클러스터링 웹 서버 환경에서 차별화 서비스를 위한 동적 부하분산 시스템

이명섭° 김환섭 박창현
영남대학교
skydream@yu.ac.kr tobluueing@hotmail.com park@yu.ac.kr

A load balancing system for Differentiated Service in clustering web server

Myungsub Lee° Hwansub Kim Changhyeon Park
Dept. of Computer Engineering, Yeungnam University

요 약

인터넷 사용자의 폭발적인 증가로 인하여 차별화된 웹 서비스를 제공해주는 웹 응용프로그램들의 개발이 요구되고 있다. 그러므로 웹 서버내의 품질향상을 보장해주는 웹 Qos 기술은 전자상거래나 웹 호스팅 같은 부분에서 점점 더 중요한 문제로 대두되고 있다. 그러나 대두분의 웹 서버들은 FIFO 방식의 최선 서비스만을 제공하며, 정보의 중요도나 정보를 제공받는 사용자의 중요도에 따라 차별화된 품질보장을 제공하지 못한다. 본 논문에서는 클러스터링 웹 서버 환경에서 차별화 서비스를 위한 동적 부하분산 시스템을 제안한다. 웹 서비스의 차별화된 품질보장을 제공하는 웹 서버 구현을 위해 커널 수준 접근 방식과 응용프로그램 접근방식을 제시한다. 제안 시스템에서는 웹 서비스의 신뢰성과 반응속도를 개선시키기 위해 IP수준의 가장법과 터널링 기술을 이용하여 웹 서버의 부하를 분산한다. 그리고 SNMP의 시스템 부하관련 MIB-II 정보를 검출하여 부하 분산에 반영함으로써 동적인 부하분산이 가능하도록 한다.

1. 서 론

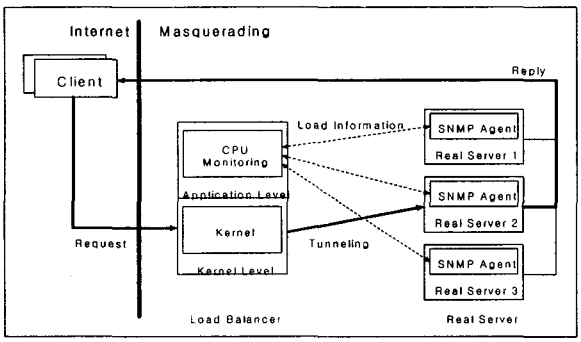
최근 몇 년 동안 인터넷의 사용이 대중화되고, 사용자 또한 급속도로 증가함에 따라 웹 페이지에 대한 접속은 폭발적으로 증가하고 있다. 이런 수요 증가에 비해 제공되는 서비스들의 품질은 그만큼 따라가지 못하는 것이 현실이다. 그 주요 원인은 웹 서버의 서비스 성능의 부족에서 오는 것이며, 클러스터형 웹 서버가 중요한 해결 수단으로 제시되고 있다. 클러스터형 웹 서버란 수많은 사용자들의 요청을 처리하기 위해 서버들을 네트워크로 연결한 것으로 사용자들의 요청을 서버들이 나누어 처리하는 구조를 갖는 웹 서버를 말한다. 클러스터 시스템(cluster system)은 이와 같은 부하 분산(load balancing)을 하기 위해서 라운드 로빈(round-robin), 가중치 기반 라운드로빈(weighted round-robin), 최소 접속(least connection), 가중치 기반 최소 접속(weighted least-connection)과 같은 방법의 스케줄링 알고리즘(scheduling algorithm)을 사용한다.[1]
기존의 클러스터 시스템의 경우 단순히 스케줄링 방식에 의해서 클라이언트 요구를 실제 서버(real server)에게 전달하는 기능만을 가지고 있으며, 부하 분산 방법에서 실제 서버상의 프로세스 부하(load)정보를 고려하지 않는 스케줄링 방식에 대한 연구가 대부분이다. 본 논문에서는 클러스터링 웹 서버 환경에서 차별화 서비스를 위한 동적 부하분산 시스템을 제안한다. 제안 시스템은 실제 서버 상에서의 프로세스 부하정보를 이용하여 실제 서버상의 부하 정도에 따라 동적 부하 분산을 제공한다. 이를 위해 SNMPv2(Simple Network Management Protocol Version 2)[2][3]의 시스템 부하 관련 MIB-II(Management Information Base-II)[4] 정보를 검출하여 부하 분산에 반영함으로써 동적인 부하 제어기(load balancer)의 구현이 가능하도록 한다. 또한, 네트워크 주소 변환 기법(NAT: Network Address Translation)과 IP 터널링(tunneling) 기법[5]을 결합하여 IP수준 스케줄링에서 발생하는 패킷 오버헤드(overhead)와 부하 제어기의 병목 현상(bottleneck)을 제거한다. 따라서 기존 논문의 클러스터링 구성 방식에 비해 응답 속도 및 사용자 중요도에 따라 차별화된 품질 보장을 제공하며 웹 서비스의 신뢰성과 효율성을 보장한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 본 논문에서 제안하는 시스템에 관하여 기술한다. 3장에서는 본 논문에서 제안한 시스템의 테스트 환경과 실험 결과를 보인다. 마지막으로 4장에서는 결론과 앞으로의 연구 방향을 기술한다.

