# 클러스터링 웹 서버 환경에서 차별화 서비스를 위한 3단계 동적 부하분산기법

정회원 이 명 섭*, 박 창 현**

# The three-level load balancing method for Differentiated service in clustering web server

Myung Sub Lee*, Chang Hyeon Park** ***Regular Members***

요 약

최근 들어, 인터넷 사용자의 폭발적인 증가로 인하여 차별화된 웹 서비스를 제공해주는 웹 응용프로그램들의 개발이 활발해지고 있다. 이에 따라 웹 서버내의 품질향상을 보장해주는 웹QoS 기술은 전자상거래나 웹 호스팅 같은 부분에서 점점 더 중요한 문제로 대두되고 있다. 그러나 대부분의 웹 서버들은 FIFO 방식의 최선 서비스만을 제공하고 있으며, 정보의 중요도나 정보를 제공받는 사용자의 중요도에 따라 차별화된 품질보장을 제공하지 못한다. 본 논문에서는 클러스터링 웹 서버 환경에서 차별화 서비스를 위한 3단계 동적 부하분산 기법을 제안한다. 먼저, 커널 수준 접근 방식에서는 커널 상에 실시간 스케줄링 프로세스를 두어 웹서버에서 수행중인 스케줄링 프로세스와 연동시키고, 커널 내부에서도 웹 서버에서 할당된 사용자 요청 우선순위를 유지하도록 한다. 둘째, 웹 서비스의 신뢰성과 반응속도를 개선하기 위하여 IP수준의 가장법과 터널링 기술을 이용하여 웹 서버의 부하를 분산을 수행한다. 셋째, 동적 부하분산을 제공하기 위해 SNMP중에 시스템 부하관련 MIB-II 정보를 검출하여 부하 분산에 반영한다.

**Key Words :** Dynamic load balancing, Web QoS, real-time scheduler, SNMP, MIB-II

ABSTRACT

Recently, according to the rapid increase of Web users, various kinds of Web applications have been being developed. Hence, Web QoS(Quality of Service) becomes a critical issue in the Web services, such as e-commerce, Web hosting, etc. Nevertheless, most Web servers currently process various requests from Web users on a FIFO basis, which can not provide differentiated QoS. This paper presents a load balancing method to provide differentiated Web QoS in clustering web server. The first is the kernel-level approach, which is adding a real-time scheduling process to the operating system kernel to maintain the priority of user requests determined by the scheduling process of Web server. The second is the load-balancing approach, which uses IP-level masquerading and tunneling technology to improve reliability and response speed upon user requests. The third is the dynamic load-balancing approach, which uses the parameters related to the MIB-II of SNMP and the parameters related to load of the system such as memory and CPU.

* 영남대학교 전자정보공학부 객원교수(skydream@yu.ac.kr), ** 영남대학교 컴퓨터공학과 부교수 (park@yu.ac.kr), 교신저자
논문번호 : KCS2004-11-276, 접수일자 : 2004년 11월 12일
※ 이 논문은 2004학년도 영남대학교 학술연구조성비 지원에 의한 것임

## Ⅰ. 서 론

최근 인터넷의 사용이 대중화되고, 사용자 또한 급속도로 증가함에 따라 웹 페이지에 대한 접속은 폭발적으로 증가하고 있다. 이런 수요 증가에 비해 제공되는 서비스들의 품질은 그만큼 따라가지 못하는 것이 현실이다. 그 주요 원인은 웹 서버의 서비스 성능의 부족에서 오는 것이며 클러스터링 웹 서버가 중요한 해결 수단으로 간주되고 있다. 클러스터링 웹 서버란 수많은 사용자들의 요청을 처리하기 위해 서버들을 네트워크로 연결한 것으로 사용자들의 요청을 서버들이 나누어 처리하는 구조를 갖는 웹 서버를 말한다. 기존의 클러스터 시스템의 경우 단순히 스케줄링 방식에 의해서 클라이언트 요구를 실제 서버(real server)에게 전달하는 기능만을 가지고 있으며, 부하 분산 방법에서 실제 서버상의 부하(load) 정보 보다 스케줄링 방식에 대한 연구가 대부분이다.[1]

본 논문에서는 클러스터링 웹서버 환경에서 차별화 서비스를 위한 3단계 동적 부하분산 기법을 제시한다. 첫째, 웹 서버 상에는 클라이언트 요청정보의 중요도에 따라 우선순위를 부여할 수 있는 스케줄링 모듈을 추가하고, OS 커널 상에는 이와 연동되는 실시간 스케줄러를 두어 웹 서버에서 부여된 우선순위 정보가 커널 내부의 프로세스에도 유지될 수 있도록 함으로써 더 효율적인 차별화 서비스를 제공할 수 있도록 한다. 둘째, IP 가장법(masquerading)과 터널링(tunneling)[2]기술을 이용한 분산 웹 서버를 구성하여 웹 서버의 부하를 클래스별로 분산함으로써 웹 서비스의 신뢰성과 응답속도를 개선한다. 셋째, 동적 부하분산을 제공하기 위해 SNMP (Simple Network Management Protocol)v2[3][4] 중에 시스템 부하관련 MIB-Ⅱ(Management Information Base-II)[5] 정보를 검출하여 부하 분산에 반영한다.

