DOI: 10.3745/KIPSTA.2008.15-A.4.199

분산형 레이어 7 서버 부하 분산

권 희 웅[†] · 곽 후 근[††] · 정 규 식[†††]

요 약

무선 인터넷 프록시 서버 클러스터링에서 저장공간을 최소화하기 위해서는 URL 해싱기법을 가진 Layer 7 부하분산기가 필요하다. 서버 클러스터 앞단에 위치한 Layer 4 부하분산기는 TCP 또는 UDP와 같은 트랜스포트 계층에서 컨텐츠 내용을 확인하지 않고 사용자 요청들을 똑같은 컨텐츠를 가진 서버들에게 분배한다. 서버 클러스터 앞단에 위치한 Layer 7 부하분산기는 응용계층에서 사용자 요청을 분석하여 요청 컨텐츠 유형에 따라 해당되는 서버들에게 분배한다. Layer 7 부하분산기를 이용하면 서버들이 배타적으로 각기 다른 컨텐츠를 가지게 할 수 있어서 서버들 저장공간을 최소화할 수 있으며 전체 클러스터 성능을 향상할 수 있다. 그러나 Layer 7 부하분산기는 응용계층에서 사용자 요청을 분석하는데 요구되는 큰 처리 부담으로 인해 Layer 4 부하분산기와 다르게 확장성이 제한된다.

본 논문에서는 그 확장성 제한을 극복하기 위해서 분산형 Layer 7 부하분산기를 제안한다. 종래의 방법에서는 한 대의 Layer 7 부하분산기를 사용하는데 본 논문에서 제안한 방법에서는 서버 클러스터 앞에 한 대의 Layer 4 부하분산기를 설치하고 서버들에게 Layer 7 부하분산기들을 각각 설치한다. 클러스터 기반의 무선 인터넷 프록시 서버에서 종래의 방법을 리눅스기반의 Layer 7 부하분산기인 KTCPVS를 이용하여 구현하였다. 본 논문에서 제안한 방법에서는 리눅스기반의 Layer 4 부하분산기인 IPVS를 사용하고 각 서버들에게 Layer 7 부하분산기인 KTCPVS를 설치하여 같이 동작하게 구현하였다. 실험은 16대의 컴퓨터를 사용하여 수행되었고, 실험 결과에 의하면 제안 방법이 종래 방법에 비해 서버 대수가 증가함에 따라 확장성 및 높은 성능 향상률을 가짐을 확인하였다.

키워드 : 클러스터링, 해싱, Layer-4 부하분산기, Layer-7 부하분산기, 확장성

A Distributed Layer 7 Server Load Balancing

Huiung Kwon[†] · Hukeun Kwak[††] · Kyusik Chung[†††]

ABSTRACT

A Clustering based wireless internet proxy server needs a layer-7 load balancer with URL hashing methods to reduce the total storage space for servers. Layer-4 load balancer located in front of server cluster is to distribute client requests to the servers with the same contents at transport layer, such as TCP or UDP, without looking at the content of the request. Layer-7 load balancer located in front of server cluster is to parse client requests in application layer and distribute them to servers based on different types of request contents. Layer 7 load balancer allows servers to have different contents in an exclusive way so that it can minimize the total storage space for servers and improve overall cluster performance. However, its scalability is limited due to the high overhead of parsing requests in application layer as different from layer-4 load balancer.

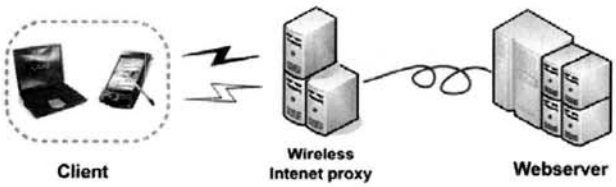
In order to overcome its scalability limitation, in this paper, we propose a distributed layer-7 load balancer by replacing a single layer-7 load balancer in the conventional scheme by a single layer-4 load balancer located in front of server cluster and a set of layer-7 load balancers located at server cluster. In a clustering based wireless internet proxy server, we implemented the conventional scheme by using KTCPVS(Kernel TCP Virtual Server), a linux based layer-7 load balancer. Also, we implemented the proposed scheme by using IPVS(IP Virtual Server), a linux-based layer-4 load balancer, installing KTCPVS in each server, and making them work together. We performed experiments using 16 PCs. Experimental results show scalability and high performance of the proposed scheme, as the number of servers grows, compared to the conventional scheme.

Keyword : Clustering, Hashing, Layer-4 Load Balancer, Layer-7 Load Balancer, Scalability

1. 서 론

무선 인터넷에 대한 관심이 증가하는 가운데 핸드폰, PDA등의 무선 인터넷 단말기의 수요가 늘어나며 보편화 되어가고 있다. 그리고 무선 인터넷 서비스도 기존의 정보검색 위주의 간단한 서비스에서 전자 상거래나 멀티미디어 서

※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0) 지원 및 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 정 회 원 : 숭실대학교 전자공학과 대학원 박사과정
†† 정 회 원 : 숭실대학교 전자공학과 대학원 (postdoc) (교신저자)
††† 정 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2008년 4월 24일
수 정 일 : 2008년 6월 11일
심사완료 : 2008년 6월 11일



(그림 1) 무선 인터넷 프록시 서버

비스 등의 복잡한 서비스로 사용자들의 욕구가 상승하고 있다. 이에 따라 무선 인터넷 대역폭의 효율적인 사용과 빠른 이용 시간을 위하여 무선 인터넷 프록시 서버의 필요성이 증대되고 있다. 무선 인터넷 프록시 서버는 무선 사용자를 유선 인터넷 서버에 연결 시켜주는 역할을 한다. (그림 1)은 무선 인터넷에 사용되는 무선 인터넷 프록시 서버를 나타내고 있다.

무선 인터넷 프록시 서버는 급증하는 사용자의 요청에 대한 확장성(Scalability)을 보장하기 위해 클러스터링 구조를 가진다. 본 논문에서 확장성이란 호스트 개수가 늘어나는 것에 비례하여 서버의 성능이 늘어나는 것을 의미한다. 대용량 서비스로 갈 경우 서버의 확장성은 매우 중요한 이슈 중의 하나이다.

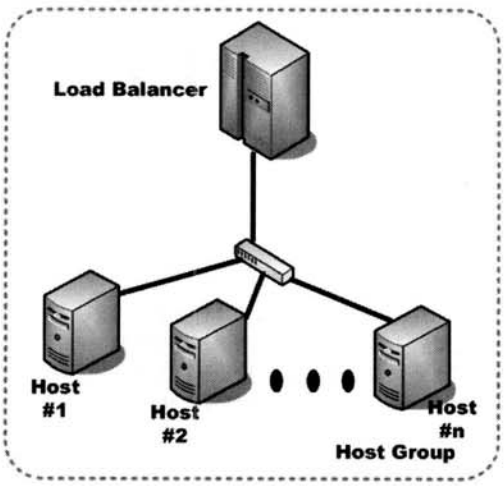
무선 인터넷 프록시 서버 클러스터는 여러 대의 서버를 하나의 서버처럼 동작하도록 연결함으로써 고성능 및 고가용성의 효과를 가진다. 무선 인터넷 프록시 서버를 클러스터링 하기 위해서는 서버들을 하나로 묶어주고 스케줄링을 해주는 부하 분산기(Load Balancer)[1] 및 저장 공간의 효율성을 위한 해싱 기법[2-6]을 필요로 한다.