2. 제안 시스템

본 논문에서는 클라이언트의 요청에 우선순위를 제공하기 위하여 웹 서버에서 컨텐츠를 분류하고, 스케줄링을 수행

한다. 또한, 커널 수준의 실시간 스케줄러에게 프로세스를 매핑(mapping)하는 커널 수준 접근법을 지원하며, 웹 서비스의 신뢰성 보장과 응답속도를 개선하기 위하여 IP 수준의 가장법과 터널링 기술을 이용하여 웹 서버의 부하를 분산하는 시스템이다. [그림 1]에서 제안 시스템의 구성을 보이며 구성 요소들에 대한 설명은 다음의 각 절에 주어진다.



[그림 1] 제안시스템 전체구성도

2.1 부하제어기

부하 제어기의 구조는 커널 수준(kernel level)과 응용프로그램 수준(applicationlevel)으로 구성된다. 커널 수준에서는 웹 서비스의 신뢰성보장과 응답 속도를 개선하기 위하여 IP 수준의 가장법과 터널링 방법이 사용된다. 또한, 커널 수준에서는 하나의 요청에 대한 효율적인 우선순위를 보장하기 위한 것으로, 아파치 웹 서버의 스케줄러가 커널 내부의 실시간 스케줄러에게 프로세스를 맵핑하는 방법으로 시스템을 구현한다. 클라이언트의 요청이 NIC(Network Interface Card)를 통해서 들어오면 웹 서버는 TCP listen 버퍼에서 80번 포트로 요청을 받아들이고, 연결 관리자에서 분류 정책(클라이언트 IP, URI, 파일이름, 디렉토리, 사용자 인증 등)에 따라 각 연결별로 분류를 한다. 분류된 요청은 우선순위를 부여하여 각각의 요청 큐에 입력되고 부하 제어기에서 획득한 모니터링 정보와 결합되어 부하 분산 된다. 이때, 웹 서버에서 스케줄링을 수행하는 각 프로세서들은 커널 내의 실시간 프로세서와 1:1로 연결되어 수행한 후 클라이언트에게 응답 패킷을 전송한다.

응용프로그램 수준은 컨텐츠 추출 모듈과 분류 모듈 그리고 CPU Monitor로 구성된다. 클라이언트로부터의 요청이 들어오면 IP 계층의 컨텐츠 추출과 분류 모듈에서 분류 작업을 수행한다. IP 계층에서 수신된 요청 메시지를 다루기 위해서 본 논문에서는 리눅스 커널의 자료구조인 'sk_buff'를 이용하며, 이 자료구조의 포인터를 이용하여 HTTP 요청 데이터의 패스(path) 부분을 읽어온다. 또한, 각 서버의 부하 정보를 비교하기 위한 CPU Monitor가 존재하며 동작 과정은 다음과 같다.

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

2.2 모니터링 파라메타

제안 시스템은 SNMPv2 의 부하 관련 MIB-II 값을 가공하여 이더넷 이용률과 시스템 부하률을 분석해 실제 서버의 부하를 분석 한다[6][7]. 본 논문이 제시하는 시스템의 부하 정보 분석은 모든 이더넷 이용률에 시스템 이용률을 더한 값으로 표현된다. 즉, 송신측에서 전송한 패킷의 총 비트 수와 수신측에서 받은 총 비트 수를 합하여 네트워크 링크의 전체 대역폭으로 나눠주면 된다. 본 논문에서 이더넷 이용률을 측정하기 위해 사용된 변수는 [표 1]과 같으며, 식 (1)에서 이더넷 이용률 분석 식을 보인 것이다.