본 논문의 2장에서는 차별화된 웹 서비스에 관련된 기존 연구를 기술하고, 3장에서 제안기법 및 차별화된 웹 서비스 시스템의 구조 및 동작방식을 기술한다. 4장에서 제안시스템의 성능평가를 보이고, 5장에서 결론을 기술한다.

## Ⅱ. 관련 연구

이 절에서는 클러스터링 웹 서버에서 차별화된 서비스를 제공하기 위한 기존의 연구방법들을 소개하고 그들의 문제점을 비교분석한다.

### 2.1 클러스터링 웹서버

클러스터링 웹 서버에 관한 기존 연구는 크게 4가지로 클라이언트 측과 서버 측의 round-robin DNS(DNS: Domain Name Service)[7], 응용프로그램 수준(application level)에서의 스케줄링, IP 수준(IP level)에서의 스케줄링으로 나눌 수 있다.

첫 번째 방법은 클라이언트 측 접근 방법으로 클라이언트 측에서 제공하는 애플릿(applet)이 분산 서버의 부하 정보를 얻기 위해 요청 메시지를 전송하고 서버에서 전송한 응답 메시지에서 수집된 서버들의 부하 정보를 기반으로 서버를 선택하여 클라이언트의 요청 메시지를 전달한다 버클리 대학의 Smart Client[8]가 이에 해당한다. 이 방법은 클라이언트 입장에서 보면 서버가 하나로 인식(transparent)되지 않고, 모든 클라이언트 응용프로그램 부분을 수정해 주어야 하는 단점이 있다.

두 번째 방법은 서버측에서의 round-robin DNS 방법이 있다. 이 방법은 서버 측에서만 변경되는 단순한 방법으로 DNS에 round-robin방식을 적용하여 순차적으로 IP주소를 대응시켜 서버를 다르게 선택하고 부하를 분산 시키는 방법이다 NCSA의 scalable web server[9]가 이에 해당한다. 이 방법은 클라이언트 캐싱(caching)과 계층적인 DNS 시스템 구성으로 인하여 각 서버가 과부하 상태에 도달할 때 서버를 제어하기 힘들다는 단점이 있다.

세 번째로 서버측에서의 응용프로그램 수준 스케줄링을 이용한 것으로 EDDIE[10], Reverse-proxy [11], pWEB[12], sWEB[13] 등이 여기에 속한다. 이 방법은 분산되어 있는 서버들이 자신의 부하를 측정하여 클라이언트로부터 HTTP요청이 들어 올 경우, 자신의 부하 상태에 따라 처리 여부를 판단하여 만약 자신이 처리할 수 없으면 분산 서버 내의 다른 서버로 HTTP 요청을 전달(forward)하고, 그 결과를 되받아 이를 다시 클라이언트에게 최종적으로 전송한다. 이 방법의 단점은 두 번 이상의 TCP 연결로 인해 전송지연이 심해지고 응용프로그램 수준에서 HTTP요청과 응답을 처리하는데 많은 오버헤드가 발생한다는 것이다.

마지막으로 서버측에서의 IP 수준 스케줄링 방법이 있으며, 버클리 대학의 Magic router[14]와 Cisco의 Local Director[15]가 이 방식을 사용한다 이 방법은 네트워크 주소 변환 기법을 사용하여 서로 다른 서버상의 병렬 서비스를 단일 IP 주소에 의한

가상의 서비스처럼 보이게 한다. 구성요소로는 부하를 분산시키기 위한 스케줄러 역할을 하는 부하 제어기(load balancer)와 실제 웹 서비스를 제공하는 실제 서버들로 구성 된다.

클라이언트가 분산 서버에서 제공하는 서비스에 접속하고자 할 경우, 클라이언트의 요구 패킷은 부하 제어기로 간다. 부하 제어기에서는 클라이언트에서 요청한 패킷의 목적지 주소와 포트번호를 검사하게 된다. 그 내용이 가상 서버의 서비스와 일치하게 되면 스케줄링 알고리즘에 따라 부하 제어기에서 실제 서버를 선택하고, 해시 테이블에 새로운 접속을 추가한다. 서버가 선택되면 부하 제어기에서는 패킷변환(rewriting) 작업을 수행한 후 패킷을 서버로 전송(forward)한다. 응답 패킷에 대한 처리가 완료되면 서버에서는 부하제어기로 응답 패킷을 보내고, 부하제어기에서는 응답 패킷을 클라이언트에게 전송하기 위해 다시 패킷 변환 작업을 수행한다. 네트워크 주소 변환 기법은 실행 서버의 수가 20대가 넘어가게 되면 부하제어기에 병목현상이 발생하며, 패킷 변환에 대한 오버헤드가 크다는 단점이 있다.

### 2.2 리눅스 가상 서버의 서비스 처리방식

리눅스 가상 서버에서는 웹 서버 부하분산을 위하여 네트워크 주소 변환 기법, IP 터널링, 디렉트 라우팅(direct routing)을 제공하고 있다. 먼저, 네트워크 주소 변환 기법을 이용한 부하 분산은 Cisco의 Local Director에 기반 하여 다음과 같이 작동하며, [그림 1]에서 구조를 보인다.

