웹 어플리케이션 전송 네트워크

한국과학기술원 임민열 · 김정숙 · 송준화 · 이윤준 *

1. 서 론

최근에 인터넷 사용자가 폭발적으로 증가하면서 여러 가지 다양하고 새로운 서비스들이 제공되기 시작하였다. 이로 인해, 웹 서비스를 제공 받는 사용자들의 서비스 질에 대한 요구는 서비스 내용뿐만 아니라 빠르고 안정적인 서비스를 제공하도록 계속적으로 요구되고 있다.

초기의 웹 서비스가 대부분 정적 컨텐츠로 구성되어 쉽게 복제하여 분산시킬 수 있었던 반면, 최근의 웹 서비스들은 클라이언트로부터 요청이 들어왔을 때마다 서버에서 생성된 동적 컨텐츠로 구성된다. 동적 컨텐츠를 제공하는 서버는 정적 컨텐츠를 처리하는 서버보다 사용자의 요청을 처리하기 위해 훨씬 많은 부하를 받게 된다. 또한, 단순히 복제하여 분산시키는 것이 거의 불가능 하기 때문에, 사용자의 요구에 따라 생성되어 서비스되는 것이 일반적이다. 따라서 위와 같은 서비스를 제공하는 서버는 단순히 정적 컨텐츠를 제공하는 웹 서버보다 확장성(scalability)를 더 크게 요구하게 된다.

최근의 웹 서비스의 형태를 고려하였을 때, 웹 서비스의 지연이나 에러와 같은 서비스의 질적 저하는 다음에서 원인을 찾아 볼 수 있다.

첫 번째, 네트웍 정체로 인해 서비스가 지연되거나 단절될 수 있다. 이것은 인터넷에서 상호 연결을 제공하는 라우터나 DNS(Domain Name System)의 병목 현상이나 시스템 에러에 의해 또는 네트웍 회선 대역폭의 한계로 인해 발생할 수 있다.

두 번째, 데이타베이스 서버의 처리 능력이 한계가 되거나 커넥션과 같이 서비스를 제공하기 위해 사용하는 시스템 자원(예를 들어, 메모리)의 제한으로 인해 요청 서비스를 원활히 제공하지 못하는 것에 기인할 수 있다.

세 번째, 두 번째와 마찬가지로 미들웨어라고 부르는 웹 서버나 웹 어플리케이션 서버가 처리 능력의 한계가 되거나 사용되는 자원의 제한으로 인해 클라이언트의 요청을 제대로 처리할 수 없을 경우 발생할 수 있다.

위의 다양한 원인들 중에서, 주로 웹 어플리케이션 서버(Web Application Server, WAS)를 포함하는 미들웨어 계층은 서비스의 다양화에 따라 가장 크게 변화해 왔다. 따라서 무엇보다도 확장성이 가장 문제가 되고 있는 부분이다.

지금까지 서비스 제공자는 웹 사용자들에게 빠르고 효율적인 서비스를 제공하기 위해서 사전에 수요를 예측하고 그것에 따라 웹 서버를 클러스터링하고 부하 분산하여 서비스의 확장성을 해결해 왔다. 하지만 전자상거래와 같은 서비스를 제공하기 위해서 웹 어플리케이션 서버가 사용되면서 웹 사용자의 수요를 예측하기도 어려워 졌을 뿐 아니라, 각 사용자가 서비스를 제공받기 위해 사용하는 서비스 프레임워크의 리소스도 다양해 졌다. 이와 같은 환경에서, 서비스 제공자는 서비스 프레임워크의 확장성을 사용자가 원하는 만큼 효율적으로 제공하지 못하게 되었다. 사용자의 수가 급격하게 변화할 경우, 필요한 서비스 프레임워크의 리소스를 예측하기 힘들게 되고 서비스 지연이나 접속이 불가능한 일이 벌어지게 된다.

본 논문에서는 웹 어플리케이션 서버의 계층에서 동적 확장성을 제공할 수 있는 서비스 프레임워크을 제안하여 확장성 문제를 해결한다. 즉, 웹 어플리케이션 서버 내부에서 동적 컨텐츠를 생성하는 웹 어플리케이션을 근거리 또는 원거리에 있는 서버로 전송하여 효율적으로 동적 컨텐츠를 처리하는 방법을 본

* 종신회원

논문에서 제안하고자 한다(그림 1).

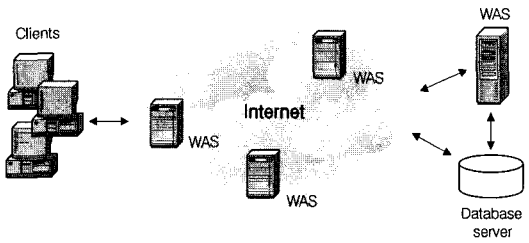


그림 1 웹 어플리케이션 전송 네트웍의 간단한 구조

클라이언트가 근원 서버에 동적 컨텐츠에 대한 요청을 보냈을 경우, 서버는 자신의 부하에 대한 상태를 가지고 지정된 정책에 따라 웹 어플리케이션을 다른 서버로 보낸다. 이 서버를 대상 서버라고 부른다. 그 후, 근원 서버는 클라이언트에게 대상 서버의 주소를 알려주게 된다. 클라이언트는 대상 서버를 통해 서비스를 받는다.

위와 같이, 웹 어플리케이션을 전송할 경우 다음과 같은 이점이 있다.

