)방식으로 스케줄링하는 것이 가장 최적일 것이다. 하지만 컴포넌트 각각은 서로 다른 특성을 가지며, 수행 시간 역시 다르다. 가령 데이타베이스를 접근하여 트랜잭션을 수행하는 트랜잭션 컴포넌트인 경우 데이타베이스를 접근하지 않는 일반 컴포넌트에 비해 그 수행시간이 길다. 따라서 이러한 여러 가지 종류의 컴포넌트들을 일괄적으로 스케줄링하면 부하의 불균형을 가져올 수 밖에 없다. 그러므로 본 논문에서는 컴포넌트를 여러 부류로 나누고 각각의 특성에 맞추어 스케줄링을 수행함으로써 부하 균형을 이룰 수 있는 알고리즘을 제안하고자 한다.

4. 특성 기반 스케줄링 알고리즘

이번 절에서는 WATS의 전달자가 부하 균형을 위해 사용하는 특성 기반 스케줄링(property-based scheduling) 알고리즘에 대해서 살펴본다. 특성 기반 스케줄링 알고리즘의 기본은 컴포넌트를 특성에 따라 분류하고 특성에 맞게 스케줄링하는 것이다. WATS에서는 기본 응용 단위가 컴포넌트이며, 개발자는 자신이 개발한 컴포넌트가 어떠한 특성을 갖는 것이지 시스템에 등록시킬 때에 지정해 주게 된다. 그러면, 전달자는 각 컴포넌트의 특성에 맞춰 스케줄링을 하게 된다. 부하 불균형에 영향을 줄 수 있는 인자에 의해 컴포넌트를 분류하면 다음과 같다.

● 트랜잭션 / 비트랜잭션 컴포넌트

WATS에서 트랜잭션은 데이타베이스 트랜잭션을 의미하며, 트랜잭션 컴포넌트는 데이타베이스를 접근하는 컴포넌트를 의미한다. 이에 반해 일반 컴포넌트는 데이타베이스를 사용하지 않고 파일 시스템 및 계산 작업을 수행하는 컴포넌트로 게시판 등을 예로 들 수 있다. 일반적으로 데이타베이스를 접근하는 컴포넌트는 접근하지 않는 일반 컴포넌트에 비해 긴 수행 시간을 갖는다. 따라서 두 가지의 컴포넌트를 동일하게 취급하면 부하의 불균형을 가져올 수 있다 <그림 5>가 이를 나타내고 있다. 그림에서 프로세스 A는 수행 시간이 긴 트랜잭션 컴포넌트를 수행하고 있고, B나 C의 경우는 일반 컴포넌트 만 수행시키고 있다 만일 두 가지 컴포넌트를 동일하게 취급하면, 프로세스 A가 C에 비해 적은 수의 쓰레드를 수행하고 있으므로 전달자는 A에게 새로운 메시지를 보내게 된다(<그림 5>의 (B) 상황).

하지만 A에서 수행 시간이 긴 트랜잭션 컴포넌트를 수행하는 쓰레드는 C의 일반 컴포넌트를 수행하는 모든 쓰레드가 다 종료된 후에도 계속 수행하게 될 것이다. 따라서 단순히 컴포넌트를 수행하는 쓰레드의 개수만으로 부하를 계산하여 스케줄링 할 때에는 이와 같은 부하 불균형 상황이 발생된다. 그러므로 부하를 계산할 때 트랜잭션 컴포넌트인 경우는 일반 컴포넌트 보다 더 큰 부하 값을 주어 부하 불균형을 피해야 한다. 예를 들어, 트랜잭션 컴포넌트가 일반 컴포넌트에 비해 3배정도의 수행 시간을 요구한다면 일반 컴포넌트에게 1의 부하 값을 배정할 때 트랜잭션 컴포넌트에게는 3의 부하 값을 배정하는 방식으로 가중치를 주어 부하를 균등하게 해줘야 한다.