① 클라이언트로부터 패킷이 부하 제어기에게 도달하게 되면, 부하 제어기는 스케줄링 방법에 따라 실제 서버들 중 한 서버를 선택한다.
② 선택한 실제 서버의 주소를 패킷의 목적지(destination) 필드를 실제 서버의 주소로 변경하여 실제 서버에게 패킷을 보내준다.
③ 실제 서버로부터 응답 패킷이 부하 제어기에게 오게 되면, 부하 제어기는 응답 패킷의 출발지(source) 필드를 가상 서버의 주소로 변경하여 클라이언트에게 보내주게 된다.

네트워크 주소 변환을 이용한 부하 분산 기법에서, 모든 패킷은 부하 제어기를 지나가고 항상 변경되어져야 하므로 많은 처리 오버헤드를 필요로 한다. 따라서 실제 서버들의 수가 20대 이상으로 많아지면 부하 제어기에서 병목 현상이 일어난다. 따라서 확장성에 문제가 발생하는 단점을 가진다.

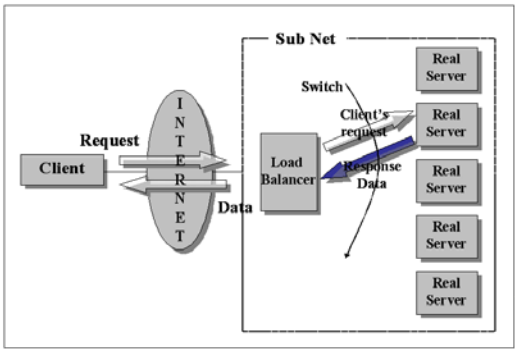


그림 1. NAT(Network Address Translation) 구조

둘째로 IP 터널링은 IP 데이터그램 안에 IP 데이터그램을 넣을 수 있는 기술로써 리눅스 가상 서버에서 다음과 같이 동작하며, [그림 2]에서 구조를 보인다.

① 부하 제어기에게 패킷이 도달 하게 되면, 부하 제어기는 이 패킷을 IPinIP 패킷에 포장하여 실제 서버에게 보낸다.
② 실제 서버는 부하 제어기로부터 받은 IPinIP 패킷을 풀어 처리한 후 부하 제어기를 거치지 않고 클라이언트에게 바로 응답을 보낸다.

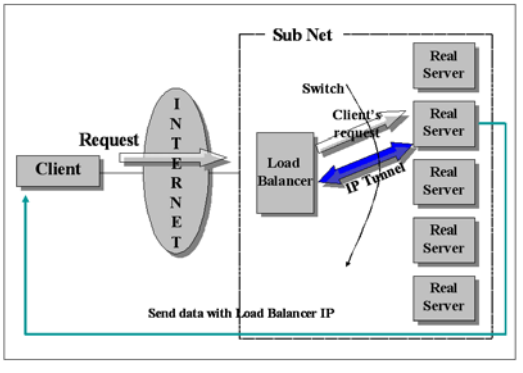


그림 2. IP 터널링(tunneling) 구조

IP 터널링 기법을 이용한 부하 분산은 실제 서버가 IP 터널링 기능을 지원해야 하는 단점이 있다. 그러나 실제 서버가 사용자에게 직접 응답을 보내므로 네트워크 주소 변환 기법 보다 확장성이 좋으며 부하 제어기에서의 병목 현상을 줄일 수 있다. 셋째로 디렉터 라우팅을 이용한 가상 서버는 IBM의 NetDispatcher[17]에 기반 하여 다음과 같이 작동하며, [그림 3]에서 구조를 보인다.

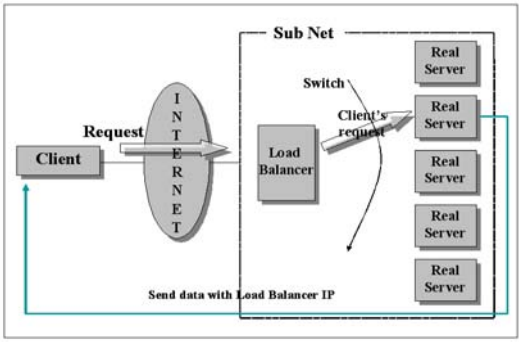


그림 3. IP direct routing 구조

① 부하 제어기에게 패킷이 도달하게 되면, 부하 제어기는 패킷의 데이터그램에서 MAC 주소만을 변경하여 실제 서버에게 패킷을 전달한다.

② 부하 제어기와 같은 가상 IP를 공유하고 있는 실제 서버는 이 패킷을 전달받아 처리한 후 부하 제어기를 거치지 않고 클라이언트에게 바로 응답을 보낸다.

디렉트 라우팅을 이용한 부하 분산 기반은 데이터그램의 MAC 주소만을 변경하여 재전송하게 되므로, 부하 제어기와 실제 서버들은 단일한 물리적 세그먼트(segment) 안에 있어야 하는 제약을 가진다. 또한 실제 서버들은 ARP에 반응하지 않는 가상 IP를 부하 제어기와 공유해야한다. 하지만, IP 터널링과 마찬가지로 실제 서버가 사용자에게 직접 응답을 보내므로 네트워크 주소 변환 기법 보다 좋은 확장성을 갖는다. <표 1>은 이 상에서 언급한 리눅스 가상 서버의 서비스 처리 방식과 본 논문에서 제안한 방식을 비교 분석한 결과이다.