1.1 무선 인터넷 프록시 서버

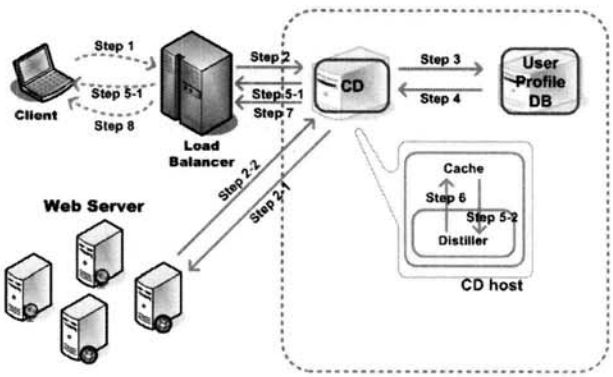
무선 인터넷 프록시 서버는 무선 인터넷의 낮은 대역폭을 해결하기 위해 캐싱(Caching)[7]과 압축(Distillation)[8]을 사용해야 하며, 대용량 트래픽에 대한 확장성(Scalability)이 고려되어야 한다. TranSend[9]는 대용량 트래픽에 대한 확장성을 고려하여 클러스터링으로 구현된 무선 프록시 서버이다. 본 논문에서는 무선 인터넷 프록시 서버 클러스터인 TranSend를 확장성과 구조적인 관점에서 개선한 부하분산기-캐시 구조[10]를 사용하였다. (그림 2)는 부하분산기-캐시 프록시 시스템의 전체적인 구조를 나타낸다.

부하분산기-캐시의 전체 구조는 (그림 2(a))와 같고 자세한 설명은 다음과 같다. 부하분산기-캐시 구조는 부하 분산을 위한 부하 분산기와 캐시 호스트들로 구성되며 호스트는 CD(Cache & Distiller)로 구성된다. 부하 분산기는 클라이언트의 요청을 받아서 각 호스트들에게 전달하는 역할을 담당하며, CD는 클라이언트의 요청을 처리하는 Cache와 데이터에 대한 압축을 수행하는 Distiller의 역할을 함께 수행한다.

부하분산기-캐시의 동작 과정은 (그림 2(b))와 같고 자세한 설명은 다음과 같다. 부하 분산기가 스케줄링 알고리즘을 통하여 클라이언트의 요청을 각 호스트에 전달하게 되면, 호스트 내의 CD는 해당 데이터가 Cache에 존재하면 부하



(a) 구조



(b) 동작 과정

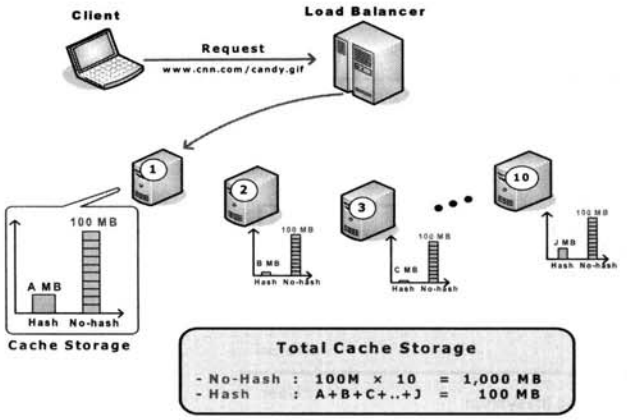
(그림 2) 부하분산기-캐시 프록시 시스템 구조

분산기에 전달하고, 존재하지 않으면 웹 서버로부터 데이터를 요청하여 얻어온 후 압축과정을 거쳐 부하 분산기에 전달하고 부하 분산기는 그 데이터를 클라이언트에 보내는 방법으로 동작한다.

1.2 해싱의 필요성

무선 인터넷 프록시 서버를 클러스터로 구성함에 있어 캐시간의 협동성을 보장하는 것은 클러스터 간 중복되는 데이터가 차지하는 공간을 줄이고, 효율적인 캐싱을 통해 사용자가 요청한 데이터를 빠르게 얻을 수 있게 한다. 캐시가 협동성을 가진다는 것은 캐시 서버 수에 무관하게 전체적인 캐싱 공간을 일정하게 할 수 있다는 것을 의미하며 캐시가 협동성을 가지지 않는 다면 각 캐시 서버가 모든 캐시 데이터를 가져야 하므로 전체 캐싱 크기가 서버 수에 비례하여 증가하는 것을 의미한다.

(그림 3)은 캐시간 협동성을 가지는 부하 분산 방식(해싱을 이용한 방법)과 가지지 않는 부하 분산 방식(해싱을 이용하지 않는 방법)의 캐시 저장 공간 비교를 나타낸다. 해싱 방법을 사용하면 해싱 방법을 사용하지 않는 방법에 비해 저장 공간이 상당히 줄어들 수 있음을 보여주고 있다. 예를



(그림 3) 캐시 서버의 저장 공간 비교

들어, 저장해야할 데이터가 크기가 총 100 Mbytes이고, 서버의 개수가 10대라면 해싱은 10대의 서버가 100 Mbytes의 데이터를 나누어 가지는 반면 해싱을 사용하지 않는 경우는 10대의 서버가 100 Mbytes를 중복저장하게 되어 총 1 GBytes(=100 Mbytes x 10대)의 데이터를 가지게 된다.

캐시간 협동성을 위해 널리 사용되는 캐시 선택 방법은 해싱(Hashing)을 이용한 방법이다. 즉, 사용자 주소(Source IP), 목적지 주소(Destination IP) 혹은 URL의 해쉬 값을 이용하여 캐시를 선택함으로써 동일 주소 혹은 URL에 대해 동일 캐시가 선택되는 것을 보장한다. (그림 4)는 무선 인터넷 프록시 서버 클러스터에서 캐시간 협동성을 나타낸다.

무선 인터넷 프록시 서버 클러스터에서 캐시간 협동성에 대한 전체적인 동작 과정은 다음과 같다.

- 사용자가 목적지 URL(www.daum.net)로 이미지를 요청한다.
- 부하 분산기가 사용자의 목적지 URL을 통해 해쉬 값을 생성한다.
- 해쉬 값을 통해 해당 해쉬 값에 할당된 캐시 서버로 이미지를 요청한다. (해쉬 값을 이용해서 캐시 서버를 할당하는 방식은 (캐시 서버 = 해쉬값 MOD 전체 캐시 서버의 수)를 이용한다.)
- 이후의 과정은 새로운 URL 요청에 대해서는 위의 과정을 반복하며, 동일 URL 요청에 대해서는 동일 캐시 서버로 데이터를 요청한다.

1.3 Layer-4 서버 부하분산기와 Layer-7 서버 부하분산기

(그림 2(a))에서 부하 분산기가 들어오는 패킷에 대해 어느 호스트로 전송할 것인지 결정함에 있어서 Layer 4 계층까지 정보를 참조하면 Layer 4 부하분산기라고 하고, Layer 7 계층까지 정보를 참조하면 Layer 7 부하분산기라고 한다.