① 웹 성능 향상

웹 어플리케이션을 원거리로 분산함으로써 클라이언트는 근원 서버보다 토플로지(topology)상 보다 가까운 서버로부터 서비스를 받을 수 있게 된다. 따라서, 응답시간이 줄어들게 되고, 네트웍 대역폭을 절약할 수 있게 된다. 또한, 시스템 전체적으로 확장성과 가용성이 증가하게 된다.

② 동적 부하 분산

서버의 부하에 따라 동적으로 요청을 분산할 수 있다.

③ 시스템 관리 향상

근거리 또는 원거리에 분산되어 있는 웹 어플리케이션을 근원 서버에서 쉽게 통합하여 관리할 수 있다.

향후 동적 웹 컨텐츠의 비중이 점점 커질 경우, 웹 어플리케이션 자체를 다른 서비스 제공자에게 전송하여 확장성 문제를 해결할 수 있게 된다. 즉, 근원 웹 어플리케이션 서버에서 미리 예측된 클라이언트 요청를 처리하도록 하고, 그 이상의 요청들은 다른 서비스 제공자를 통해 서비스 받도록 아웃소싱(outsourcing)할 수 있는 것이다.

본 논문의 나머지 부분의 구성은 다음과 같다. 2장은 기존의 정적 컨텐츠 위주의 서비스에 확장성을 제공하는 방법을 설명한다. 3장에서는 기존의 연구와의 차이점을 설명하고, 효율적인 동적 컨텐츠 처리 기법을 제안한다. 4장에서는 실험을 통해 제안된 방법의 성능을 평가한다. 마지막으로 5장에서는 결론과 향후 연구 방향을 기술한다.

2. 관련 연구

본 장에서는 기존에 웹 서비스 확장성을 높이는 방법인 CDN(Contents Delivery Network)에 대해 기술한다.

아래의 그림 2는 CDN을 구성하여 웹 서비스를 제공하는 구조를 보여준다. 컨텐츠 제공자(Content Provider)가 특정 서버를 통해 웹 서비스를 제공할 경우, 클라이언트들이 빠르고 안정적으로 이 서비스를 제공 받을 수 있도록 인터넷 상에 캐쉬 서버(Cache server)를 분산시켜 두고 클라이언트의 요청을 가까운 캐쉬 서버에서 처리할 수 있도록 네트웍을 구성한다. 클라이언트가 요청한 컨텐츠는 Cache server에 의해 미리 캐쉬되어 있을 경우 바로 서비스가 이루어지고, 그렇지 않으면 컨텐츠 제공자로부터 직접 가져와 서비스를 제공한다.

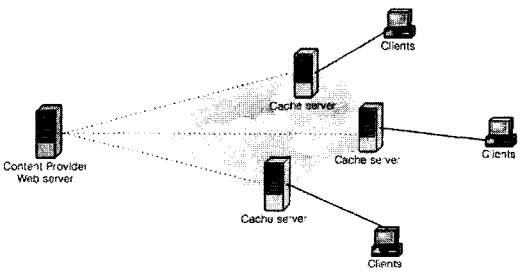


그림 2 Contents Delivery Network

이 방법은 우선 네트웍 회선의 사용을 줄이게 되고, 웹 서버의 부하를 분산시키게 되어, 서비스의 가용성을 높아지게 되며, 클라이언트는 웹 서버로부터 직접 서비스 받는 것보다 서비스 지연이 획기적으로 감소하게 된다. 하지만, 기존의 캐쉬 서버는 단순히 정적 컨텐츠만을 캐쉬할 수 있기 때문에, 최근의 웹 서비스의 특성을 반영하지 못한다.

만약, 캐쉬 서버의 동작을 수정하더라도 동적 컨텐츠를 캐쉬하는 것은 정적 컨텐츠를 캐쉬하는 것에 비해 매우 복잡하다. 또한, 동적 컨텐츠를 캐쉬하더라도, 그것의 사용빈도가 높지 않기 때문에 효용성이 떨어 질 수 있다. 이를 보완하기 위해서 동적 컨텐츠

에 대해 유효성을 유지하기 위한 방법을 제공하더라도 효용성보다 이를 유지하기 위한 오버헤드가 크다.

3. 웹 어플리케이션 전송 방법을 이용한 웹 어플리케이션 서버의 동적 확장 기법

본 장에서 동적 컨텐츠의 요청을 처리하는 웹 어플리케이션을 전송하기 위한 구조를 제안하고, 주요한 구성 요소들을 설명한다.

3.1 개 요

웹 어플리케이션 전송 네트워크상에서 사용자가 요청한 서비스를 수행하는 간단한 시나리오는 다음과 같다(그림 3).

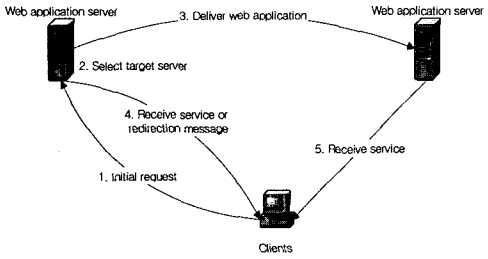


그림 3 사용자 시나리오

① 웹 어플리케이션 서버는 클라이언트로부터 정적 및 동적 컨텐츠에 대한 HTTP(Hyper Text Transfer Protocol)을 통해 요청를 받는다. 일반적으로 동적 컨텐츠에 대한 요청은 사용자 입력 정보 또는 쿠키 정보를 포함하게 된다.

② 첫 요청를 받은 서버는 자신의 부하 상황을 파악한 후, 미리 정해진 정책(policy)에 따라 전송 여부에 대한 결송을 내리게 된다. 만약, 전송을 결정하게 되면 클라이언트의 정보를 가지고 대상 서버를 결정하게 된다. 즉, 클라이언트에게 가장 빠른 응답을 제공하는 서버를 대상 서버로 설정한다.