● 세션(session) 가능/불가능 컴포넌트

하나 더 고려해야 할 사항은 세션에 관한 측면이다. 이는 HTTP가 상태 비 보존(stateless) 프로토콜[1][16] 이기 때문에 발생한다. 즉, 하나의 요청 메시지는 이전의 메시지와 아무런 상관 관계를 가지지 않는다. 이러한 특성은 초창기 정적인 문서 위주의 웹 서비스에서는 성능 향상 및 대량의 요청에 대해서 서버의 부담이 적다는 장점으로 작용하였으나, 점차 웹 응용이 복잡해짐에 따라 문제점으로 대두되었다. 가령, 여러 페이지에 걸친 트랜잭션 처리를 생각해 볼 수 있으며, 이러한 실제 응용으로 가상 쇼핑을 생각해 볼 수 있다. 사용자는 가상 쇼핑 공간에서 여러 페이지를 돌아다니며, 자신이 원하는 물품들을 선택한 후 최종적으로 구매를 하게 된다. 이 때 트랜잭션은 비로소 종료된다. 물론 마지막에 그동안 구매 예정이었던 모든 물품들에 대한 선택을 취소할 수도 있다. 이러한 경우 응용 프로그램은 사용자가 이전 페이지에서 선택한 물품에 대한 정보를 알아야 하는데, 웹에서 사용하는 HTTP는 상태 비 보존의 성질을 가지기 때문에 그것 자체로는 처리할 수 없으며, 다른 보조 수단이 필요하다. 이러한 문제가 제기된 이후로 여

러 해결 방법이 등장하였는데, 그 중 가장 보편화된 방법이 쿠키(cookie)를 사용하는 방법이다[17][18]. 쿠키는 Netscape사에서 처음 제안된 것으로 일종의 아이디라고 할 수 있다. 사용자가 서버에 처음 접속하였을 때, 서버는 브라우저로 쿠키를 보내게 된다. 그 후 브라우저가 동일한 서버로 재접근할 경우 서버에게 자신이 이전에도 접속했음을 인식시키기 위하여, 이전에 서버에서 주었던 쿠키를 보내게 된다. 그러면 서버는 쿠키를 이용하여, 해당 브라우저의 방문 여부를 확인하게 된다. WATS 역시 이러한 기법을 이용하여 여러 페이지에 걸친 트랜잭션을 지원한다. 즉, 응용 프로그래머는 컴포넌트를 등록시킬 때 해당 컴포넌트가 세션 가능 컴포넌트임을 명시한다. 처음 세션 가능 컴포넌트에 대한 요청이 도착하면 패키지 프로세스는 새로운 쓰레드를 생성하여 처리를 맡긴 후 해당 쓰레드를 종료시키지 않고 보류 상태로 두며 자동으로 쿠키를 할당하여, 다음 요청에 대해 세션을 유지하기 위해 필요한 정보를 브라우저 측으로 전달한다. 세션은 웹 클라이언트와 서버 사이에 존재하는 가상 연결이라 생각하면 된다. 순차적인 요청이 브라우저로부터 전달되어지면 패키지 프로세스 내의 쓰레드 관리자는 쿠키 정보를 이용해 이전에 보류 상태로 두었던 쓰레드를 재실행 시켜 기존에 했던 작업을 이어나간다(<그림 4> 참조).

이러한 세션 가능 컴포넌트의 특성을 살펴보면 다음과 같다.

첫번째로 세션 가능 컴포넌트에 대한 요청 메시지는 처리해야 할 패키지 프로세스가 미리 정해져 있다는 것이다. WATS에서 동일한 작업을 수행하는 패키지 프로세스는 여러 개 있을 수 있다.(관리자가 미리 정해진 개수의 프로세스 인스턴스를 생성한다.) 그러므로 해당 컴포넌트에 대한 요청은 여러 패키지 프로세스들 중 임의로 하나를 선택하여 전달되면 된다. 하지만 세션 가능 컴포넌트에 대한 순차적인 메시지는 반드시 처음 컴포넌트가 수행 시작된 패키지 프로세스에게 계속 전달되어야 하며, 이러한 특성은 부하 불균형을 초래할 수 있다. <그림 5>의 (A)를 살펴보자. 패키지 프로세스 A의 진하게 되어있는 컴포넌트가 세션 가능 컴포넌트이며, 처음 상태에서는 프로세스 A가 가장 부하가 적으므로 연속된 요청들이 패키지 프로세스 A로 전달될 것이고, 얼마간의 시간 경과 후에는 A의 부하가 가장 크게 될 것이다(<그림 5>의 (B)). 이 상태에서 다음 메시지가 패키지 A에 보류 상태로 존재하는 컴포넌트[3] 에 대한 메시지인 경우 이 메시지는 다른 패키지들의 부하가 적음에도 불구하고 반드시 패키지 프로세스 A로 전달 되야 한다. 따라서 이로 인해 부하의 불균형을 가져올 수 있다. 그러므로 이러한 문제점을 해결하기 위해서는 세션 가능 컴포넌트 역시 트랜잭션 컴포넌트와 마찬가지로 부하 값을 더 크게 해주어야 한다.