<표 1>에서의 LVS/NAT&TUN은 본 논문에서 제안한 3단계 동적 부하분산 기법 중 2번째 단계로 네트워크 주소 변환 기법과 IP 터널링을 결합한 형태이다. 이 방식은 네트워크 주소 변환 기법에서 발생하는 패킷 변환 오버헤드와 부하 제어기의 병목 현상을 제거하며, IP 터널링에서 병렬 서비스를 단일 IP 주소에 의한 가상 서비스처럼 구성하지 못한다는 단점을 제거한다.

## Ⅲ. 제안시스템 및 기법

본 논문에서 제안하는 기법은 클라이언트의 요청에 우선순위를 제공하기 위하여 웹 서버에서 컨텐츠를 분류하고 스케줄링을 수행한다. 또한, 커널 수준의 실시간 스케줄러에게 프로세스를 맵핑하는 커널 수준 접근법을 지원하며, 웹 서비스의 신뢰성보장과 응답속도를 개선하기 위하여 IP 수준의 가장 법과 터널링 기술을 이용하여 웹 서버의 부하를 분산하는 시스템이다. 시스템의 전반적인 구성은 [그림 4]에 보이고 있으며, 자세한 구조 및 동작에 대한 설명은 아래의 각 절에서 주어진다.

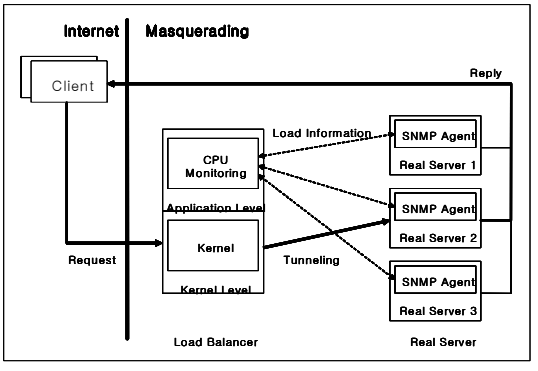


그림 4. 시스템 전체 구성도

### 3.1 부하제어기(Load balancer)

부하 제어기의 구조는 커널 수준(kernel level)과 응용프로그램 수준(application level)으로 구성된다. 커널 수준 접근 방식은 하나의 요청에 대한 효율적인 우선순위를 보장하기 위한 것으로, 아파치 웹 서버의 스케줄러가 커널내부의 실시간 스케줄러에게 프로세스를 맵핑하는 방법으로 시스템을 구현한다. [그림 5]에서 커널 수준 접근법의 구조를 보인다. [그림 5]에서, 클라이언트의 요청이 NIC(Network

표 1. 클러스터링 서비스 비교분석

| | LVS/NAT | LVS/TUN | LVS/DR | LVS/NAT/TUN |
|---|---|---|---|---|
| Server | any | tunneling | non-arp dev | tunneling |
| Server network | private | LAN/WAN | LAN | LAN/WAN |
| Server counter | low(10~20) | high(100) | high(100) | high(100) |
| Server gateway | load balancer | own router | own router | own router |
| 단 점 | 2번의 패킷변환 오버헤드 | 병렬서비스복수 IP필요 | 단일 물리 세그먼트만 가능 | 1번의 패킷 변환 오버헤드 |

Interface Card)를 통해서 들어오면 웹 서버는 TCP listen 버퍼에서 80번 포트로 요청을 받아들이고, 연결 관리자에서 분류 정책(클라이언트 IP, URL, 파일이름, 디렉토리, 사용자 인증 등)에 따라 각 연결별로 분류를 한다.

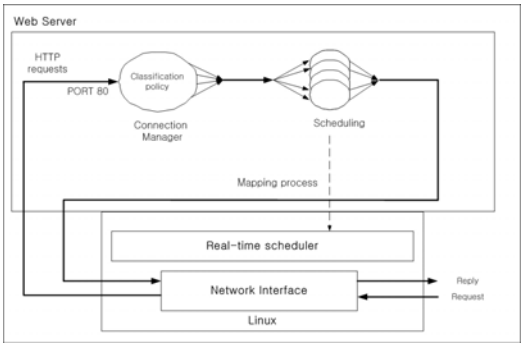


그림 5. 커널수준 접근법

분류된 요청은 우선순위를 부여하여 각각의 요청 큐에 입력되고 응용 프로그램 수준의 SNMP 모니터링 정보와 연계하여 스케줄링 된다. 이때, 웹 서버에서 스케줄링을 수행하는 각 프로세스들은 커널 내의 실시간 프로세스와 1:1로 연결되어 수행한 후 클라이언트에게 응답 패킷을 전송한다.

응용프로그램 수준에서는 각 서버의 부하 정보를 비교하기 위한 CPU Monitor가 존재하며 동작 과정은 다음과 같다.

① CPU Monitor가 각 실제 서버의 부하 정보를 수집한다.
② 수집된 각각의 부하 정보를 비교하여 부하가 가장 적은 실제 서버의 IP주소를 선택한다.
③ 서비스 되던 기존 서버의 IP 주소는 ②과정에서 선택된 IP 주소로 수정한다. 그리고 클라이언트로부터 수신된 패킷은 수정된 IP 주소의 실제 서버로 전달한다.