③ 근원 서버는 동적 컨텐츠에 필요한 웹 어플리케이션을 묶은 뒤, 대상 서버로 전송한다. 대상 서버는 웹 어플리케이션을 받아 배치한 뒤, 근원 서버에 결과를 전해준다.

④ 근원 서버는 자신의 요청을 대상 서버로 리데이렉트시킨다.

⑤ 클라이언트는 리다이렉트 메세지를 받은 후, 대상 서버로부터 서비스를 받는다.

3.2 아키텍쳐

앞에서 설명한 기능을 제공하기 위해서는 먼저 각 서버의 부하를 측정하고 이를 유지하면서 실제 서비스를 제공하는 대상 서버를 결정하는 모듈이 필요하게 된다. 또한, 대상 서버가 필요한 웹 어플리케이션을 커스터마이징하여 빠르고 안전하게 전송하기 위해 하나로 묶어 전송하는 모듈이 필요하다. 그림 4은 웹 어플리케이션 전송 기법을 적용한 웹 어플리케이션 서버 아키텍쳐를 보여준다. 간단히 설명하면, system monitor는 각 서버의 부하를 측정하고, Converter는 웹 어플리케이션을 커스터마이징하고, Packer/Unpacker는 하나로 묶는 역할을 하며, Deliverer는 대상 서버로 전송하는 하게 된다. 그리고, Delivery Manager는 각 모듈간의 코디네이터로써 동작하게 된다.

클라이언트가 컨텐츠를 요청하면, 요청 처리기는

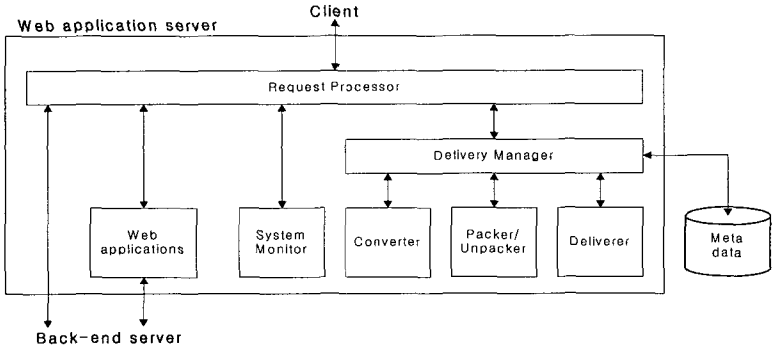


그림 4 웹 어플리케이션 전송 가능한 서버 아키텍쳐

요청된 컨텐츠의 종류를 구분한다. 정적 컨텐츠일 경우, 웹 서버처럼 디스크로부터 필요한 컨텐츠를 받아서 서비스한다. 하지만 동적 컨텐츠일경우, 요청 처리기는 서버의 부하 상태에 따라 직접 서비스를 할 것인지 아니면 다른 서버로 요청을 리다이렉션(redirection)할 것인지를 결정한다. 서버의 부하 상태는 시스템 모니터링을 위한 에이전트로부터 얻는다. 직접 서비스를 제공할 경우 요청 처리기는 요청된 동적 컨텐츠를 생성하는 서버 어플리케이션을 호출한다. 호출된 서버 어플리케이션은 필요에 따라 백엔드 서버로부터 데이터를 접근한다. 다른 서버로 요청을 리다이렉션할 경우, 요청처리기는 전송 매니저를 호출한다. 전송 매니저는 실제로 서비스를 제공하는 서버를 결정한 뒤, 필요한 웹 어플리케이션의 존재여부를 메타데이타로부터 얻는다. 웹 어플리케이션의 전송이 요구되면 전송 매니저는 필요한 웹 어플리케이션을 Converter, Packer를 통해 가공한 뒤 Deliverer가 전송하게 된다. 전송 매니저가 수행된 다음 요청 처리기는 클라이언트에 리다이렉션 정보를 보내 다른 서버로부터 서비스를 받게 한다. 이와 함께, 새로운 웹 어플리케이션을 전송 받은 서버는 서비스를 제공할 수 있도록 전송 매니저를 통해 필요한 설정을 변경한 뒤 배치한다.

그림 5는 웹 어플리케이션 서버에 웹 어플리케이션 전송 기법을 적용하였을 때, 클라이언트의 요청 처리에 대한 흐름도를 보여준다. 클라이언트의 요청이 들어오면, 먼저 서버는 시스템의 부하에 따라 전송 여부를 결정한다. 만약, 서버의 부하가 크지 않으면 직접 요청을 처리하게 된다. 하지만, 서버의 부하가 크면 먼저 클라이언트의 요청을 가장 효율적으로 처리할 수 있는 대상 서버를 결정하게 된다. 이 때, 대상 서버가 클라이언트의 요청을 처리하기 위한 웹 어플리케이션을 가지고 있다면, 바로 리다이렉션을 통하여 대상 서버로부터 서비스를 제공 받도록 한다. 반대로 가지고 있지 않으면, 전송 매니저는 클라이언트의 요청을 처리하기 위해 필요한 웹 어플리케이션을 구성하고 있는 요소들을 찾는다. 그런 다음, 전송 매니저는 Converter, Packer, Deliverer를 차례로 호출한다.