두번째 특징으로 세션 가능 컴포넌트는 여러 번의 요청들에 의해 전체 작업이 종료될 때까지 계속해서 자원을 점유한다는 것이다. 즉, 세션 가능 컴포넌트를 수행시키는 쓰레드는 처음 수행 후 연속된 요청이 모두 도착할 때까지 대기하게 된다. 한 패키지 프로세스가 수행 가능한 최대 쓰레드 개수는 정해져 있으므로 이러한 대기 상태의 쓰레드들 때문에 새로운 요청이 처리되지 못하고 대기하게 된다. 또한 첫번째 특징을 해결하기 위해 세션 가능 컴포넌트에 대한 부하 값을 높여주게 되면, 대기 상태의 쓰레드가 다수 존재하는 프로세스가 생겨, 부하 불균형 및 메시지 대기 상황이 발생할 수 있다. <그림 6>을 보면, 한 패키지가 생성할 수 있는 최대 쓰레드 개수가 3으로 제한 되어 있는 경우 세션 가능 컴포넌트를 수행하는 쓰레드가 다수 존재함으로써 나타나는 문제점을 나타내고 있다. 패키지 프로세스 A 의 경우 3개의 쓰레드 전부가 세션 가능 컴포넌트로, 생성되어 있는 쓰레드의 개수는 최대 개수인 3이지만 실제로 수행하는 작업은 하나도 없다. 즉, 전부 보류중인 쓰레드들로 인해 새로운 메시지들이 처리되지 못하고 있는 상황이다. 이러한 문제점에 대한 해결책은 세션 가능 컴포넌트에 대한 요청 메시지를 다른 일반 컴포넌트에 대한 요청 메시지 보다 먼저 수행시켜주는 것이다. 그렇게 함으로써 부하 불균형 및 메시지 대기 상황을 줄일 수 있다. 이를 위해 전달자가 갖는 메시지 큐를 일반적인 큐와 다른 일종의 우선 순위 큐로 구현하여, 임의의 위치에 삽입이 가능하게 하였다. 따라서 세션 가능 컴포넌트에 대한 요청이 들어올 경우 삽입 위치 비율에 의해 메시지 큐에 존재하는 다른 일반 컴포넌트들에 대한 요

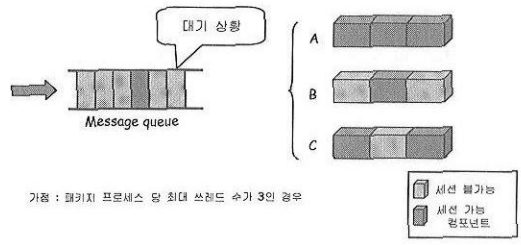

그림 7 세션 가능 컴포넌트의 문제점

3) 컴포넌트는 정적인 상태를 의미하고 실제 동작 시에는 쓰레드가 컴포넌트를 로드시켜 수행된다.

청 메시지 보다 앞에 삽입하여 우선적으로 수행할 수 있게 하였다.

● 특성 기반(property-based scheduling) 스케줄링 알고리즘

```
알고리즘 1 특성에 따른 부하 값 배정
LOAD_FACTOR   n      // 수행 시간을 고려하여 설정
while(TRUE) {        // 무한 루프
   fetch (message);  // 새로운 메시지를 큐에서 가져온다
   최소 부하를 가지는 패키지 프로세스 선택,
   해당 패키지 프로세스로 메시지 전달,
   if (message type = transactional or sessionable)
      LOAD_FACTOR 만큼 부하 값 증가;
   else
      1 만큼 증가;      // 일반 컴포넌트인 경우
}
```

```
알고리즘 2. 특성에 따른 큐 삽입
POSITION_FACTOR   f   // 큐 삽입 위치 비율 지정.
// 0  큐의 시작 위치, 1 : 큐의 마지막 원소 위치

while (TRUE) {
   대기자로 부터 메시지 도착;
   if (message type = sessionable) {  // 해당 메시지가 세션 가능
컴포넌트에 대한 메시지인 경우
      삽입 위치 = 큐 원소 개수 * POSITION_FACTOR;
      AddQueue(삽입위치),
      // POSITION_FACTOR에 의해 큐에 삽입,
      // 일반 컴포넌트 보다 우선적 처리
    }
   else
      큐의 마지막에 삽입;   // 일반적인 큐 삽입 연산
}
```

그림 7 특성 기반 스케줄링 알고리즘

<그림 7>은 특성 기반 스케줄링 알고리즘을 나타낸다. 알고리즘 1은 컴포넌트의 특성에 따라 서로 다른 부하 값을 할당하고, 알고리즘 2는 메시지 큐에 삽입 시 세션 가능 컴포넌트에 대한 처리를 나타낸다.