<표 1> 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, bps로 표시

<표 2> 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

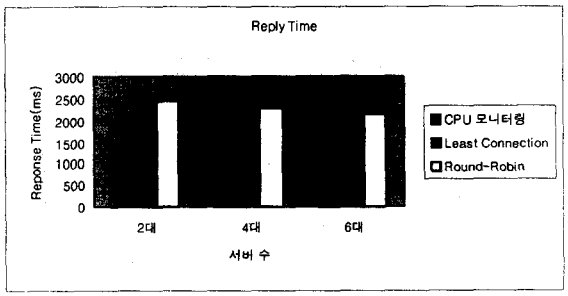
시스템 이용률은 메모리 이용률, CPU 평균 사용률, 디스크 이용률을 더한 값이다. 본 논문에서 실제 서버의 시스템 이용률을 분석하기 위해서 사용된 항목은 <표 2>와 같다.

식 (2)에서는 memTotalSwap 값과 memAvailSwap 값을

이용하여 Swap 메모리의 이용률을 구한다. 식 (3)에서는 memTotalReal 값과 memAvailReal 값을 이용해 물리적 메모리 사용률을 구한다. 식 (4)에서는 x분 동안 CPU 평균 사용률을 %단위로 구하며, 식 (5)에서는 dskTotal 값과 dskUsed 값을 이용해 디스크 이용률을 구한다.

3. 실험 및 결과 분석

본 논문에서 제시한 차별화 서비스를 위한 부하 분산 시스템은 Linux Kernel 2.4.7 운영체제 상에서 구현되었으며, CPU Monitoring 하기 위해 사용된 MIB의 구성은 UCD-SNMP를 이용한다. 시스템의 동작을 실험하기 위해 클라이언트 2대, 부하 제어기 1대, 서버 6대, 모니터링 서버 1대를 네트워크로 연결하여 실험 환경을 구성한다. 웹 서버는 Apache Web Server 2.4.17을 수정하여 이용한다.



[그림 3] 특정 웹 서버에 과부하가 발생한 경우

본 논문에서 수행한 실험은 실제 서버 1의 CPU 부하(load)가 증가하도록 스크립트 코드를 작성하여 특정 웹 서버에 과부하가 발생하도록 하였다. [그림 3]은 실제 서버의 증가 수에 대한 스케줄링 알고리즘 방법에 따라 응답 시간(response time)을 체크하여 그래프로 나타낸 것이다. 이 실험에서 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.

4. 결론

본 논문에서는 리눅스 환경에서 SNMPv2의 시스템 부하관련 MIB-II정보를 검출하여 부하 분산에 적용하는 동적 부하 분산 시스템을 제안한다. 이 시스템은 네트워크 주소 변환 기법과 IP 수준의 가장법, 터널링 기법을 사용하여 부하 제어기를 구현하므로 응답 속도를 개선하고, 병목 현상을 제거한다. 본 논문에서는 SNMPv2의 MIB-II 정보를 이용하여 실제 서버의 부하를 분석 할 수 있는 항목을 정의한다. 또한, 이를 분석한 결과를 부하 제어기에게 보내주어 부하가 많은 실제 서버로 요청이 가지 않도록 부하 분산시스템을 구현함으로써 기존의 시스템 보다 향상된 성능을 보인다.

5. 참고문헌

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

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

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

[4] K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II," RFC1213, March 1991.

[5] A. Robertson, "High-Availability Linux Project", May 1998

[6] 안성진, "TCP/IP망 관리를 위한 시스템 분석 파라미터 계산 알고리즘", 성균관대학교 대학원 박사학위 논문, 1997.

[7] 이명섭, 박창현, "SNMP 분석 파라메타를 이용한 웹 기반의 트래픽 제어 환경 설계", 영남대학교 정보통신연구소 논문집, 제 8권 제 2호, pp. 59-69, 2002년 6월.

Ethernet Utilization:

$$\frac{(\text{if } InOctets_{(x+t)} - \text{if } InOctets_{(x)} + \text{if } OutOctets_{(x+t)} - \text{if } OutOctets_{(x)}) \times 8}{(sysUpTime_{(x+t)} - sysUpTime_{(x)}) \times \text{if } Speed \times 10} \times 100 \quad \text{(식 1)}$$

$$memSwapLoad \frac{memTotalSwap - memAvailSwap}{memTotalSwap} \times 100 \quad \text{(식 2)}$$

$$memRealLoad \frac{memTotalReal - memAvailReal}{memTotalReal} \times 100 \quad \text{(식 3)}$$

$$laLoad = laLoad\_x \times 100 \quad \text{(식 4)}, \qquad dskLoad \frac{dskUsed}{dskTotal} \times 100 \quad \text{(식 5)}$$