Converter는 이들을 대상 서버의 환경에 적합하도록 필요한 설정을 변경하게 되고, Packer는 이들을 하나로 묶는다. Deliverer는 이 패키지를 대상 서버로 전송하게 된다. 여기에서 클라이언트의 요청을 2가지 방법으로 처리할 수 있다. 한가지 방법은 클라이언트에게 빠른 응답 시간을 제공하기 위해서 전송 매니저가 동작하는 동안 직접 클라이언트에게 서비스를 제공하는 것이다. 또 다른 방법은 전송 매니저가 모든 일을 처리한 후에 대상 서버로 클라이언트의 요청을 리다이렉션하는 것이다.

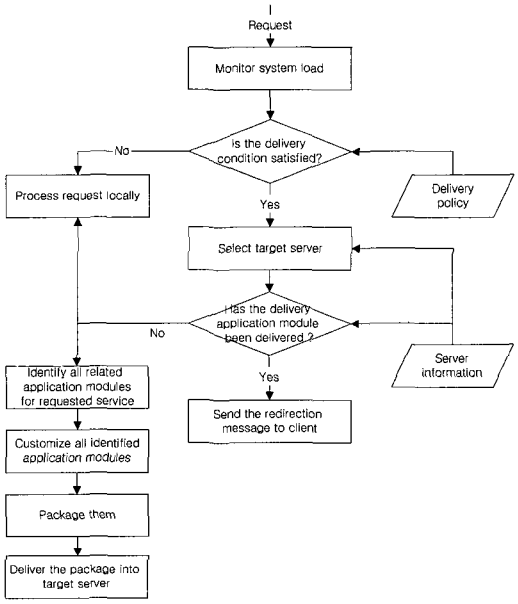


그림 5 처리 흐름도

3.3 구성요소

3.3.1 Converter

Converter는 웹 어플리케이션을 대상 서버에게 전달하기 위하여, 그 서비스 환경에 맞도록 변환한다. 일반적으로 웹 어플리케이션은 서비스를 제공하기 위해 외부 자원들을 요구한다.

예를 들어, 데이타베이스, 화일 서버 또는 디렉토리 서버등을 접근한다. 이 경우, 외부 자원을 접근하기 위한 코드를 대상 서버의 환경에 맞도록 수정하여야 한다. 그러나, 웹 어플리케이션이 외부 자원을 사용하지 않고 독립적으로 수행하면서 서비스를 제공할 경우는 문제가 되지 않는다.

Converter의 동작 방법은 다음과 같이 2가지가 있을 수 있다.

첫 번째, 소스 코드중에서 외부 자원를 접근하는 부분을 찾아서 자원에 대한 이름을 대상 서버의 환경에 맞도록 변환하는 것이다. 이는 이미 개발되어 사

용되고 있는 웹 어플리케이션을 전송하기 위해 필요하다.

두 번째, 어플리케이션 개발 단계에서 사용되는 외부자원에 대한 이름을 소스 코드내에서 고정된 값으로 지정하지 않고, 환경 변수를 통하여 얻을 수 있도록 명명 인터페이스(Naming interface)를 이용하는 것이다. 명명 인터페이스는 어플리케이션이 수행중 필요한 외부 자원들을 환경과 독립적으로 접근하도록 실행 시 동적 바인딩을 제공한다. 이렇게 함으로써, 웹 어플리케이션 내에 시스템 의존적인 부분을 제거할 수 있다.

3.3.2. Packer

하나의 서비스를 제공하는 웹 어플리케이션은 일반적으로 여러 개의 구성요소들로 구성되어 있다. 예를 들어, 서버 어플리케이션, HTML 문서와 이미지들이다. 특히, 하나의 서비스를 제공하기 위해서 웹 어플리케이션들을 연결하기도 한다. Packer는 이들을 효율적으로 전송하기 위해 하나로 묶는다. 이때, 어떻게 패키지해야 할 요소들을 찾아내는 것이 해결해야 할 문제이다. 가능한 방법을 살펴보면 다음과 같다.

첫 번째, 웹 어플리케이션 코드를 분석하여 필요한 구성요소들을 찾는 것이다. 이것은 Converter의 첫번째 방법과 마찬가지로 소스 코드를 직접 파싱(parsing)하여 연관된 코드를 분석하여 구성 요소를 찾아내는 소스 분석 기법으로 가능하다.

두 번째, 어플리케이션 개발 단계에서 관계된 구성 요소들을 패키지하여 웹 어플리케이션 서버에 배치(deploy)하는 것이다. 이 경우, 매번 전송 할 때마다 관계된 요소들을 찾기 위한 오버헤드(overhead)가 없게 된다.

3.3.3 Deliverer

Deliverer는 Packer에 의해 하나로 패키지되어 있는 웹 어플리케이션을 다른 웹 어플리케이션 서버에게 전달하게 된다. 이 때, 다른 서버에 있는 웹 어플리케이션과의 일관성(consistency)을 유지하기 위한 유효성 검사 기술(validation checking mechanism)이 필요하다. 이것은 TTL(Time To Live)를 설정하거나 버전(version) 번호를 통해 가능하다.

만약 TTL을 설정하게 되면 전송된 웹 어플리케이션이 수행될 때 TTL 값을 보고 서비스를 제공한다. TTL은 시간에 따라 계속 감소하게 되는데, 0가 되면 근원 서버로부터 새로운 웹 어플리케이션을 가져온다. 버전 번호를 사용할 경우, 클라이언트의 요청을 리다이렉션할 때 이미 전송된 웹 어플리케이션의 버전 번호가 이전 버전이면 근원 서버가 새로운 웹 어플리케이션을 전송한다.