5. WATS 성능 평가

5.1 개요

이번 절에서는 특성 기반 스케줄링 알고리즘으로 구현된 WATS의 성능 측정 결과를 제시하고 분석함으로써 일반적인 라운드 로빈 스케줄링 기법을 사용하는 것보다 비교 우위에 있음을 확인한다. 현재 대부분의 웹 응용 서버들은 특별한 스케줄링 기법이 없이 라운드 로빈 방식의 스케줄링 기법을 채택하고 있다. 본 논문에서 성능 측정의 비교 대상은 두 가지 종류의 WATS, 즉 일반적인 라운드 로빈(round-robin) 방식으로 구현된 것과 특성 기반 스케줄링 알고리즘을 적용한 것 간의 비교이다. 따라서 성능 측정 결과의 절대적인 값을 논하기 보다는 상대적인 값으로 두 알고리즘 간의 성능 비교를 해야 옳을 것이다. 이미 확장 API 응용 서버 방식이 다른 방식 보다 좋은 성능을 나타낸다는 것은 자명하며[1], 다른 동일한 방식의 응용 서버와의 비교는 시스템의 복잡성 등의 요인들로 인해 정확한 비교가 어렵다

앞서 언급했듯이 웹 응용 서버에 대한 성능 평가 도구는 아직까지 없다. 하지만 웹 서버의 성능 평가 도구를 사용하되 데이타베이스와의 연동 및 컴포넌트 종류에 따른 비교를 추가적인 요소를 가함으로써 웹 응용 서버에 대한 성능 측정에도 적용하는 것이 가능하다. 따라서 본 성능 평가에서는 여러 웹 서버 성능 평가 도구 중 Webpest[19]를 사용하여 측정하였다.

5.2 Webpest

Webpest[11][7]는 웹 환경을 큐잉 네트워크로 모델링 하였다. 모든 웹 요구들은 서로 독립적으로 요구된다고 가정하여 각 웹 요구의 도착 시간 간격을 지수 분포로 모델링 한다. Webpest의 시험 환경은 <그림 8>에서 보듯이 서버 시스템과 클라이언트 시스템으로 이루어진다. 서버 시스템은 웹 서버와 WATS를 수행 시켜 webpest가 생성해내는 요청 메시지들을 받아 처리한 후 결과값을 반환하며, 클라이언트 시스템에서는 webpest가 주어진 여러 인자들을 바탕으로 WATS에 대한 요청 메시지들을 생성해낸 후 서버 시스템으로부터 전송되는 결과값을 받아 각종 자료를 기록한다. Webpest 수행 시 사용자가 주어야 하는 인자들에는 평균 도착간 시간(mean interarrival time), 시뮬레이션 시간, 안정 상태(steady state)에 도달하기 전까지 무시할 요청의 개수 등이 있다. 또한 부하 명세 파일이 있는데, 이에는 각 요청이 전체에서 차지하는 비율, 호출 방식(GET, POST, HEAD), 요청 URL 등이 기술되며, webpest는 수행 시 이 파일을 읽어 들여 주어진 비율에 따라 주어진 HTTP 요청을 생성한다.

이와 같은 Webpest의 동작 구조는 부모 프로세스와 자식 프로세스 그리고 부하 명세 파일로 이루어진다 (<그림 8> 참조). 먼저 부모 프로세스는 부하 기술 파

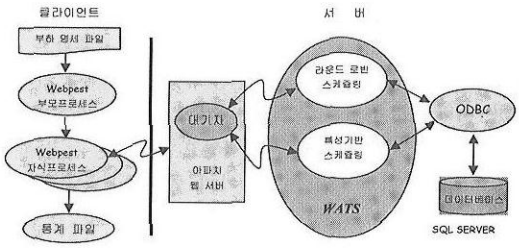

그림 8 성능평가 실행 구조

일을 읽어 들이고, 입력받은 평균 도착간 시간 값을 이용 지수분포로 매 요구마다 자식 프로세스를 생성하고, 자식 프로세스는 요구를 처리한 후 결과값을 계산하여 출력 파일에 기록한다. 모든 자식 프로세스들이 종료되면, 부모 프로세스는 결과파일에 수집된 측정치를 이용하여 통계자료를 생성한다. 부모 프로세스는 정해진 시간동안 자식 프로세스의 종료를 대기하며, 이 시간이 경과되면 시간 초과(time-out)로 기록한다.