1.3.1 Layer-4 서버 부하 분산

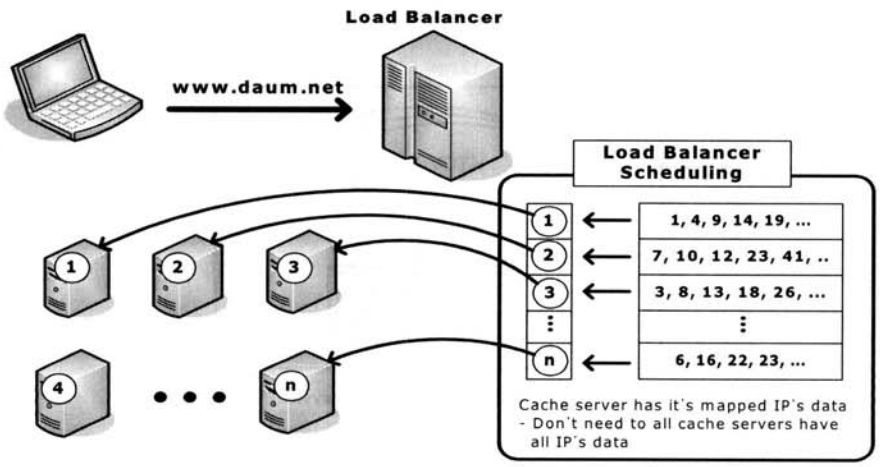
(그림 2(a))에서 호스트들은 동일한 컨텐츠를 보유한다고 가정하면, 부하 분산기는 들어오는 패킷에서 Layer 4 계층의 정보(IP 및 Port 정보)를 참조하여 해당 패킷을 어느 호스트로 전송할 것인지를 결정한다.

1.3.2 Layer-7 서버 부하 분산

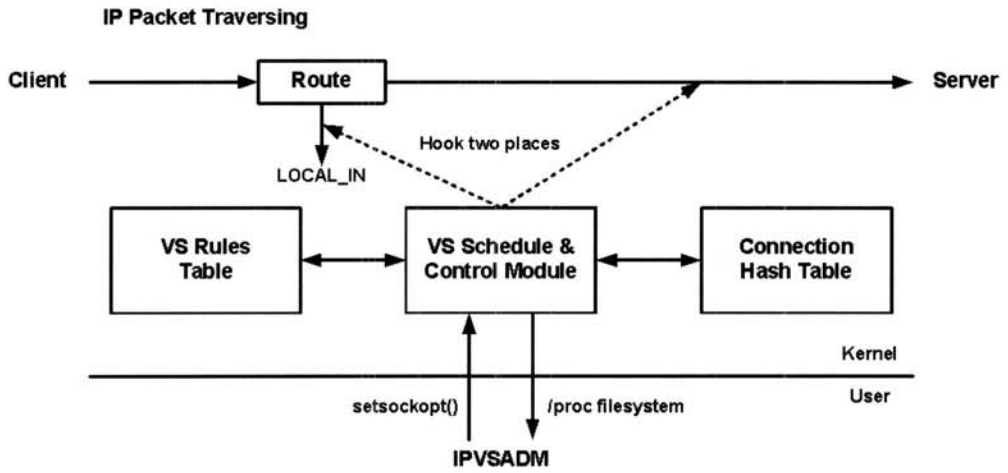
(그림 2(a))에서 호스트들은 서로 다른 컨텐츠를 보유한다고 가정하면, 부하 분산기는 들어오는 패킷에서 Layer 7 계층의 정보, 즉 패킷 내 Payload 정보(예를 들면 HTTP 헤더 정보)를 참조하여 해당 패킷을 어느 호스트로 전송할지를 결정한다.

L7 부하 분산은 L4 부하분산보다 Payload까지 보아야 하기 때문에 부하가 많이 걸린다. 그러나 호스트들이 동일한 컨텐츠를 가질 필요가 없기 때문에 정교한 부하 분산 방법이다. 예를 들어, L7 서버 부하 분산을 이용하여 텍스트 서버, 이미지 서버, 메일 서버 등 서버별로 컨텐츠를 구분하여 저장하면 저장 공간을 효율적으로 관리할 수 있다.

본 논문에서는 확장성 있는 부하 분산기를 위해 새로운 L7 부하 분산 구조를 제안한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 기존 부하 분산 구조인 IPVS(IP Virtual



(그림 4) 무선 인터넷 프록시 서버 클러스터에서 캐시 협동성



(그림 5) IPVS의 전체 구조

Server)와 KTCPVS(Kernel TCP Virtual Server)와 그 문제점을 소개한다. 3장에서는 기존의 구조에서 장점만을 이용하여 기존 구조의 단점을 해결하는 새로운 L7 부하 분산 구조를 제안한다. 4장에서는 실험을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

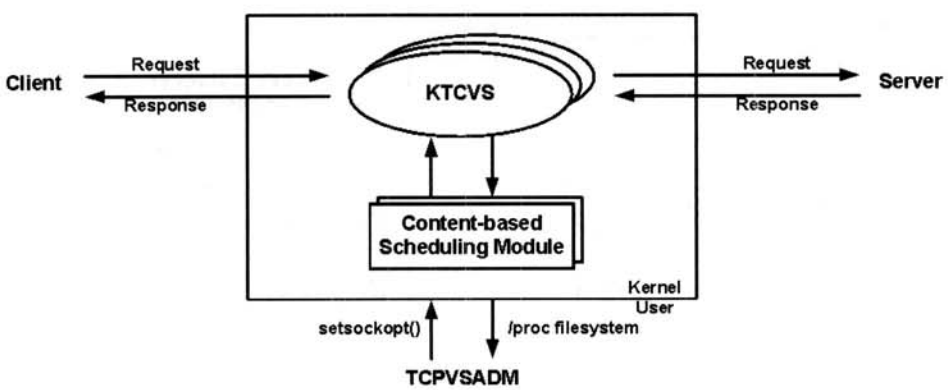
2. 기존 연구

(그림 2(a))의 부하 분산기에서 L4 부하분산 및 L7 부하분산을 구축함에 있어서 리눅스 기반 오픈 소스인 IPVS와 KTCPVS를 각각 사용할 수 있다.

2.1 IPVS (IP Virtual Server)

IPVS[1]는 IP 가상 서버(IP Virtual Server)를 나타낸다. IPVS는 리눅스 커널 내부에 Layer-4 레벨 부하 분산을 구현한다. (그림 5)는 IPVS의 전체적인 구조를 나타낸다. VS Schedule & Control Module은 IPVS의 메인 모듈이고, 이 모듈은 IP 부하 분산을 수행한다. IP 패킷들을 grab/rewrite 하기 위해 IP 패킷이 통과하는 2곳에서 훅(Hook)을 수행한다. 이 모듈은 새로운 커넥션에 대해 VS Rules 해쉬 테이블을 룩업(Look-up)하고, 기존 커넥션을 위해 Connection Hash Table을 체크한다. 유저 영역 프로그램인 IPVSADM은 가상 서버를 관리한다. 이 프로그램은 커널 안의 가상 서버 규칙들(Rules)을 수정하기 위해 setsockopt 함수를 사용하고, /proc 파일 시스템을 통해 가상 서버 규칙들을 읽는다.

IPVS는 사용자(Client)로부터 받은 요청을 처리하는 방식(Packet Forwarding Methods)에 따라 세 가지 방식[11]이 존재한다. 이러한 방식은 서버가 처리한 요청을 부하 분산기를 통해 사용자에게 전송하는 한 가지 방식(NAT: Network Address Translation)과 서버가 직접 사용자에게 보내는 두 가지 방식(Direct Routing과 IP Tunneling)으로 나눌 수 있다. 또한 IPVS는 사용자 요청을 실제 서버들로 스케줄링 하는 방식에 따라 8가지 방식(RR: Round-Robin, WRR: Weighted Round-Robin, LC: Least Connection, WLC: Weighted Least Connection, LBLC: Locality-Based Least Connection, LBLCR: Locality-Based Least Connection with Replication, SH: Source Hash, DH: Destination Hash)[12]이 존재한다. 스케줄링 방식은 실제 서버들로 요청을 분산할 때, 순서대로 분산하는 방식(RR,



(그림 6) KTCPVS의 전체 구조

WRR), 가상 서버와 실제 서버와의 연결 개수를 이용해서 분산하는 방식(LC, WLC), 실제 서버들을 몇 개의 단위의 묶어서 분산하는 방식(LBLC, LBLCR) 및 해쉬를 이용하는 방식(SH, DH)으로 나눌 수 있다.