위의 유효성 검사 기술 및 전송 서비스 정책과 관련하여 웹 어플리케이션의 전송 방법으로 다음과 같은 2가지가 존재한다.

① Push-based delivery

근원(Origin) 웹 어플리케이션 서버의 전송 정책에 따라 대상 서버로 웹 어플리케이션을 전송하는 것을 말한다.

② Pull-based delivery

대상 웹 어플리케이션 서버가 근원 서버에게 필요한 웹 어플리케이션을 요청하여 받는 것을 의미한다. Pull-based delivery는 클라이언트로부터 받은 요청을 처리하기 위해 필요한 웹 어플리케이션이 존재하지 않을 경우에도 사용될 수 있다.

3.3.4 System monitor

동적 컨텐츠를 처리하는 것은 서버 측의 부하를 외부에서 예측하기 어렵게 만든다. 단순히 정적 컨텐츠를 처리하는 웹 서버의 경우 각 클라이언트의 요청을 처리하는 단위 시간당 서버가 받는 부하가 비교적 일정하다고 볼 수 있다. 이에 반해, 동적 컨텐츠을 서비스하기 위해서 웹 어플리케이션 서버가 받는 부하는 사용자 서비스 요구에 따라 웹 어플리케이션마다 다양하다.

Monitor는 서버와 네트웍과 같이 서비스를 제공하기 위해 사용되는 모든 자원들의 부하를 파악한다. 특히, 웹 어플리케이션 서버에 존재하는 Monitor는 클라이언트의 요청을 처리할 때, 리다이렉션을 적용하기 위해 시스템의 부하를 측정한다. 이 때, 서버측의 부하를 측정하기 위해서 CPU 점유율, 부하 평균(load average)등을 포함할 수 있다. CPU 점유율은 현재 클라이언트의 요청을 처리하기 위해 서버측의 CPU 이용도를 나타낸다. 부하 평균은 단위 시간당 큐에서 수행되고 있는 평균 일(job)의 개수로서 서버의 부하를 예측하는데 사용할 수 있다.

시스템 내부의 자원들에 대한 활용도(utilization) 이외에도, 특정한 URL에 대해 request를 보낸 후 측정된 응답시간(response time)을 가지고 시스템의 부

하를 예측할 수 있다. 일반적으로 이것은 미리 정해진 응답시간에 대한 제약(constraint)을 가지고 이루어 진다. 예를 들어, 중간에 다른 중계장치(예를 들어, 라우터)가 없다고 가정할 때, 응답시간에 대한 제약이 3초인 웹 컨텐츠가 4초 만에 도착했다면 시스템에 많은 부하가 존재한다고 알 수 있다.

3.3.5 Meta information

요청처리기가 클라이언트의 요청에 대한 리다이렉션 여부를 결정하거나, 전송 매니저가 웹 어플리케이션을 대상 서버로 전송할 때 다양한 메타 데이터가 필요하다. 메타 데이터는 데이터베이스나 파일에 저장될 수도 있지만, 데이터의 속성상 읽기를 위한 접근이 많기 때문에 LDAP 서버와 같은 디렉토리 서버에 유지되어 전송 매니저와 통신하도록 할 수 있다.

① Redirection policy

리다이렉션 정책은 클라이언트의 요청을 직접 서비스할 것인지 다른 서버를 통해 서비스를 제공하도록 할 것인지를 결정하게 한다. 이 때 적용 가능한 정책을 살펴보면 다음과 같다.

첫 번째, 서버의 부하를 측정하는 각각의 값들에 대해 임계값(threshold)을 설정해 두는 것이다. 일반적으로 임계값은 서버에 많은 부하가 존재하지 않도록 설정한다. 만약, 서버의 부하가 임계값을 넘어서면 웹 어플리케이션의 전송을 수행하지만 요청된 동적 컨텐츠는 직접 서비스를 제공하도록 한다.

두 번째, 서버에 많은 부하가 존재하여 클라이언트 요청에 대한 서비스가 지연될 경우 다른 서버로 요청을 리다이렉션 한다.

② Server information

서버 정보는 웹 어플리케이션을 전송하여 클라이언트의 요청을 서비스할 수 있는 서버들의 리스트 및 각 서버에 전송되어 있는 웹 어플리케이션들에 대한 정보를 의미한다. 이 외에도 각 서버의 지리적인 위치나 현재 서버의 부하에 대한 정보를 포함하기도 한다. 위와 같은 정보는 클라이언트에게 빠르게 서비스를 제공하기 위한 대상 서버를 결정하는데 사용되어진다.

4. 구 현

4.1 구현환경

자바(JAVA) 언어를 사용하여 구현된 웹 어플리케이션을 전송하기 위하여, JAVA 2 Enterprise 플랫폼(platform)을 기반으로 구현하였다. 즉, JAVA 2 Enterprise 어플리케이션을 만들기 위한 Enterprise JAVA Bean(EJB), JAVA Naming and Directory Interface(JNDI), JAVA Database Connectivity (JDBC), 압축 포맷 등을 사용하였다.

이 장에서는 3장에서 설명된 각 구성 요소들에 대한 구현 방법을 기술한다.

4.2 Converter

Converter의 구현은 다음과 같이 2가지 방법을 포함한다.