5.3 성능 평가 환경

본 성능 평가의 비교 대상은 특성 기반 스케줄링과 라운드 로빈 스케줄링 간의 비교로, 각각을 이용하여 구현된 두 가지의 WATS를 사용하여 측정한다. 따라서 본 성능 평가의 결과값으로 확장 API 응용 서버 방식의 WATS와 다른 구조를 따르는 웹 응용 서버와 절대적인 비교를 하는 것은 무의미하다. 성능 평가 실행 구조는 <그림 8>과 같으며, 클라이언트 컴퓨터와 서버 컴퓨터 각각의 사양은 <표 1>와 같다. 클라이언트 쪽에는 webpest가, 서버 쪽에는 아파치 웹 서버, WATS, 그리고 데이타베이스 서버인 SQL Server가 각각 수행된다.

표 1 성능평가에 사용된 클라이언트 컴퓨터와 서버 컴퓨터 사양

	클라이언트	서버
시스템 기종	SUN ULTRA 1	Pentium II
CPU	UltraSparc(167MHz)	Pentium II 266MHz
메인 메모리	128MB	128MB
운영체제	Solaris 2.5.1	Windows NT 4.0
망 속도	10Mbps	

서버 쪽에 설치된 웹 서버는 동시 요구량을 50개로 하였으며, WATS의 동시 요구량 역시 50으로 하였다. 이는 아파치 웹 서버가 허용하는 한 프로세스 당 최대 생성 쓰레드의 개수이다. 너무 많은 쓰레드를 생성할 경우 교체 비용으로 인한 쓰레싱(thrashing) 현상이 일어날 수 있기 때문이다. 데이타베이스는 Microsoft사의 SQL Server 6.5를 사용하였으며, 이에 대한 접속은 ODBC를 사용하였다.

그밖에 한 패키지 프로세스당 허용 가능 쓰레드의 수는 6으로 하였다. 쓰레드의 수가 너무 적을 경우 부하가 너무 적어 두 스케줄링 간의 비교가 불가능하다. 그리고 패키지 프로세스의 인스턴스 수는 4로 하였다. 이 역시 개수가 너무 적으면 스케줄링에 의한 효과가 적어지게 된다.

5.4 수행 환경

3절에서 언급했듯이 WATS는 컴포넌트를 특성에 따라 분류한다. 즉, 데이타베이스에 대한 접근 여부와 세션의 유지 여부에 따라 네 가지 종류로 나눌 수 있다. 따라서 본 성능 평가에서는 이 네 가지 종류의 컴포넌트에 대한 요청 메시지를 생성하여 각각의 응답 시간을 기록하였다. 앞서 언급했듯이 webpest는 부하 명세 파일에 기록된 URL 요청을 지정된 비율만큼 생성해 낸다. 이를 이용하여 부하 명세 파일에 WATS의 네 가지 종류의 컴포넌트에 대한 요청과 각각의 비율을 적어주면 webpest가 각각의 요청을 생성해낸다. 이들 네 가지 종류의 요청에 대해 살펴보면 <표 2>와 같으며, 각각을 25%씩 생성하였다.

(A) 트랜잭션/세션 불가능 컴포넌트 : 데이타베이스를 접근하는 컴포넌트로 각 요청에 의해 데이타베이스에 접근하여 데이타베이스 연산을 수행한 후 그 결과값을 반환한다.

표 2 WATS의 컴포넌트 분류

		세션 유지	
		O	X
데이타베이스 접근	O	(B)	(A)
	X	(D)	(C)

(B) 트랜잭션/세션 가능 컴포넌트 : 연산은 트랜잭션 컴포넌트와 동일하나 해당 연산을 수행한 쓰레드는 종료되지 않고 보류 상태로 대기하고 있다가 다음 요청 메시지가 도착하면 다시 수행을 계속한다. 요청 회수는 3회로 3번째 요청 메시지가 해당 쓰레드로 전달되면 더 이상 보류하지 않고 종료한다.

(C) 일반/세션 불가능 컴포넌트 : 해당 컴포넌트는 요청 메시지에 의해 새로운 쓰레드를 생성하여 파일 연산을 수행한 후 종료한다. 트랜잭션 컴포넌트에 비해 수행 시간이 적다.

(D) 일반/세션 가능 컴포넌트 : 작업 내용은 일반 컴포넌트와 같으며, 해당 쓰레드가 작업 수행 후 종료하지 않고 다음 요청 메시지를 기다리며 보류 상태로 대기하다가 다음 요청 메시지가 도착하면 수행을 계속 한다. 이것 역시 3회 수행한다.