2.2 KTCPVS (Kernel TCP Virtual Server)

KTCPVS[13]는 커널 TCP 가상 서버(Kernel TCP Virtual Server)를 나타낸다. KTCPVS는 리눅스 커널 내부에 어플리케이션 레벨 부하 분산(Layer-7 부하분산)을 구현한다. 유저 영역에서 Layer-7 부하분산은 부하(Overhead)가 너무 크기 때문에, Layer-7 부하분산은 커널 내부에서 수행되는 것이 좋다. 그리고 이것은 유저 영역과 커널 영역 사이의 컨텍스트 스위칭과 메모리 복사 부하를 피할 수 있다. KTCPVS가 IPVS에 비해 확장성을 떨어지지만, 효용성을 가진다. 왜냐하면, KTCPVS는 요청 내용을 요청이 서버로 전달되기 전에 알 수 있기 때문이다. (그림 6)은 KTCPVS의 전체적인 구조를 나타낸다.

KTCPVS의 동작 과정은 다음과 같다. 커널 쓰레드들은 크게 3가지 일을 수행한다. 첫째는, 요청의 내용을 분석(Parse)하고, 둘째는 요청을 스케줄링 규칙에 따라 서버들에게 전달(Forward)하고, 마지막으로 셋째는 클라이언트와 서버 사이에서 데이터를 중계(Relay)한다. 유저 영역 프로그램인 tcpvsadm은 KTCPVS를 관리한다. tcpvsadm은 setsockopt를 통해 커널 내부에 가상 서버 규칙들을 설정하고, getsockopt 또는 /proc 파일 시스템을 통해 커널로부터 KTCPVS 규칙들을 읽어들인다.

2.3 접근 방식

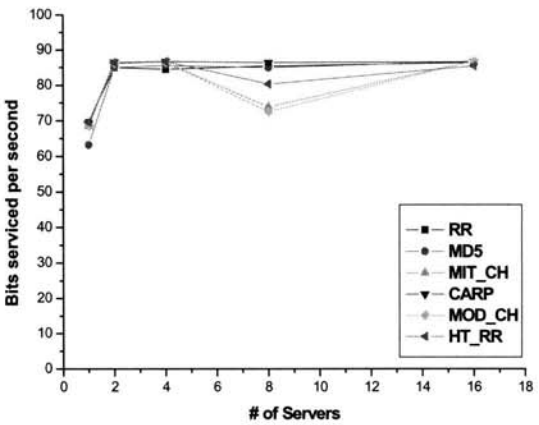
본 절에서는 IPVS 및 KTCPVS 구조가 가지는 문제점을 확장성과 Layer-7 부하분산 측면에서 분석하고, 분석된 문제점을 바탕으로 본 연구의 접근 방식을 소개한다.

2.3.1 IPVS 구조의 문제점

IPVS는 Layer-4 부하분산으로서 요청을 Layer-4에서 빠르게 처리할 수 있기 때문에 확장성을 가지지만, Layer-7 부하분산을 할 수 없다는 단점을 가진다. 즉, 사용자 주소(Source IP), 목적지 주소(Destination IP)를 통한 해싱은 가능하지만, 사용자의 요청 URL을 통한 해싱은 할 수 없다.

2.3.2 KTCPVS 구조의 문제점

KTCPVS는 Layer-7 부하분산으로서 요청을 Layer-7에서 처리할 수 있기 때문에 사용자의 요청 URL을 통한 해싱을 할 수 있다는 장점을 가진다. 그러나 모든 요청에 대해 Layer-7에서의 처리를 거쳐야 함으로 매 요청마다 이를 처리하는 부하(CPU 혹은 네트워크)가 상당히 크다. 이러한 부하는 KTCPVS의 확장성을 제한한다. (그림 7)은 각각의 해싱 알고리즘(4장 실험 참조)에서 서버의 개수에 따른 초당 처리하는 비트 수를 나타낸다. 그림에서 보면 서버의 개수가 늘어나도 Layer-7 부하분산을 처리하는 부하 분산기에



(그림 7) Layer-7 부하분산에서의 네트워크 병목

네트워크 병목이 발생하여 초당 처리하는 비트 수가 증가하지 않는 현상을 볼 수 있다.

2.3.3 본 연구의 접근 방식

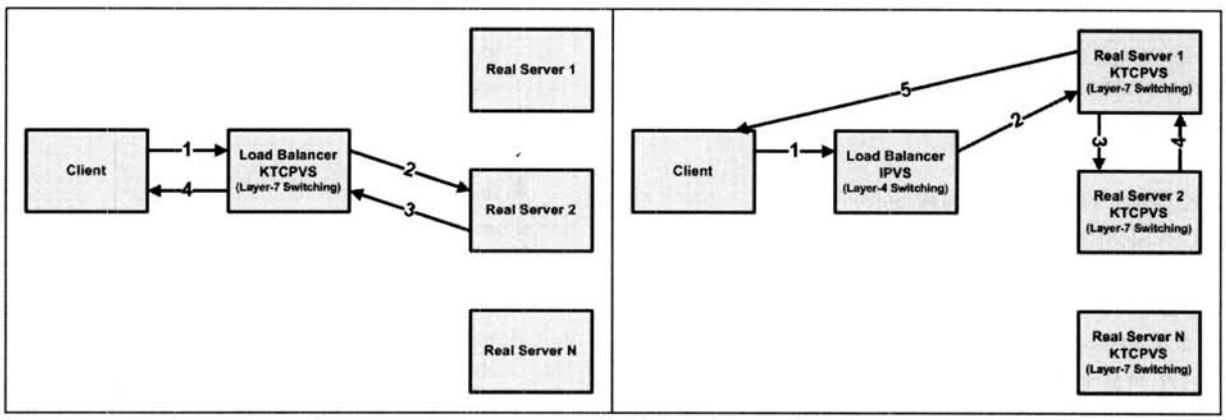
기존 연구에서는 (그림 2(a)) 부하 분산기를 위해 Layer 4 부하분산을 위해서는 IPVS를 사용하고, Layer 7 부하분산을 위해서는 KTCPVS를 사용한다. 본 논문에서는 확장성 있는 Layer 7 부하분산을 위해 IPVS와 KTCPVS를 같이 사용하는 새로운 구조를 제안한다.

3. 제안된 구조

3.1 전체 구조

제안된 구조는 IPVS의 확장성과 KTCPVS의 Layer-7 부하분산의 장점을 결합한 구조이다. IPVS는 부하 분산기에서 설치되어 확장성을 가지도록 했으며, KTCPVS는 Real Server에 설치되어 Layer-7 부하분산이 가능하도록 하였다. 즉, KTCPVS의 큰 부하(Layer-7 부하분산)를 Real Server들이 나누어가지도록 함으로써, Layer-7 부하분산에서 확장성을 보장하도록 만든 구조이다.