첫 번째, 소스 코드 중에서 외부 자원을 접근하는 부분을 찾아낸 뒤, 자원을 접근하기 위한 이름을 대상 서버의 환경에 맞도록 변경하는 것이다. 예를 들어, 데이터베이스, 파일 서버 또는 디렉토리 서버 등을 접근하게 된다. 이 경우, 웹 어플리케이션 소스 코드 안에 외부 자원을 접근하는 코드를 대상 서버의 환경에 맞게 수정하게 된다. 예를 들어, JAVA 언어로 만들어진 서버 어플리케이션이 데이터베이스를 접근한다면 다음과 같은 코드를 포함하게 된다.

```
DriverManager.getConnection(url);
```

여기서 url은 다음과 같은 포맷을 가진 문자열이다.

```
url := jdbc:subprotocol:subname
```

여기서 subprotocol과 subname에 대한 포맷은 사용되는 드라이버의 벤더(vendor)에 따라 다양하므로 url이 다르게 된다. 만약, 웹 어플리케이션이 다른 서버로 전송되었을 때 접근하는 데이터베이스가 변경될 경우, 위의 url이 바뀌게 된다. 따라서, Converter는 url을 설정하는 부분을 찾아 대상 서버의 데이터베이스 url로 변경하게 된다. 이 방법은 기존에 개발된 웹 어플리케이션의 전송을 지원하기 위해 필요하다.

두 번째, 웹 어플리케이션 개발 단계에서 외부 자원에 대한 이름을 소스 코드 내에서 고정하여 사용하지 않고, 환경 변수를 통하여 얻을 수 있도록 네이밍 인터페이스(naming interface)를 이용하는 것이다. 네이밍 인터페이스는 웹 어플리케이션이 수행 중에 필요한 외부 자원들을 투명하게 접근하도록 실행 중에 동적 바인딩(dynamic binding)을 제공한다.

따라서, 웹 어플리케이션 소스에서 시스템에 특정한 코드를 제거할 수 있다. 예를 들어, Java Naming and Directory Interface(JNDI)를 사용하여 데이터베이스를 접근하는 방법을 살펴보면 다음과 같다.

① 프로그램 코드

```
DataSource ds=(DataSource)initCtx.lookup("java:
comp/env/jdbc/AccountDB");
Connection con=ds.getConnection();
```

② 프로퍼티(property) 파일

```
jdbc.datasources=jdbc/Oracle|jdbc:oracle:thin@rt
c:1521:acct
```

위와 같이, 프로그램 코드는 데이터베이스를 접근하기 위해 데이터베이스의 논리적인 이름을 사용한다. 논리적인 이름에 대한 실제 url은 프로퍼티 파일에 존재한다. 따라서 프로그램 코드는 실행 중에 데이터베이스에 대한 실제 url을 얻게 된다. 이 경우, Converter가 전송된 웹 어플리케이션의 프로퍼티 파일을 변경함으로서 외부 자원을 접근 하는 코드를 쉽게 변경하게 된다.

4.3 Packer

JAVA로 구성된 서버 어플리케이션을 하나로 묶기 해서는 다음 2가지 방법이 존재한다.

첫 번째, 웹 어플리케이션 소스 코드를 분석하여 필요한 요소들을 찾는 것이다.

이것은 Converter의 구현에서 언급된 첫 번째 접근 방법과 유사한 소스 분석 기술(source trace mechanism)을 구현하여 가능하다.

예를 들어, 다음과 같이 JAVA 언어로 만들어진 서버 어플리케이션을 고려해 보자.

```
RequestDispatcher rd=this.getServletContext().
getRequestDispatcher(url );
rd.include(req, res);
```

서버 어플리케이션에 위와 같은 코드가 존재하면, 여러 개의 서버 어플리케이션이 체인(chain)되어 하나의 서비스를 제공하게 된다. 따라서, 소스 분석 기술을 이용하여 url이 지시하는 서버 어플리케이션을 찾는다.

두 번째, 웹 어플리케이션 개발 단계에서 관계된 요소들을 패키지하여 배치하는 것이다. 이 경우, 매번 전송할 때마다 관계된 요소들을 찾을 필요가 없게 된다. 예를 들어, JAVA 언어로 구성된 웹 어플리케이션을 WAR(Web Archive) 포맷으로 묶어서 전송할 수 있다.

4.4 Deliverer

HTTP PUT 요청 방식을 이용하여 미리 패키지된 웹 어플리케이션을 대상 서버로 전송하게 된다. HTTP PUT 요청 방식을 사용할 경우 효율적이고 손쉽게 전송할 수 있을 뿐만 아니라 차후에 HTTPS 프로토콜을 사용하게 될 경우 안전하게 전송할 수 있다는 장점이 있다.

5. 실 험

5.1 실험 환경

실험은 웹 서비스를 제공하는 사이트를 구축한 뒤, 클라이언트의 요청의 수에 따라 서버측의 성능을 평가하게 된다. 실험에서의 성능 평가 요소로는 서비스 요청에 따른 응답 시간, 서버측의 throughput, 서비스 에러 개수를 사용하였다.

먼저, 웹 서비스를 제공하는 사이트는 Transaction Processing Performance Council의 TPC-W 벤치마크에서 정의된 인터넷 온라인 서점을 구축하였다. 이것은 JAVA 언어의 JAVA Servlet을 사용하여 구현되었고, TPC-W 1.0.1 버전의 specification을 기반으로 하였다. 구현된 JAVA Servlet 모듈을 서비스하는 웹 어플리케이션 서버는 아파치 웹 서버(버전 1.2) 와 JAVA Servlet 및 JAVA Server Pages(JSP)에 대한 오픈 소스 구현 모듈인 TOMCAT을 사용하였다. 데이타베이스 서버는 Oracle 8.1.7 버전을 사용하였다. 각 서버는 Pentium3 1GHz, 512M memory의 시스템 사양을 가지고, 100Mbps의 local area network에 연결되어 있다.