데이타베이스에 대한 연산은 SQL 문을 사용하여 검색 질의를 하였으며, 파일 연산은 지정된 파일을 연 후 읽기 연산을 수행하였다. 이 때 각 연산은 반복 작업에 의해 그 수행 시간을 길게 하였는데, 이는 각 컴포넌트의 수행 시간이 짧은 경우 네트웍 시간으로 인해 수행에 따르는 시간 비중이 너무 적기 때문에 수행 시간이 전체적으로 차지하는 비율이 높게 하였다. 웹 응용 서버는 기본적으로 복잡한 데이타베이스 작업 및 여러 파일에 대한 읽기/쓰기 작업 등 많은 일을 수행하므로 현실적으로 그 수행 시간이 매우 길어지게 된다.

세션 가능 컴포넌트에 대한 요청 처리 시 발생할 수 있는 문제점은 webpest가 일반 브라우저와 달리 쿠키를 지원하지 않는다는 점이다. 또한 이 점은 쿠키를 지원하지 않는 브라우저를 사용하여 WATS의 세션 가능 컴포넌트를 사용할 때에 발생하는 문제이기도 하다. 이러한 문제를 해결하기 위해 본 성능 평가에서는 전달자가 세션 가능 컴포넌트 수행 후 생성되는 세션 정보에 관한 쿠키를 저장하였다가 해당 컴포넌트에 대한 다음 요청이 전달되면 이전에 저장했던 쿠키를 이용하여 세션을 유지하도록 전달자를 구현하였다. 이 방법을 사용하면 쿠키를 지원하지 않는 브라우저도 WATS에서 제안한 세션을 유지 방법을 그대로 적용하여 세션 가능 컴포넌트에 대한 수행을 행할 수 있다.

각 수행마다 앞 수행의 영향을 최소화하기 위해 서버 환경을 초기화한 후 수행하였으며, 각 수행 시간은 360초로 하였다. 그리고 서버 시스템이 평형 상태(steady state)에 이르는 동안의 결과를 무시하기 위해 앞 60초 동안의 측정치는 무시하고 나머지 300초 동안의 측정치만 사용하였다.

Webpest는 수행 시 인자로 받는 평균 도착간 시간에 의해 요청을 생성해낸다. 평균 도착간 시간이 짧아지면 각종 큐에 대기하는 요구의 개수가 증가하고 부하가 증가해 점차 수행 속도가 느려지게 된다. 본 시뮬레이션에서는 처음에 순차적인 요청으로 시작하여 점차 평균 도착간 시간을 작게 줌으로써 부하를 증가 시켜가며 두 스케줄링 알고리즘간의 응답 시간의 변화를 기록하였다.

5.5 성능 평가 결과

다음의 <표 3>은 평균 도착간 시간이 600ms인 상황에서 특성 기반 스케줄링 알고리즘의 두 가지 중요 인자인 부하 비율과 큐 삽입 비율 값의 차이에 따른 응답 시간의 변화를 보이고 있다. 표에서 보면 메시지 큐의 삽입 위치가 뒤(rear)로 갈 수록 응답 시간이 빠르다(0.6과 0.7간의 비교). 하지만 1.0인 경우는 일반적인 큐 연산과 동일하여 오히려 느려짐을 알 수 있다. 메시지

표 3 동일 평균 도착간 시간(600ms)하에서 부하 비율 및 큐 삽입 비율에 따른 응답시간

		큐 삽입 비율(0-1)		
		0.6	0.7	1.0
부하 비율	5	6788	6565	7079
	8	6780	6451	

큐는 배열로 이루어져 있으며 임의의 위치에 삽입하게 될 경우 다른 원소들은 차례로 옮겨진다. 즉, 삽입 위치가 앞(front) 쪽에 가까워질수록(큐 삽입 비율이 0에 가까워질수록) 원소 이동에 따른 부담이 커지게 된다. 또한 평균 도착간 시간이 짧아져 요청이 많아질 경우 이러한 이동 연산에 대한 부담이 커져서 이동이 적은 쪽이 빠른 응답 시간을 나타낸다. 하지만 평균 도착간 시간이 커짐에 따라 요청 메시지가 줄어들게 되면 큐 원소 이동에 대한 부담은 상대적으로 적어지며 큐 삽입 비율을 적게 해주는 것이 유리함을 알 수 있었다. 실제로 평균 도착간 시간이 800ms 인 경우는 큐 삽입 비율이 0.5일 경우에 가장 좋게 나왔으며, 1000ms 인 경우 0.4가 좋은 결과를 보였다. 따라서 큐 삽입 비율은 평균 도착간 시간에 따라 적절히 조정해 주어야 한다. 반면, 동일한 큐 삽입 비율 하에서 부하 비율에 따른 응답 시간의 비교를 보면, 부하 비율이 8인 경우가 5인 경우 보다 더 적은 응답 시간을 보였다. 이는 트랜잭션 컴포넌트와 비트랜잭션 컴포넌트 간 수행 시간의 차이에 따라 적절히 부하 비율을 조정해줘야 함을 의미한다.