(그림 8)은 Layer-7 부하분산을 위한 기존 구조와 제안된 구조를 나타낸다. 기존 구조에서는 부하 분산기가 클라이언트로부터 요청을 받아 Layer-7 부하분산을 직접 수행한 후 Real Server를 찾는다. Real Server가 클라이언트의 요청을 처리하면 응답을 다시 부하 분산기를 통해서 수행함으로 부하 분산기에 부하(CPU 및 네트워크)가 상당히 큰 것을 알 수 있다. 반면, 제안된 구조에서는 부하 분산기는 IPVS(Layer-4 부하분산)로써 요청을 라운드 로빈 스케줄링 방식을 이용해서 빠르게 처리하며(Forward), 요청을 받은 Real Server에서 KTCPVS(Layer-7 부하분산)를 통해 요청을 처리할 Real Server를 찾는다. 이때, KTCPVS가 동작하는 Real Server와 요청을 처리할 Real Server는 동일하거나 동일하지 않을 수 있다. 해당 Real Server가 요청한 컨텐츠



(a) 기존 구조 (b) 제안된 구조

(그림 8) Layer-7 부하분산을 위한 기존 구조와 제안된 구조

를 갖고 있으면 바로 처리하며, 그렇지 않으면 요청한 컨텐츠를 갖고 있는 다른 Real Server로 요청하여 결과를 받아온 뒤 처리한다. Real Server가 클라이언트에 대한 요청을 처리하면 요청을 할당한 Real Server로 응답을 한다. 응답을 받은 Real Server는 부하 분산기를 거치지 않고 직접 클라이언트에게 응답함으로써 부하 분산기의 부하(CPU 및 네트워크)를 크게 줄일 수 있다.

3.2 동작 과정

제안된 구조의 동작 과정을 정리하면 다음과 같다.

단계 1 : 클라이언트가 부하 분산기(IPVS)에게 URL을 요청한다.

단계 2 : 부하 분산기(IPVS)는 라운드 로빈 스케줄링 방식을 이용해서 Real Server들의 KTCPVS 중의 하나로 URL을 요청한다.

단계 3 : 요청을 받은 Real Server(실제 서버 3번이라 가정) 내의 KTCPVS는 사용자의 요청 URL을 해싱하고, 내부 규칙에 따라 Real Server들 중 하나를 선택한다.

단계 4 : 요청을 받은 Real Server(실제 서버 7번이라 가정)는 다음과 같이 요청을 처리한다.

단계 4-1 : Real Server(실제 서버 7번)의 캐시 내부에 요청 데이터가 존재하면, 해당 데이터를 KTCPVS(실제 서버 3번)로 보낸다.

단계 4-2 : Real Server(실제 서버 7번)의 캐시 내부에 요청 데이터가 존재하지 않는다면, 웹 서버에 데이터를 요청한다. 웹 서버로부터 얻은 데이터를 압축 및 캐시에 저장하고, 해당 데이터를 KTCPVS(실제 서버 3번)로 보낸다.

단계 5 : 응답을 받은 KTCPVS(실제 서버 3번)는 부하 분산기(IPVS)를 거치지 않고 클라이언트에게 바로 데이터를 보낸다(Direct Routing).

3.3 비교 (정성적 비교)

<표 1>은 기존 구조와 제안된 구조를 Layer-4, Layer-7, 확장성 관점에서 비교한 표이다.

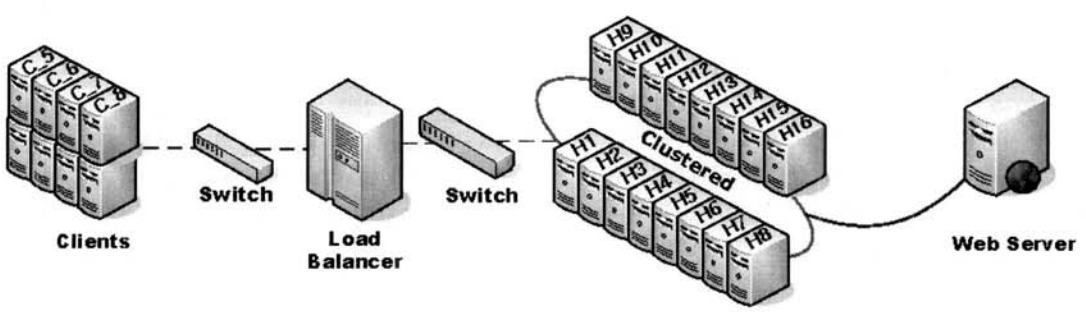
〈표 1〉 기존 구조 vs. 제안된 구조

	IPVS	TCPVS	제안된 구조
Layer-4 Switching	O	X	O
Layer-7 Switching	X	O	O
확장성	O	X	O

4. 실험

4.1 실험 방법

제안된 방법을 테스트하기 위해, 본 논문에서는 (그림 9)에서 보이는 실험 환경을 구축하였다. 실험 환경은 4 종류의 서버들(클라이언트, 부하 분산기 기반의 Front-End 서버,



(그림 9) Experimental Setup

〈표 2〉 Specifications of the machines

		하드웨어		소프트웨어	개수
		CPU (MHz)	RAM (MB)		
사용자 / 관리자		P-4 2260	256	Webstone[14, 15], Surge[16, 17]	10 / 1
부하 분산기		P-4 2400	512	DR	1
서버	캐시	P-2 400	256	Squid[18, 19]	16
	압축기			JPEG-6b	
웹 서버		P-4 2260	256	Apache	1

클러스터 서버들 및 웹 서버)로 구성된다. 서버들의 사양은 <표 2>와 같다.

클라이언트들은 인터넷으로부터 얻을 수 있는 웹 트래픽 생성 도구인 Webstone[14, 15]과 Surge[16, 17]를 이용해서 요청을 생성한다. 이러한 요청들은 텍스트 파일 및 이미지들로 구성된다. KTCPVS는 해싱 방법에 기반해서 요청들은 어디로 어떻게 보낼 것인지를 결정한다. 본 논문에서 기존의 해싱 방법(RR, MD5[2], MIT_CH[3], CARP[4], MOD_CH[5], 및 HT_RR[6])들이 구현되었고, 이들의 결과를 토대로 성능을 분석할 것이다.

요청된 페이지들이 16개의 서버들 중 하나에 저장(캐싱)되면, 선택된 서버는 요청을 부하 분산기를 통하지 않고 클라이언트에게 직접 보낸다. 부하 분산기는 들어오는 요청에 대한 처리만을 담당한다. 요청된 페이지가 캐시 서버에 저장되어 있지 않다면, 웹 서버에게 요청된다. 웹 서버로부터 새로운 컨텐츠를 받았다면, 이들은 선택된 캐시 서버에 저장(캐싱)된 후에 클라이언트에게 직접 전송될 것이다. Squid[18, 19]는 웹 서버로부터 받은 컨텐츠를 저장(캐싱)하는데 사용된다.

웹 서버로부터 받은 새로운 페이지가 저장될 때, 라운드 로빈 방식은 이들을 클러스터 내의 모든 서버에 저장된다. 반면, 다른 알고리즘은 선택된 하나의 캐시 서버에만 저장된다. 해당 요청에 대한 새로운 컨텐츠를 하나의 캐시 서버에만 저장하게 되면, 캐시 중복 문제를 제거함으로써 스토리지(저장 공간)의 확장성을 보장하게 된다. 라운드 로빈 방법은 비교 목적을 위해 사용된다. 왜냐하면, 모든 스케줄링 알고리즘 중에 가장 좋은 성능을 가지기 때문이다. 라운드 로빈 방법에서는 모든 서버가 동일한 파일들을 가지고 있음으로 특별한 요청 분배 방법이 필요 없고, 이는 특정 페이지에 대한 Hot Spot 문제가 발생하지 않음을 의미한다. 여기서 Hot Spot이란 다수의 사용자가 특정 페이지(Real Server)를 집중적으로 의미한다.