실험을 위해 사용되는 서버가 제공하는 서비스 방법은 2가지로 나누어진다.

첫번째, 기존에 서비스를 제공하는 방법으로, 단순히 한 개의 웹 어플리케이션 서버로부터 서비스를 제공하는 것이다. 이것은 하나 또는 여러 개의 서버를 클러스터링하여 하나의 시스템처럼 동작하도록 하는 환경을 포함한다.

두번째, 앞에서 설명된 웹 어플리케이션 전송 방식을 통해 동적으로 웹 어플리케이션 서버를 확장하여 서비스를 제공하는 것이다. 이것은 컨텐츠 제공자가 웹 어플리케이션 서버의 풀을 유지하는 곳과 미리 계약을 맺어 서비스를 제공하는 것을 말한다. 여기서, 클라이언트가 단순히 동적 페이지에 대한 요청만을 리다이렉션하거나, 모든 동적 및 정적 페이지에

대한 요청을 리다이렉션하는 2가지 방법으로 실험을 진행하였다.

실험은 기본적으로 웹 어플리케이션 서버에 많은 서비스 요청에 의한 과부하 상태에서 서버의 성능을 측정하게 된다. 여기에서 서버의 성능은 TPC-W 벤치마크의 Remote Browser Emulator(RBE)를 사용하여 클라이언트의 행동을 에뮬레이션(emulation)하는 것이다.

이를 이용하여, TPC-W에서 시스템을 과부하 테스트하는 방법에 맞추어 실험을 수행하였다.

RBE를 수행할 때 주어지는 think time(TT) 값을 0로 설정하였고, RBE가 생성하는 EB(Emulated Browser)의 수를 200개부터 100개씩 늘리면서 시스템의 부하를 증가시켰다.

그리고, 총 30분동안 실험을 수행하면서 앞뒤 10분을 제외한 서버의 부하가 가장 큰 기간은 측정기간(Measurement interval)로 잡았다. 클라이언트의 리다이렉션은 초기 서비스 화면에서 이루어지며, 리다이렉션하는 시점은 응답 시간이 2초를 넘어서면 진행되도록 하였다.

실험에서 사용된 웹 인터렉션(Web interaction)의 종류는 TPC-W 사양에 있는 Shopping mix이다.

모든 실험 결과는 같은 환경하에서 5번씩 수행한 다음, 가장 좋은 결과와 가장 나쁜 결과를 제외한 3번의 결과를 평균하여 보여준다.

5.2 TPC-W benchmark을 이용한 실험 결과

여기서 3가지 서로 다른 서비스 프레임워크상에서 실험을 수행하고 그 결과를 비교하였다.

① Exist : 기존에 존재하는 서비스 프레임워크

② WADN : 모든 정적 및 동적 컨텐츠에 대한 요청을 대상 서버로 리다이렉션하는 서비스 프레임워크

③ WADN2 : 단순히 동적 컨텐츠에 대한 요청만 대상 서버로 리다이렉션하고 정적 컨텐츠는 근원 서버로부터 서비스 받는 프레임워크

위의 실험 결과에서 응답 지연으로 인해 총 인터렉션(interaction)의 수에 차이가 발생하게 된다.

각각의 EB가 동작할 때 think time을 0로 설정했기 때문에 서버로부터 응답을 받는 즉시 다음 요청을 보내기 때문이다.

결과에 나타나듯이 Exist는 에러가 없을지라도 응답 시간은 상당히 큰 값을 가지는 반면, WADN는 빠른 응답 시간으로 서비스를 제공함을 알 수 있다. WADN2도 static contents를 근원 서버로부터 서비스 받기 때문에 전체 응답 시간이 WADN보다 크지만 Exist에 비해서는 상당히 작은 값을 가지게 됨을 알 수 있다.

표 1 200개의 EB를 이용한 테스트

측정 기준	Exist	WADN	WADN2
총 interactions 수	27039	51689	46965
하위 90%에 해당하는 WIRT(msec)	53845	3645	4370
하위 90% 평균 WIRT(msec)	7120	1987	2197
연결 에러의 수	0	0	0
리다이렉션의 수	0	23458	20437

표 2 300개의 EB를 이용한 테스트

측정 기준	Exist	WADN	WADN2
총 interactions 수	22846	48082	45447
하위 90%에 해당하는 WIRT(msec)	68125	5906	31275
하위 90% 평균 WIRT(msec)	14444	2490	3315
연결 에러의 수	16	0	9
리다이렉션의 수	0	22260	20428

표 2의 결과는 첫 번째 실험보다 서버측에 훨씬 많은 부하를 주었을 경우 서비스 상태를 보여주고 있다.

여기서 주목해야 할 것 중에 하나는 connection error의 개수이다.

즉, Exist에서 발생한 connection error가 WADN에서는 전혀 발생하지 않고 안정된 서비스를 제공한다는 것이다.

표 3 400개의 EB를 이용한 테스트

측정 기준	Exist	WADN	WADN2
총 interactions 수	22261	47484	47433
하위 90%에 해당하는 WIRT(msec)	85345	15183	43640
하위 90% 평균 WIRT(msec)	16188	2503	3991
연결 에러의 수	104	31	34
리다이렉션의 수	0	23646	21556

표 3은 400개의 EB를 가진 RBE를 수행했을 경우 나타난 결과이다.

WADN이나 WADN2의 평균 응답시간이 Exist보다 2배이상 높은 값을 가지는 것을 알 수 있다. 또한 서비스 에러의 수도 획기적으로 감소하였고, WADN이나 WADN2에서 발생한 에러는 모두 근원 서버에서 발생하였다.