<그림 9>는 평균 도착간 시간의 변화에 따른 특성 기반 스케줄링과 라운드 로빈 스케줄링 각각의 평균 응답 시간 변화 과정을 보여준다. 처음 순차적인 요청에 대해서 가장 짧은 응답 시간을 나타내다가 점차 평균 도착간 시간이 줄어들면서 평균 응답 시간이 증가함을 알 수 있다. 평균 도착간 시간이 점차 짧아지면 동시에 처리해야 할 메시지들이 증가하고 각 패키지 프로세스들은 많은 수의 쓰레드를 생성하여 컴포넌트를 수행시키므로 이와 같은 현상이 나타나며, 부하가 커지면서 프로세스 부하 균형 여부에 따른 응답 시간의 차이가 증

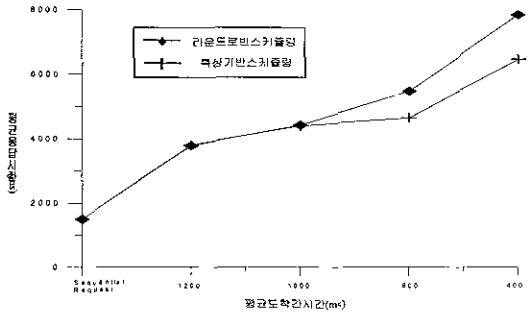

그림 9 두 가지 스케줄링 방법의 평균 응답 시간 결과

지면 프로세스 간 부하 불균형을 초래하고, 이는 특성 가하게 된다. 즉, 라운드 로빈 스케줄링은 요청이 많아 기반 스케줄링 보다 급격한 응답 시간의 증가를 가져온다. 처음 순차적인 요청을 보내는 경우 패키지 프로세스는 동시에 하나의 요청만을 처리하므로 두 스케줄링 간의 차이는 거의 없다(라운드 로빈 스케줄링 1495ms, 특성 기반 스케줄링 1496ms). 하지만 평균 도착간 시간이 점차 감소함에 따라 두 가지 스케줄링 간의 응답 시간의 차이는 증가하며, 600ms로 감소하였을 때에, 라운드 로빈 스케줄링의 평균 응답 시간은 7831ms이고 특성 기반 스케줄링의 평균 응답 시간은 6451ms이며, 이는 대략 17.6% 정도의 응답 시간의 차이이다.

<그림 10>는 평균 도착간 시간을 점차로 줄여가며 처리량을 측정한 결과이다. 처리량은 300초 동안 완료된 HTTP 트랜잭션의 개수이다 그림에서 보듯이 비록 적은 양이지만 특성 기반 스케줄링 알고리즘을 적용한 것이 보다 많은 처리를 하는 것을 알 수 있다. 또한 100ms가 되면서 나타나는 처리량의 감소도 보다 완만하게 진행되는 것도 그림에서 확인할 수 있다.

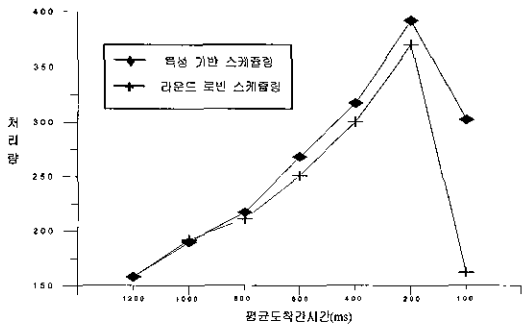

그림 10 두 가지 스케줄링 방법의 처리량 결과

6. 결론 및 향후 연구 계획

본 논문에서는 여러 가지 웹 응용 서버의 구조에 대해 알아보았고, 대량의 메시지 처리에 적합한 구조인 확장 API 응용 서버 구조를 적용한 WATS에 관하여 알아보았다. WATS는 컴포넌트 단위의 웹 응용 개발로 인해 응용 작성의 용이함과 재사용성을 높이고, 각 컴포넌트의 특성에 맞는 효율적인 스케줄링인 특성 기반 스케줄링 알고리즘을 사용하여 프로세스 부하 균형을 이루었다. 또한 성능 측정을 통해 특성 기반 스케줄링 알고리즘이 일반적인 라운드 로빈 방식의 스케줄링 알고리즘 보다 성능상의 비교 우위에 있음을 확인해 보았다.