<표 3>은 실험에 사용된 중요 변수들을 나타낸다. 총 16대의 캐시 서버를 사용하였고, 부하 분산 정책으로는 RR(라운드 로빈), MD5(MD5를 이용한 해싱 방법[2]), MIT_CH(MIT에서 개발한 해싱 방법[3]), CARP(Microsoft에서 개발한 해싱 방법[4]), MOD_CH(MIT의 해싱 방법을 개선한 방법[5]), HT_RR(본 연구자들이 다른 논문에서 제안한 방법[6])을 사용하였다. 웹 트래픽은 가상 웹 트래픽(웹 트래픽 생성 도구인 Surge를 이용)과 실제 웹 트래픽(UNC-University of North Carolina와 Berkeley 대학의 실제 웹 트래픽)을 이용하였다. 각 서버당 최대 동시 연결 개수는 63개이고, 성능 측정 지표는 초당 처리된 요청 수와 초당 처리된 비트 수를 이용하였다.

〈표 3〉 Key variables in the experiments

Variables	Values
캐시 서버 수	16
부하 분산 정책	RR, MD5, MIT_CH, CARP, MOD_CH, HT_RR
웹 트래픽 생성 도구	Surge : Image + HTML (서로 다른 가중치를 가지는 100개의 파일) 실제 웹 트래픽 : UNC [20, 21], Berkeley [22, 23] (서로 다른 가중치를 가지는 100개의 파일)
각 서버당 동시 연결 개수	63
성능 측정 지표	초당 처리된 요청 수, 초당 처리된 비트 수

4.2 실험 결과 (정량적 비교)

4.2.1 Surge-100

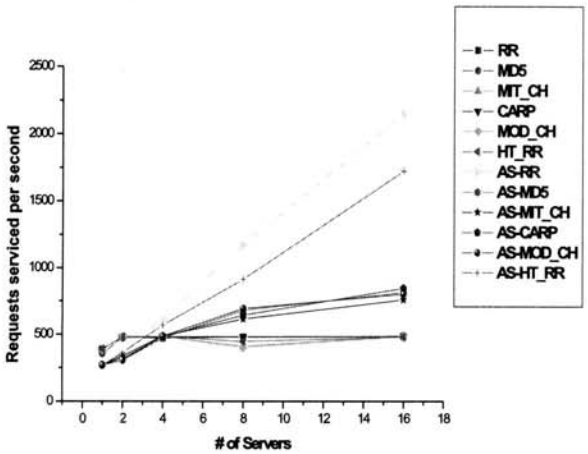
<표 4>, <표 5> 및 (그림 10)은 부하 발생기인 Surge를 통해 만든 웹 Trace(파일 크기마다 가중치를 가지는 100개의 파일)를 통한 실험 결과이다(초당 처리된 요청 수와 비

〈표 4〉 Surge-100에서의 초당 처리된 요청 수 (Req/sec)

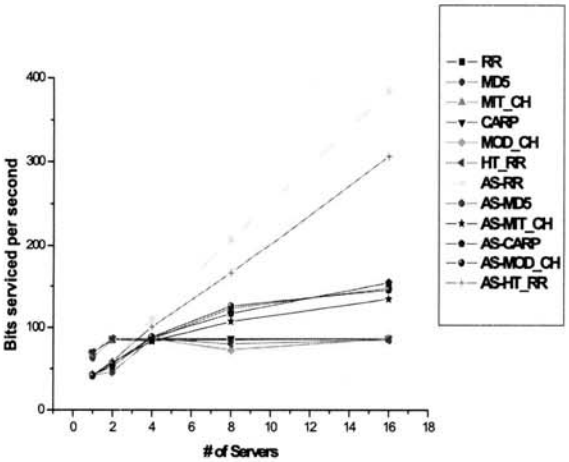
Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	389	354	381	388	383	389	270	268	269	270	268	266
2	476	474	484	480	485	478	345	306	338	321	306	362
4	471	477	482	481	482	482	618	470	486	481	484	568
8	478	477	411	483	401	447	1170	678	614	644	694	912
16	485	483	487	484	484	479	2147	815	760	845	802	1722

〈표 5〉 Surge-100에서의 초당 처리된 비트 수 (Mbit/sec)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	69	63	68	69	69	70	42	42	42	42	41	42
2	85	85	86	86	86	86	58	46	58	53	56	59
4	84	85	86	87	86	87	110	86	83	88	89	101
8	85	85	74	86	72	80	207	123	107	117	126	166
16	86	87	87	86	87	85	384	148	134	154	146	306



(a) 초당 처리된 요청 수



(b) 초당 처리된 비트 수

(그림 10) Surge-100

트 수). <표 4>는 (그림 10(a))를 나타내고, <표 5>는 (그림 10(b))를 나타낸다. 기존 구조(KTCPVS)를 통한 실험 결과(RR, MD5, MIT_CH, CARP, MOD_CH, HT_RR)는 서버의 수가 늘어남에 따라 성능이 증가하지 않음을 볼 수 있다. 이는 모든 요청이 부하 분산기를 통과해야 함으로써, 부하 분산기에 병목이 발생하였음을 의미한다. 병목 가능성은 2가지로 분류할 수 있는데, 하나는 CPU 병목이고 나머지 하나는 Ethernet 병목이다. 본 실험에서는 (그림 10(b))를 통해 Ethernet 병목이 발생하였음을 알 수 있다. 이에 반해, 제안된 구조에서의 실험 결과(AS-RR, AS-MD5, AS-MIT_CH, AS-CARP, AS-MOD_CH, AS-HT_RR)는 서버가 늘어남에 따라 성능도 증가함으로써 확장성을 가짐을 알 수 있다.

4.2.2 UNC-2001

<표 6>, <표 7> 및 (그림 11)은 UNC-2001을 통해 만든 웹 Trace(파일 크기마다 가중치를 가지는 100개의 파일)를 통한 실험 결과이다(초당 처리된 요청 수와 비트 수). <표 6>은 (그림 11(a))를 나타내고, <표 7>은 (그림 11(b))를 나타낸다. 기존 구조(KTCPVS)를 통한 실험 결과(RR, MD5, MIT_CH, CARP, MOD_CH, HT_RR)는 서버의 수가 늘어남에 따라 성능이 증가하지 않음을 볼 수 있다. 이는 모든 요청이 부하 분산기를 통과해야 함으로써, 부하 분산기에 병목이 발생하였음을 의미한다. 병목 가능성은 2가지로 분류할 수 있는데, 하나는 CPU 병목이고 나머지 하나는 Ethernet 병목이다. 본 실험에서는 (그림 11(b))를 통해 Ethernet 병목이 발생하였음을 알 수 있다. 이에 반해, 제안된 구조에서의 실험 결과(AS-RR, AS-MD5, AS-MIT_CH, AS-CARP, AS-MOD_CH, AS-HT_RR)는 서버가 늘어남에 따라 성능도 증가함으로써 확장성을 가짐을 알 수 있다.