### 3.2 시스템 모니터링

제안 시스템은 SNMPv2 의 부하 관련 MIB-Ⅱ 값을 가공하여 이더넷 이용률과 시스템 부하율을 분석해 실제 서버의 부하를 분석 한다.

#### 3.2.1 이더넷 이용률(Ethernet Utilization)

본 논문이 제시하는 시스템의 부하 분석은 이더넷 이용률에 시스템 이용률을 더한 값으로 부하 분석 식은 식 (1)과 같다. 이더넷 이용률은 식 (2)와 같이 부하 제어기의 모든 입·출력 트래픽 량을 의미 한다. 즉, 송신측에서 전송한 패킷의 총 비트 수와 수신측에서 받은 총 비트 수를 합하여 네트워크 링크의 전체 대역폭으로 나눠주면 된다.

이더넷 이용률을 측정하기 위해 본 논문에서 사용된 변수는 <표 2>와 같다.

표 2. 이더넷 이용률

| 항 목 | 설 명 |
|---|---|
| x | 폴링 주기에서 이전 폴링시간 |
| T | 폴링 주기 |
| ifInOctets | 부하 제어기에 수신된 옥텟의 총 수 |
| ifOutOctets | 부하 제어기 외부로 전송되는 옥텟의 총 수 |
| sysUpTime | 시스템이 마지막으로 재 초기화된 이후의 시간 |
| ifSpeed | 부하 제어기의 현재 대역폭, bps(bit per second)로 표시 |

이더넷 트래픽의 이용률 분석을 위한 처리 과정은 다음과 같고, 식 (3)에서 이더넷 이용률 분석 식을 보인 것이다.

① ifInOcetet의 변화량과 ifOutOctet의 변화량을

$$Total\_Load = Ethernet\_Utilization + Sys\_Utilization \quad (1)$$

$$Ethernet\_Utilization = Input/Output\ traffic\ of\ Load\_balancer.$$
$$Input/Output\ traffic\ of\ Load\_balancer = (total\_bit\_sent + total\_bit\_received)/bandwidth. \quad (2)$$

$$Ethernet\ Utilization: \quad (3)$$
$$\frac{(\text{if } InOctets_{(x+t)} - \text{if } InOctets_{(x)} + \text{if } OutOctets_{(x+t)} - \text{if } OutOctets_{(x)}) \times 8}{(sysUpTime_{(x+t)} - sysUpTime_{(x)}) \times \text{if } Speed \times 10} \times 100$$

구해서 합산한다.

② 비트 단위로 변환한다.

③ 측정한 시간 값으로 나눠준다.

④ 1초당 선로의 최대이용값(ifSpeed)으로 나눈다.

⑤ % 단위로 변환한다.

3.2.2 시스템 이용률(System Utilization)

시스템 이용률은 식 (4)와 같으며, 메모리 이용률, CPU 평균 사용률, 디스크 이용률을 더한 값으로 나타낸다. 본 논문에서 실제 서버의 시스템 이용률을 분석하기 위해서 사용된 항목은 <표 3>과 같다.

표 3. 시스템 이용률

| 항 목 | 객 체 설 명 |
|---|---|
| memTotalSwap | Swap 영역의 전체 크기 |
| memAvailSwap | 사용 가능한 Swap 영역 |
| memTotalReal | 물리적 메모리 크기 |
| memAvailReal | 사용 가능한 물리적 메모리 크기 |
| memTotalFree | 전체 사용가능한 메모리 크기 |
| laLoad_x | x분 동안의 CPU 평균 부하 |
| dskTotal | 디스크의 전체 크기 |
| dskAvail | 사용가능한 디스크 크기 |
| dskUsed | 사용된 디스크 크기 |

식 (5)는 memTotalSwap 값과 memAvailSwap 값을 이용해 Swap 메모리의 이용률을 구하기 위한 것이다. 식 (6)은 memTotalReal 값과 memAvailReal 값을 이용하여 물리적 메모리 사용률을 구한다.

식 (7)은 x분 동안 CPU 평균 사용률을 %단위로 구하는 것이며, 식(8)에서의 dskToral값과 dskUsed 값을 이용하여 디스크 이용률을 구한다.

3.2.3 SNMP PDU Type

SNMP 모듈이 GetNextRequest PDU(Protocol Data Unit)를 사용하여 실제서버에게 부하 관련 정보를 요청 하면, 에이전트 역할을 수행하는 실제서버는 Response PDU를 통하여 수집된 정보를 부하 제어기에게 응답한다. 이 과정은 주기적으로 이루어지게 되며 수집된 데이터는 가공 처리되어 로그 파일로 저장된다.

[그림 6]은 부하 제어기에서 실제 서버로 부하 정보를 요구 할 때 사용되는 GetNextRequest PDU 메시지 구조이다.