향후에는 컴포넌트 기반이라는 장점을 살려 보다 다양한 언어를 지원할 수 있도록 각 언어별 컴포넌트의 개발 및 CORBA[19]와의 연동을 통한 CORBA 컴포넌트의 지원을 고려하고 있으며, 본 연구실에서 개발한 객체지향 데이타베이스 관리 시스템인 SOP[4][20]와의 효율적인 연동을 통해 객체 지향 데이타베이스와 웹과의 효율적인 연동에 관한 연구 역시 진행 할 예정이다.

참 고 문 헌

[1] 김평천, "UniWeb 2.0 - 웹을 이용한 클라이언트-서버 데이타베이스 응용 개발 환경", 데이타베이스 저널, 3(2), pp. 119-132, 1996.

[2] 최일환, "SweS: SRP RDBMS를 위한 Web 통로", Master's thesis, 서울대학교, 컴퓨터공학과, Feb. 1998, (accepted for publication), http://wwwoopsla.snu.ac.kr/publication/.

[3] A. Luotonen and T.Berners-Lee, CERN httpd Reference Manual - A Guide to a World-Wide Web Hypertext Daemon, CERN, May 1994.

[4] Oracle application server. http://www.olab com

[5] Cybase Jaguar CTS. http://www.powersoft.com/products/jaguar/

[6] Microsoft transaction server. http://support microsoft.com/support/transaction/default.asp

[7] 김평철, 민영훈, "윈드와이드웹용 데이타베이스 통로의 성능 평가", 데이타베이스 연구회지, 제 13권 2호, pp. 30-39, 1997.

[8] Robert E. McGrath, "Measuring the Performance of HTTP Daemons," Feb. 1996. URL : http://www.ncsa.uiuc edu/InformationServers/Performance/Benchmarking/bench.html

[9] The benchmark for web servers. http //www mindcraft.com/webstone/

4) SOP는 서울대학교 컴퓨터공학과 객체지향시스템 연구실에서 개발한 객체지향 데이타베이스시스템이다

[10] SPECweb96 benchmark. http://www.spec.org/osg/web96/
[11] S. Srinivasan, "Webpest: A Tool to Evaluate Hypertext Server Performance," Presented at the poster session of 4th Int'l World Wide Web Conf., Boston, MA, Dec. 1995.
[12] Apache HTTP server project. http://www.apache.org
[13] Netscape application server. http://home.netscape.com/appserver/v2.1/index.html
[14] Philip A. Bernstein, Principles of Transaction Processing, Morgan Kaufmann Publishers, Inc, 1997.
[15] 유호동, 임성채, 김명호, "고성능 온라인 트랜잭션 처리 모니터의 설계 및 구현", 정보과학회논문지, 제21권, 제5호, pp. 816-825, 1994.
[16] H. F. T. Berners-Lee and R. Fielding, Hypertext Tranfer Protocol - HTTP/1.0, Internet RFC 1945, May 1996.
[17] D. Kristol and L. Montulli, HTTP State Management Mechanism, Internet RFC 2109, Feb. 1997.
[18] Netscape cookie specification. http://home.netscape.com/newsref/std/cookie_spec.html/
[19] Object management group. http://www.omg.org/
[20] SNU OOPSLA Lab. SOP ODBMS PLATFORM 1.0 Manual, 1996, http://wwwoopsla.snu.ac.kr/~sop/

이 형 돈
1997년 홍익대 컴퓨터공학과(학사). 1999년 서울대 컴퓨터공학과(석사). 1999년 현재 서울대 컴퓨터공학과 박사과정. 관심분야는 객체지향시스템, 데이타베이스, 웹, 미들웨어

이 병 준
1996년 서울대 컴퓨터공학과(학사). 1998년 서울대 컴퓨터공학고(석사). 1998년~현재 서울대 컴퓨터공학과 박사과정. 관심분야는 객체지향 시스템, 데이타베이스, 질의어 처리, 트랜잭션

김 형 주
제 5 권 제 1 호(C) 참조