4.2.3 Berkeley-1998

<표 8> <표 9> 및 (그림 12)는 Berkeley-1998을 통해

〈표 6〉 Surge-100에서의 초당 처리된 요청 수 (Req/sec)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	517	526	525	518	526	526	247	246	246	245	247	245
2	961	923	793	679	797	818	433	421	375	394	400	359
4	929	1001	993	732	888	972	787	654	515	652	557	615
8	839	955	1008	934	939	942	1537	934	729	873	828	986
16	966	1021	1001	995	974	990	3059	1313	1203	1178	1082	1828

〈표 7〉 Surge-100에서의 초당 처리된 비트 수 (Mbit/sec)

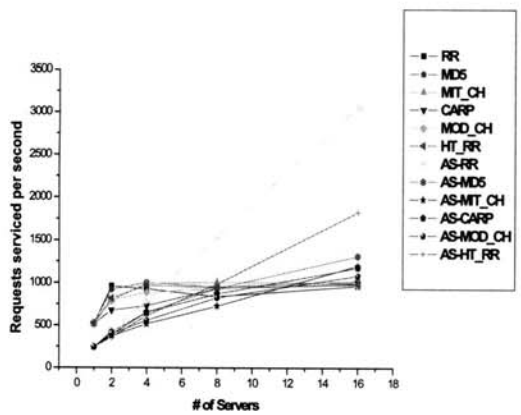
Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	40	40	41	40	40	41	18	18	18	18	18	18
2	77	74	63	54	65	65	34	33	28	28	30	30
4	76	81	81	58	71	80	62	52	38	49	43	49
8	69	79	82	77	77	78	123	72	55	68	63	79
16	79	82	82	81	81	82	242	100	95	92	83	145

〈표 8〉 Surge-100에서의 초당 처리된 요청 수 (Req/sec)

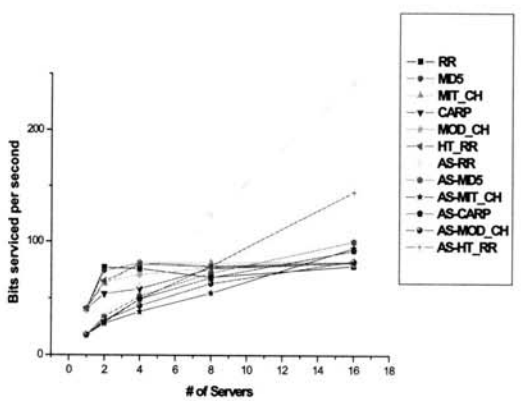
Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	484	526	517	525	525	526	238	245	245	245	245	246
2	688	664	611	759	688	759	410	373	361	416	387	383
4	714	754	630	739	710	761	772	655	632	729	567	716
8	552	741	680	742	723	764	1512	1063	778	1091	884	1164
16	772	767	730	751	771	762	2899	1490	1143	1441	1400	2295

〈표 9〉 Surge-100에서의 초당 처리된 비트 수 (Mbit/sec)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR	AS-RR	AS-MD5	AS-MIT_CH	AS-CARP	AS-MOD_CH	AS-HT_RR
1	45	51	49	50	51	50	19	19	19	19	19	20
2	78	72	62	84	71	84	39	28	28	46	30	30
4	81	84	64	84	71	85	74	73	42	54	30	67
8	55	85	75	84	80	84	143	103	41	71	50	115
16	84	85	83	85	84	85	282	115	58	91	113	240

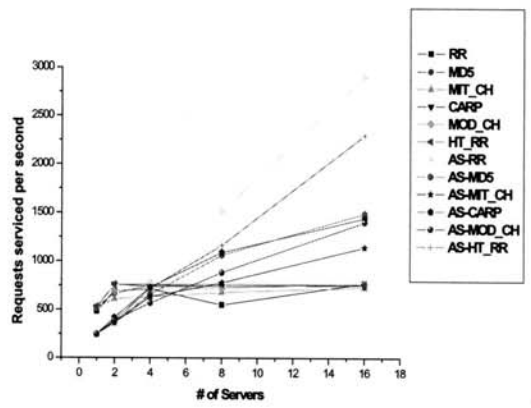


(a) 초당 처리된 요청 수

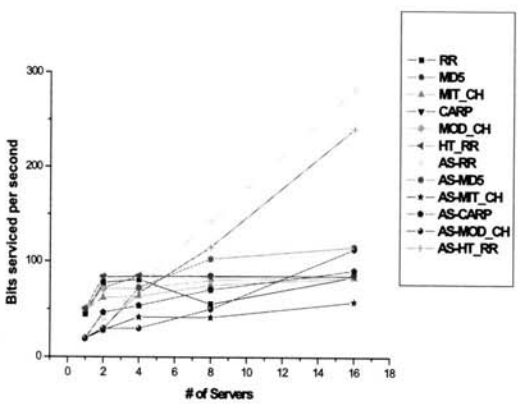


(b) 초당 처리된 비트 수

(그림 11) UNC-2001



(a) 초당 처리된 요청 수



(b) 초당 처리된 비트 수

(그림 12) Berkeley-1998

만든 웹 Trace(파일 크기마다 가중치를 가지는 100개의 파일)를 통한 실험 결과이다(초당 처리된 요청 수와 비트 수). <표 8>은 (그림 12(a))를 나타내고, <표 9>는 (그림 12(b))를 나타낸다. 기존 구조(KTCPVS)를 통한 실험 결과(RR, MD5, MIT_CH, CARP, MOD_CH, HT_RR)는 서버의 수가 늘어남에 따라 성능이 증가하지 않음을 볼 수 있다. 이는 모든 요청이 부하 분산기를 통과해야 함으로써, 부하 분산기에 병목이 발생하였음을 의미한다. 병목 가능성은 2가지로 분류할 수 있는데, 하나는 CPU 병목이고 나머지 하나는 Ethernet 병목이다. 본 실험에서는 (그림 12(b))를 통해 Ethernet 병목이 발생하였음을 알 수 있다. 이에 반해, 제안된 구조에서의 실험 결과(AS-RR, AS-MD5, AS-MIT_CH, AS-CARP, AS-MOD_CH, AS-HT_RR)는 서버가 늘어남에 따라 성능도 증가함으로써 확장성을 가짐을 알 수 있다.

4.2.4 기존 구조 vs. 제안된 구조

<표 10>, <표 11> 및 <표 12>는 기존 구조에 대한 제안된 구조의 성능 향상률(%)을 나타낸 것이고, 성능 향상률은 (제안된 구조 - 기존 구조) / (기존 구조)를 이용하여 구하였다. 표에서 보면 서버의 수가 작을 때에는 기존 구조에 비해 성능이 떨어지는 것을 볼 수 있다. 서버의 수가 1-2대일 때 기존 구조가 항상 우수하고, 서버의 수가 16대일 때는 제안 구조가 항상 우수하다. 또한 서버의 수가 4-8대일 경우 트래픽이나 분산 알고리즘의 방식에 따라 다르다. 즉, 4대일 경우는 기존 구조가 성능이 더 우수한 경우가 많고, 8대일 경우는 제안 구조가 성능이 더 우수한 경우가 많다.

이는 부하 분산기에서 병목이 발생하지 않는다면 기존 구조가 더 나은 성능을 가짐을 의미한다. 왜냐하면, 기존 구조에서는 부하 분산기가 KTCPVS 역할을 수행하고, Real Server에 부하 분산과 관련하여 어떠한 부하(Overhead)도 주지 않기 때문이다. 그러나 서버의 개수가 증가하면 기존 구조에서는 병목이 발생하고, 이때 제안된 구조는 기존 구조에 비해 상대적으로 높은 성능 향상률을 가짐을 알 수 있다. 이러한 성능 향상률을 가지고 결과를 정리하면 서버의 개수가 작을 때에는 기존 구조가 적당하지만, 서버의 개수가 계속적으로 증가하고 확장성이 필요할 때면 제안된 구조가 적당하다는 것을 알 수 있다.