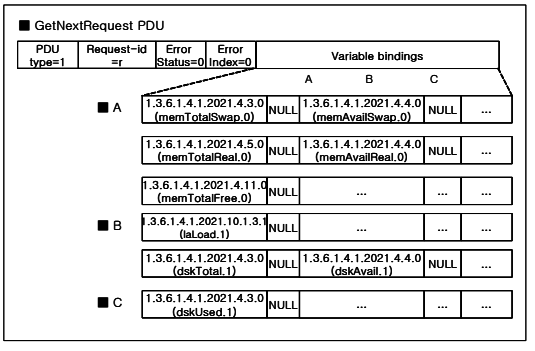


그림 6. GetNextRequest PDU type

[그림 6]에서 PDU 타입은 GetNextRequest PDU 메시지를 의미하는 1로 설정하고, Request ID 필드(field)는 관리자의 요청을 에이전트의 응답과 연관시켜 주는 값이다. Error Status와 Error Index는 에러가 없음을 의미하는 0으로 채운다. Variable bindings는 객체 식별자(OID: Object IDentifier)와

$$Sys\_Utilization = memSwapLoad + laLoad + dskLoad \quad (4)$$

$$memSwapLoad: \frac{memTotalSwap - memAvailSwap}{memTotalSwap} \times 100 \quad (5)$$

$$memRealLoad: \frac{memTotalReal - memAvailReal}{memTotalReal} \times 100 \quad (6)$$

$$laLoad = laLoad\_x \times 100 \quad (7)$$

$$dskLoad: \frac{dskUsed}{dskTotal} \times 100 \quad (8)$$

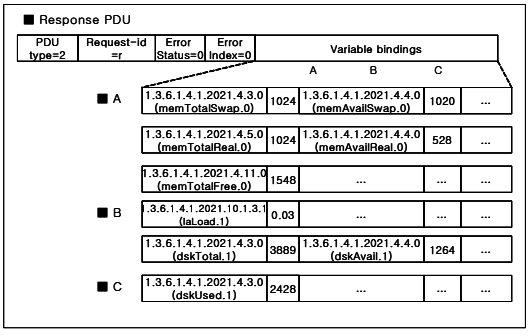


그림 7. Response PDU type

그에 할당된 값(NULL)의 목록을 의미한다.

[그림 7]은 실제 서버가 부하 제어기에서 요청한 부하 정보에 대한 결과 값을 전송 할 때 Response PDU 메시지 구조를 보여준다. PDU 타입은 Response PDU 메시지를 의미하는 2로 설정하고, Request-ID는 GetNextRequest PDU와 Response PDU를 연관시키는 값으로 설정한다.

Error Status와 Error Index 값은 에러가 없는 경우 0으로 채운다. Variable bindings는 GetNextRequest PDU 메시지에서 요청한 객체 식별자와 관련된 값들로 채운다.

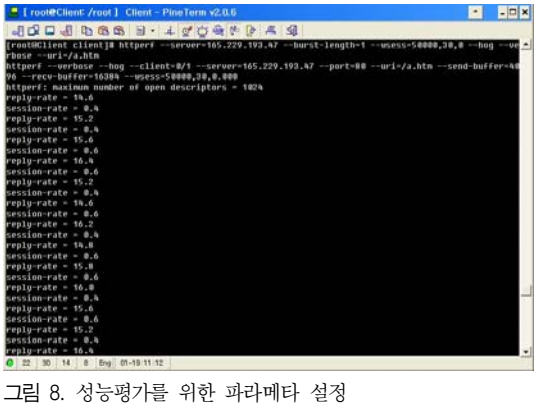


그림 8. 성능평가를 위한 파라메타 설정

## Ⅳ. 실측실험을 통한 성능분석

본 논문에서 제시한 클러스터링 웹서버 환경에서 차별화 서비스를 위한 3단계 동적 부하분산 기법은 Linux Kernel 2.4.7 운영체제상에서 구현되었으며, CPU Monitoring 하기 위해 사용된 MIB의 구성은 UCD-SNMP를 이용한다.

실측 실험을 위해 클라이언트 2대, 부하 제어기 1대, 서버 6대, 모니터링 서버 1대를 네트워크로 연결하여 실험 환경을 구성한다. 웹 서버는 Apache Web Server 2.4.17의 핵심 부분(스케줄링, 커널부분과 매핑)을 수정하여 이용한다. 본 논문에서는 웹 서버의 성능을 평가하기 위해 HP 사의 Httperf (HTTP performance)프로그램과 아파치 서버의 응답속도를 측정하는 벤치마킹 툴인 AB(Apache HTTP server Benchmarking tool)를 이용한다.

[그림 8]은 현재 웹 서버의 성능평가를 위해 Httperf 프로그램을 이용하여 클라이언트에서 부하제어기로 시스템 부하 요청을 보내는 과정이다. 웹 서버의 환경 설정은 전체 연결 개수 : 50000, 동시 접속자수 : 30명, 한 연결 당 요청 개수 : 1로 테스트를 수행한다. 이를 설정하기 위한 옵션을 정리 하면 아래와 같다.

① server : 웹 서버 주소를 나타낸다.
② burst-length : 각 연결마다 동시 접속자수를 나타낸다.
③ wsess=N1, N2, X : N1은 전체 session의 수, N2는 동시 접속자수, X는 time-out을 나타낸다.
④ hog : 이 옵션을 켜 놓지 않으면 제한된 (1024-5000) 포트를 사용하지만 이 옵션을 설정하면 가능한 많은 TCP 포트를 사용한다.
⑤ verbose : 5초마다 응답률(reply rate)등을 화면에 나타낸다.
⑥ uri : 웹 페이지를 요청 할 때 사용한다.