6. 결 론

본 논문은 동적 웹 컨텐츠를 효율적으로 서비스하기 위해 웹 어플리케이션 전송 네트워크를 구성하여 사용자의 요구에 따라 서비스의 확장성을 동적으로 제공하는 방법을 제안하였다.

즉, 클라이언트의 수에 따라 웹 어플리케이션 서버의 확장성과 가용성을 동적으로 변화시키고, 시스템의 부하에 따라 동적으로 부하를 분산시킬 수 있다. 결국, 클라이언트는 서버로부터 빠르고 안정적인 응답 시간을 보장 받게 되어 항상 최상의 서비스를 제공 받을 수 있다.

기존의 웹 서비스 제공자는 미리 클라이언트의 수요를 예측하기 힘들기 때문에 서버를 효율적으로 유지하기 어려웠다. 우리가 제안한 방법을 이용할 경우 서버 개수를 동적으로 변화시켜 서비스 리소스의 활용도를 높일 수 있게 되고, 이에 따라 총 소유비용(Total Cost of Ownership, TCO)을 획기적으로 줄일 수 있게 된다.

향후 연구 과제로는 제안된 아키텍쳐를 기반으로 구현의 완성도를 높이고, 실제 인터넷 환경 또는 유사한 환경에 적용하여 그 파생 효과를 검증하는 연구가 필요하다. 그리고, 보안 문제도 추가로 고려되어야 할 사항이다. 실제로 웹 어플리케이션은 어떤 컨텐츠 제공자의 비지니스 로직이 담겨 있기 때문에, 이를 효율적으로 전송되고 관리되도록 하여야 할 것이다. 또한, 기존의 CDN(Contents Delivery Network)과 같이 정적 컨텐츠만을 인터넷에 복제하여 두는 서비스 방법과의 통합하여 적용할 수 있는 방법에 관한 연구도 이루어져야 할 것이다. 서버 풀을 효과적으로 유지하는 방법 또한 추가로 고려해야 할 사항이다.

참고문헌

[1] Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A Scalable and Highly Available Web Server. Proc. of IEEE COMPCON, 1996.

[2] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Dynamic load balancing on Web server systems. Proc. of IEEE Internet Computing, 1999.

[3] Arun Iyengar, Ed MacNair, and Thao Nguyen. An Analysis of Web Server Performance. Proc. of IEEE GLOBECOM conference, 1997.

[4] Jia Wang. A Survey of Web Caching Schemes for the Internet. Proc. of ACM Computer Communication Review, 1999.

[5] Greg Barish and Katia Obraczka. World Wide Web Caching: Trends and Techniques. Proc. of IEEE Communications Magazine Internet Technology Series, 2000.

[6] Akamai Technologies, Inc. Content Delivery Services. http://www.akamai.com, 2001.

[7] Qiong Luo, Jeffrey F. Naughton, Rajesekar Krishnamurthy, Pei Cao, and Yunrui Li. Active Query Caching for Database Web Servers. Proc. of the International Workshop on Web and databases, 2000.

[8] K. Selcuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. Proc. of ACM SIGMOD, 2001.

[9] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process Migration. ACM Survery, 1999.

[10] Mark Nettall et al. Workload characteristics for Process Migration and Load Balancing. Proc. of International Conference on Distributed Computing Systems, 1997.

[11] Sun Microsystems, Inc. J2EE Platform Specification. http://java.sun.com/j2ee/

[12] David Mosberger and Tai Jin. httperf A Tool for Measuring Web Server Performance. Proc. of First Workshop on Internet Server Performance, 1998.

[13] TPCW2001. Transaction Processing Performance Council(TPC). TPC Benchmark W (Web Commerce) Specification. http://www.tpc.org/tpcw/

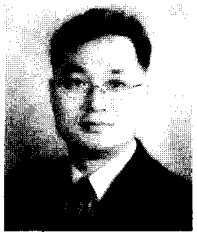
임 민 열

2000 경북대학교 컴퓨터공학과 학사
2002 한국과학기술원 전산학과 석사
현재 한국과학기술정보연구원 슈퍼컴퓨팅연구실 연구원
E-mail:mylim@dbserver.kaist.ac.kr

김 정 숙

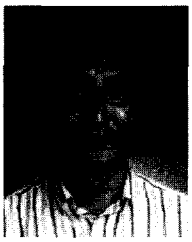
2000 한국과학기술원 전자전산학과 학사
2000~현재 한국과학기술원 전자전산학과 석사 과정
E-mail:sun@nclab.kaist.ac.kr

송 준 화



1988 서울대학교 계산통계학과 학사
1990 State University of NY, Computer Science, 석사
1997 University of Maryland, Computer Science, 박사
1994~1996 IBM T.J. Watson Research Center, S/W Engineer, Digital Library Group
1996~1997 IBM T.J. Watson Research Center, S/W Engineer, Digital Media Solutions Dept.
1997~2000 IBM T.J. Watson Research Center, Research Staff Member, Parallel Commercial Systems Dept.
2000~현재 한국과학기술원 전자전산학과 조교수
E-mail:junesong@nclab.kaist.ac.kr

이 윤 준



1977 서울대학교 계산통계학과 졸업.
1979 한국과학기술원 전산학과에서 석사학위 취득
1983 France, INPGEN- SIMAG에서 박사학위 취득
1983~1984 France, IMAG 연구원.
1984~현재 한국과학기술원 전산학과 부교수
1989 MCC(미) 초빙연구원
1990 CRIN(불) 객원교수
E-mail:yjlee@dbserver.kaist.ac.kr