5. 결 론

제안된 구조의 장점은 Layer-7 부하분산에서 서버의 개수에 비례하여 확장성을 가진다는 것이다. 기존 구조에서는 부하 분산기 1대에서 Layer-7 부하분산을 담당하였지만, 제안된 구조에서는 Layer-7 부하분산을 Real Server들에서 분산 처리하는 구조를 갖고 있다. 이는 Layer-4 부하분산을 이용하여 요청 패킷을 Real Server로 전달하고(Forward), 실제 서버는 Layer-7 부하분산을 수행함으로써 부하 분산기 1대에서 처리하던 Layer-7 부하분산의 부하(Overhead)를 실제 서버들에게 분배하였음을 의미한다.

제안된 구조의 단점은 서버의 개수가 적을 때 기존 구조에 비해 성능이 떨어진다는 것이다. 이는 제안된 구조가 매 요청을 처리할 때 마다 Layer-4 및 Layer-7 부하분산을 매번 수행해야 함을 의미한다. 그렇기 때문에 서버의 개수가 적을 때에는 부하 분산기에서 Layer-7 부하분산만을 수행하

〈표 10〉 성능 향상률 (Surge-100) (%)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR
1	-30	-24	-30	-30	-30	-32
2	-28	-36	-30	-33	-37	-24
4	31	-2	1	0	1	18
8	145	42	50	33	73	104
16	343	69	56	74	66	259

〈표 11〉 성능 향상률 (UNC-2001) (%)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR
1	-52	-53	-53	-53	-53	-53
2	-55	-54	-53	-42	-50	-56
4	-15	-35	-48	-11	-37	-37
8	83	-2	-28	-6	-12	5
16	217	29	20	18	11	85

〈표 12〉 성능 향상률 (Berkeley-1998) (%)

Server #	RR	MD5	MIT_CH	CARP	MOD_CH	HT_RR
1	-51	-53	-53	-53	-53	-53
2	-40	-44	-41	-45	-44	-50
4	8	-13	0	-1	-20	-6
8	174	43	14	47	22	52
16	276	94	56	92	82	201

는 기존 구조가 유리하다.

본 논문에서는 기존 부하 분산 구조(IPVS, KTCPVS)의 문제점을 분석하고, 이를 해결하는 새로운 구조를 제안하였다. 제안된 구조는 기존 구조의 단점을 제거하고, 장점만을 결합한 구조이다. 실험을 통해 제안된 구조가 기존 구조들에 비해 확장성 및 성능 향상에 기여하였음을 확인하였다.

참 고 문 헌

[1] LVS(Linux Vitual Server), http://www.linuxvirtualserver.org.

[2] D. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, 1992.

[3] David Karger and al. "Web Caching with consistent hashing," In WWW8 Conference, 1999.

[4] Micorsoft Corp., "Cache Array routing protocol and microsoft proxy server 2.0," White Paper, 1999.

[5] F. Baboescu, "Proxy Caching with Hash Functions," Technical Report CS2001-0674, 2001.

[6] 곽후근, 정규식, "무선 인터넷 프록시 서버 클러스터 시스템에서 라운드 로빈을 이용한 해싱 기법", 정보처리학회논문지A, 제13-A권, 제7호, pp.615-622, 2006.12.

[7] A. Feldmann, R. Caceres, F. Douglis, G. Glass and M. Rabinovich, "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments," In Proceedings of the INFOCOM Conference, 1999.

[8] A. Savant, N. Memon and T. Suel, "On the Scalability of an Image Transcoding Proxy Server," In IEEE International Conference on Image Processing, Barcelona, Spain, 2003.

[9] A. Fox, "A Framework for Separating Server Scalability and Availability from Internet Application Functionality," Ph. D. Dissertation, U. C. Berkeley, 1998.

[10] 곽후근, 정규식, "통합형 무선 인터넷 프록시 서버 클러스터 구조", 정보처리학회논문지A, 제13-A권 제3호, 2006.

[11] How Virtual Server Works?, http://www.linuxvirtualserver.org/how.html.

[12] LVS Scheduling Algorithms, http://www.linuxvirtualserver.org/docs/scheduling.html.

[13] KTCPVS, http://www.linuxvirtualserver.org/software/ktcpvs/ktcpvs.html.

[14] Mindcraft, Inc., "WebStone : The Benchmark for Web Server," http://www.mindcraft.com/web-stone.

[15] J. Nakano, P. Montesinos, K. Gharachorloo, and J. Torrellas, "ReViveI/O: efficient handling of I/O in highly-available rollback-recovery servers," The 12th Internation Symposium on High-Performance Computer Architecture, pp.200-211, 2006.

[16] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proc. ACM SIGMETRICS Conf., Madison, WI, Jul., 1998.

[17] R. Zhang, T. Abdelzaher, and J. Stankovic, "Efficient TCP connection failover in Web serer clusters," 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp.1219-1228, March, 2004.

[18] Squid Web Proxy Cache, http://www.squid-cache.org.

[19] W. Liao and P. Shih, Architecture of proxy partial caching using HTTP for supporting interactive video and cache consistency, 11th International Conference Computer Communications and Networks, 2002, pp.216-221.

[20] H. Felix, K. Jeffay, and F. Smith, "Tracking the Evolution of Web Traffic," Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp.16-25, 2003.

[21] D. Lu, Y. Qiao, P. Dinda and F. Bustamante, "Modeling and Taming Parallel TCP on the Wide Area Network," Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, April, 2005.

[22] B. A. Mah, "An Empirical Model of HTTP Network Traffic," Proceedings of INFOCOM, pp.592-600, 1997.

[23] J. Xu and W. Lee, "Sustaining availability of Web services under distributed denial of service attacks," IEEE Transactions on Computers, Vol.52, No.2, pp.195-208, Feb., 2003.

권 희 웅

e-mail : didorito@q.ssu.ac.kr

1997년 숭실대학교 정보통신전자공학부 (학사)

1999년 숭실대학교 전자공학과(석사)

2000년~현 재 (주)펌킨넷 코리아 중앙연구소 실장

1999년~현 재 숭실대학교 전자공학과 대학원 박사과정

관심분야 : 네트워크 및 어플리케이션에 대한 부하 분산, 가속, 보안

곽 후 근

e-mail : gobarian@q.ssu.ac.kr
1996년 호서대학교 전자공학과(학사)
1998년 숭실대학교 전자공학과 대학원 (석사)
1998년~2006년 숭실대학교 전자공학과 대학원(박사)
1998년 8월~2000년 7월 (주)3R 부설 연구소 주임연구원
2006년 3월~현 재 숭실대학교 전자공학과 대학원(postdoc)
관심분야 : 네트워크 컴퓨팅 및 보안

정 규 식

e-mail : kchung@q.ssu.ac.kr
1979년 서울대학교 전자공학과(공학사)
1981년 한국과학기술원 전산학과(이학석사)
1986년 미국 University of Southern California(컴퓨터공학석사)
1990년 미국 University of Southern California(컴퓨터공학박사)
1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원
1990년 9월~현 재 숭실대학교 정보통신전자공학부 교수
관심분야 : 네트워크 컴퓨팅 및 보안