[그림 9]는 모니터링 파라메타를 이용하여 모니터링 서버에서 실제 서버의 부하 정보를 모니터링 하기 위한 인터페이스이다. [그림 9]에서 제공되는 내용은 이더넷 이용률과 시스템 이용률, 전체 이용률이 라인 그래프로 표시되며, 그래프에서 X축은 시간의 변화를 나타내고, Y축은 이용률을 %로 나타낸다.

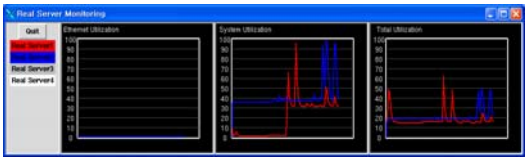


그림 9. 모니터링 인터페이스

### 4.1 실험 1 : 웹 서버 비 과부하 상태 실험

실험 1은 웹 서버에 과부하가 발생 하지 않았을 경우를 실험하였다. 가상 IP는 165.229.193.50이고 전체 연결 개수는 50000, 동시 접속자는 1명, 한 연결 당 요청 개수는 50으로 테스트 한다.

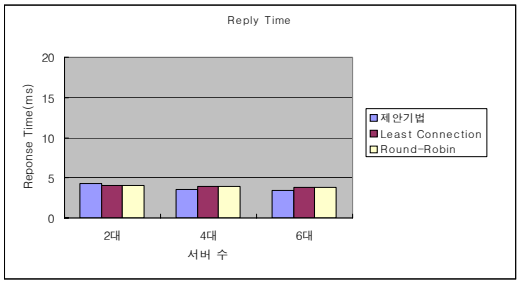


그림 10. 실험 1에 대한 성능평가

[그림 10]은 실제 서버의 증가 수에 대한 스케줄링 알고리즘 방법에 따라 응답 시간(response time)을 체크하여 그래프로 나타낸 것이다. 웹 서버에 과부하가 발생 하지 않았을 때의 응답 시간은 3가지 방법 모두 큰 차이를 보이지 않았다. 이와 같은 상황에서는 모두 서비스를 완벽하게 잘 받고 있기 때문에 차별화된 서비스가 필요하지 않은 상황이다.

### 4.2 실험 2 : 웹 서버에 과부하가 발생한 경우

실험 2는 웹 서버에 과부하가 발생한 경우를 실험 하였다. 가상 IP는 실험 1에서와 같이 165.229.193.50 이고 전체 연결 개수를 50으로 하고 동시 접속자는 30명이고 한 연결 당 요청 개수를 50000으로 실험 한다.

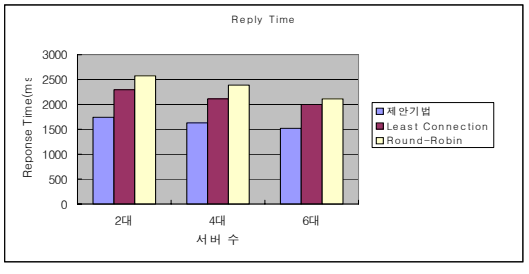


그림 11. 실험 2에 대한 성능평가

본 논문에서 제안한 시스템은 실제 서버의 이더넷 이용률과 시스템 이용률을 측정 하여 부하 분산에 적용함으로써 [그림 11]에서 알 수 있듯이 최소 접속 스케줄링 보다는 1.3배, 라운드-로빈 스케줄링은 1.5배 정도 성능 향상이 있었다.

### 4.3 실험 3 : 특정 웹 서버에 과부하 발생

실험 3은 실제 서버 1에서 과부 집중 시 실험 하였다. 실험 환경은 실험 1과 같지만, 실제 서버 1의 CPU 부하(load)가 증가하도록 스크립트 코드를 작성하여 웹 서버에 과부하가 발생하도록 하였다.

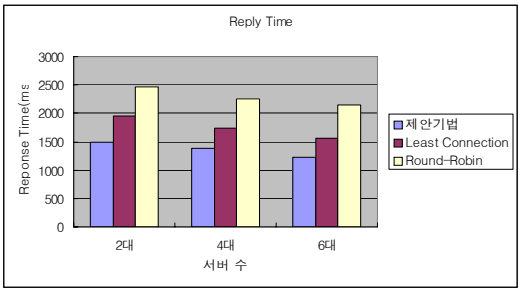


그림 12. 실험 3에 대한 성능평가

실험 3은 제안한 방법이 다른 스케줄링 알고리즘 방법에 비해 1.3~1.6 배의 성능 향상이 있었다. 이는 일정 주기로 각 실제 서버의 현재 부하를 측정해서 부하 분산을 정밀하게 하기 때문에 다른 스케줄링 알고리즘에 비해 성능이 향상된 것이다.

## V. 결 론

본 논문에서는 리눅스 환경에서 SNMPv2의 시스템 부하관련 MIB-Ⅱ정보를 검출하여 부하 분산에 적용하는 3단계 동적 부하 분산 시스템을 제안한다. 이 시스템은 네트워크 주소 변환 기법과IP 수준의 가장법, 터널링 기법을 사용하여 부하 제어기를 구현하므로 응답 속도를 개선하고, 병목 현상을 제거한다. 또한, SNMPv2의 MIB-Ⅱ 정보를 이용하여 실제 서버의 부하를 분석 할 수 있는 항목을 정의한다. 마지막으로 분석한 결과를 부하 제어기에 보내주어 부하가 많은 실제 서버로 클라이언트 요청이 가지 않도록 부하 분산시스템을 구현함으로써 동적 부하분산을 제공한다.

본 논문에서 제안한 시스템의 성능 분석을 위해 웹 서버에 과부하가 발생 하지 않았을 때와 과부하가 발생하였을 때 실험을 하였다. 실험 결과, 과부하가 발생 하지 않았을 경우에는 성능의 차이를 보이지 않는다. 그러나 웹 서버에 과부하 발생하였을 때는 다른 스케줄링 알고리즘 방법 보다 우수한 성능을 얻을 수 있었다.

그 이유는 실제 서버의 현재 부하 정보를 측정해서 부하 분산을 보다 정밀하게 수행하기 때문에 다른 스케줄링 방법에 비해 성능이 향상된 것으로 파악된다. 제안 시스템을 이용하여 웹 서버를 구축 한다면 이 시스템은 기존의 정적 부하 분산 시스템보다 정확한 부하 분산을 수행함으로 응답 속도를 개선할 수 있다. 또한, 정보의 중요도나 정보를 제공받는 사용자의 중요도에 따라 차별화된 품질 보장

을 제공하며 웹 서비스의 신뢰성과 효율성을 보장한다.

향후 과제로는 본 논문에서 실험 하지 못한 가중치 기반 라운드로빈, 가중치 기반 최소 접속 스케줄링 알고리즘에 대한 실험이 지속적으로 이루어져야 하고, 클러스터링 시스템을 통합 관리 할 수 있는 관리 시스템의 연구와 개발이 필요하다.

## 참 고 문 헌


[1] Wensong Zhang, "Linux Virtual Server Project", http://proxy.iinchina.net/~ippfv/, May 1998.

[2] C.Picoto, P. Veiga, "Management of a WWW Server Using SNMP", In 6th Joint Europeon Networking Conference, 1995.

[3] K. McCloghrie, M. Rose, Management Information Base for Network Management of TCP/IP-based Internets: MIB-II, RFC 1213, March 1991.

[4] A. Robertson, "High-Availability Linux Project", May 1998 , http://www.linuxha.org.

[5] William Stallings, "SNMP, SNMPv2, SNMPv3 and RMON: 3rd Ed", Addison- Wesley, 1999.

[6] Linux Virtual Server Project, "http://www.linuxvirtualserver.org/how.html"

[7] T. Brisco, "DNS support for load balancing", http://www.ietf.org/rfc/rfc1794.txt.

[8] Chad Yoshikawa, el al., "Using Smart Clients to build scalable services", USENIX '97, 1997

[9] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed, "NCSA's World Wide Web Server : Design and Performance", IEEE Computer, pp.68-74, November 1995.

[10] A. Dahlin, M. froberg, J. Walerud and P. Winroth, "EDDIE: A Robust and Scalable Internet Server", http://www.eddieware.org, 1998.

[11] Ralf S.Engelschall, "Load Balancing Your Web Site: Practical Approaches for Distributing HTTP Traffic", Web Techniques Magazine, Volume 3, Issue 5, May 1998.

[12] Edward Walker, "pWEB A Parallel Web Server Harness", http://www.ihpc.nus.edu.sg/STAFF/edw, April 1997.

[13] Daniel Andresen, Tao Yang, Oscar H. Ibarra, "Towards a Scalable Distributed WWW Server on Workstation Clusters", Proc. Of 10th IEEE Intl. Symp. Of Parallel Processing(IPPS'96), Jun 1996, pp.850-856

[14] Eric Anderson, Dave Patterson, and Eric Brewer, "The Magicrouter: an Application of Fast Packet Interposing", May 1996.

[15] Cisco System "Cisco Local Director", http://www.cisco.com/.

[16] P. O'Rourke and M. Keefe, "Performance Evaluation of Linux Virtual Server", LISA 2001, July 2001.

[17] G. Goldszmidt and G. Hunt, NetDispatcher: "A TCP Connection Router", IBM Research Technical Report RC 20853, July 1997.



이 명 섭 (Myung Sub Lee) 정회원

1998년 2월 경일대학교 컴퓨터공학 졸업(공학사)

2000년 2월 영남대학교 대학원 컴퓨터공학과 졸업(공학석사)

2003년 8월 영남대학교 대학원 컴퓨터공학과 졸업(공학박사)

2003년 3월~현재 영남대학교 전자정보공학부 객원교수

<관심분야> 망관리, 에이전트, QoS

박 창 현 (Chang Hyeon Park) 정회원

1986년 경북대학교 전자공학과 졸업(공학사)

1988년 서울대학교 계산통계학과 전산학전공(이학석사)

1992년 서울대학교 계산통계학과 전산학전공(이학박사)

1992년~1993년 서울대학교 컴퓨터신기술공동연구소 특별연구원

1998년~1999년 University of Maryland, Institute of Advanced Computer Systems, Visiting Researcher

1993년~현재 영남대학교 컴퓨터공학과 교수

<관심분야> 인공지능, 에이전트, 지능형 망관리