커널 수준 실시간 스케줄링과 부하 분산을 통한 차별화된 웹 서비스 시스템

정회원 이 명 섭*, 박 창 현*

A Differentiated Web Service System through Kernel-Level Realtime Scheduling and Load Balancing

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 웹 QoS 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 웹 서비스의 차별화된 품질보장을 제공하는 웹 서버 구현을 위한 두 가지 접근 방식을 제시한다. 첫째는 커널 수준 접근방법으로, 커널 상에 실시간 스케줄링 프로세서를 두어 웹 서버에서 수행중인 스케줄링 프로세서와 연동시키고, 커널 내부에서도 웹 서버에서 할당된 사용자 요청 우선순위를 유지하도록 한다. 둘째는 부하분산 접근방법으로, IP 수준의 가장법과 터널링 기술을 이용하여 웹 서버의 부하를 분산하여 웹 서비스의 신뢰성을 보장하고 응답속도를 개선한다.

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 two approaches to provide differentiated Web QoS. The first is the kernel-level approach, which is adding a real-time scheduling processor to the operating system kernel to maintain the priority of user requests determined by the scheduling processor 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.

Ⅰ. 서 론

World Wide Web(이하 웹)은 저렴한 가격과 다양하고 흥미 있는 정보를 쉽고 간편하게 찾아볼 수 있다는 장점으로 웹 사용자는 빠르게 증가되고 있으며 웹 사용자의 증가와 함께 웹을 통해 전달되는 데이터 즉, 웹 문서, 그림, 멀티미디어 데이터 등의 크기 또한 빠르게 증가되고 있다[1]. 웹의 급격한 보급과 각종 멀티미디어 데이터를 포함하는 웹 서비스의 팽창으로 인하여 웹 서버내의 서비스 품질보장을 위한 웹 QoS 기술은 웹 서비스[2,3] 부분에서 점점 더 중요한 문제로 대두되고 있다. 차별화된 웹 서비스를 제공하기 위해서는 정보의 중요도와 정보를 제공받는 사용자의 중요도에 따라 컨텐츠를 분류하는 모듈과 분류된 컨텐츠를 스케줄링 할 수 있는 모듈이 웹 서버에 포함 되어 있어야 한다. 그러나 대부분의 인

* 영남대학교 컴퓨터공학과 인공지능 및 지능정보시스템 연구실(skydream@cse.yu.ac.kr)
논문번호 : 030021-0113, 접수일자 : 2003년 1월 13일

터넷 서버들은 이러한 컨텐츠의 종류와는 무관하게 단지 FIFO방식의 스케줄링으로 최선 서비스(best effort service) 만을 제공하고 있기 때문에 차별화된 웹 서비스를 제공하지 못한다[5].

일반적으로 가장 많이 사용되고 있는 아파치 웹 서버[4]의 경우, 웹 서버상의 요청 형태를 알고 있음에도 FIFO 방식으로 요청을 처리하고 있다. 따라서 서비스 품질을 보장하고 서비스를 특정 분류 기준에 따라 분류하여 차별화 서비스를 제공하는 새로운 서버 개발은 반드시 필요하다. 차별화 서비스를 제공하는 웹 서버를 구현하는 방법에는 사용자 수준 접근법과 커널 수준 접근법이 있다. 사용자 수준 접근법은 웹 서버 부분만 수정하여 차별화 서비스를 제공하고 있기 때문에 완전한 QoS를 보장해 주지 못한다. 따라서 웹 서버의 분류 정보를 커널 수준의 스케줄러와 1:1로 맵핑시켜 처리하는 커널 수준 접근법이 필요하다.

웹 사용량은 폭발적으로 증가하고 있으나, 이런 수요 증가에 비해 제공되는 웹 서버의 성능부족으로 인해 차별화 서비스를 제공하는 웹 서버를 구축하더라도 완전한 서비스 품질을 보장하지 못한다. 이를 위해 부하 분산형 웹 서버가 중요한 해결수단으로 제시 되고 있다[13]. 부하 분산형 웹 서버는 급격히 늘어나는 서비스/접속요청에 신속한 응답을 제공하고 안정적이고 유연성 있는 서비스를 제공함으로써 서비스 품질보장을 위해 반드시 필요하다. 그러나 기존연구에서 제시하는 부하분산 웹 서버 기술은 클라이언트의 응용프로그램 변경, 서버 과부하시 처리 불능, HTTP 요청/응답 처리의 과부하, 패킷 변환 오버헤드 등의 문제점이 있다. 따라서 클라이언트의 서비스 요청에 대한 완전한(complete) QoS를 제공하기 위해서는 웹 서버 구현 시 커널수준 접근법과 부하분산 접근법을 동시에 고려해야만 한다.

본 논문에서는 웹 서비스의 차별화된 품질보장을 제공하는 부하 분산형 웹 서버 구현을 위한 두 가지 접근 방식을 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 프로세서에도 유지될 수 있도록 함으로써 더 효율적인 차별화 서비스를 제공할 수 있도록 한다. 둘째, IP 가장법(masquerading)과 터널링(tunneling)기술을 이용한 분산 웹 서버를 구성하여 웹 서버의 부하를 클래스별로 분산함으로써 웹 서비스의 신뢰성과 응답속도를 개선한다.

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

II. 관련 연구

인터넷의 이 절에서는 웹 서버에서 차별화된 서비스를 제공하기 위한 기존의 연구방법들을 소개하고 그들의 문제점을 설명한다.

2.1 웹 QoS

항상 일정한 수준의 서비스를 보장 받을 수 있도록 해주는 기술을 QoS라고 한다. 그러나 기존의 인터넷 망에서는 패킷들이 어떤 경로를 통하여 목적지까지 도달하는지를 예상할 수 없기 때문에 QoS 보장이 힘들다. IETF는 QoS 요구사항을 만족시키기 위해 많은 서비스 모델과 메커니즘을 제안하고 있으며, 이 중 IntServ/RSVP 모델[16]과 DiffServ모델[17] 등의 연구가 활발하게 진행되고 있다. 그러나 확장성, 라우터의 고 기능 요구, QoS 기능 부재 등으로 인하여 현재의 인터넷 망에 적용이 늦어지고 있다. 최근에는 차별화 서비스 개념을 하나의 서버 내에 적용하여 클라이언트의 요청에 대해 일정 수준의 QoS를 보장하는 웹 QoS 개념이 연구되고 있다[3]. 웹 QoS는 클라이언트에서 입력된 요청을 웹 서버에서 받아 파일명, 사용자 ID, 클라이언트 IP 등과 같은 분류 기준에 따라 클래스 별로 분류하고, 각 클래스에 따라 서비스 품질을 차별화하여 서비스를 제공하는 개념이다. 이러한 차별화 된 웹 서비스를 제공하기 위한 서버의 설계 방식은 두 가지 접근방식이 있다. 첫 번째는 사용자 수준 접근법으로, 웹 서버에 차별화 모듈을 추가하여 웹 서버를 수정하는 방식이다. 이 방식은 웹 서버에서 수행되는 프로세서와 커널 내부에서 수행되는 프로세서가 1:1 맵핑이 되더라도 커널 내부에서 동작하는 프로세서가 동일한 스케줄링을 수행하지 않기 때문에 웹 서버에서 분류된 우선순위가 커널수준에서 보장되지 않는다는 단점을 가진다. 두 번째 접근 방법은 커널 수준 접근법이다. 이 방식은 웹 서버뿐만 아니라 커널에도 차별화 모듈을 추가하여 웹 서버에서 수행되는 프로세서와 커널내부에서 수행되는 프로세서가 1:1 맵핑될 때 커널 내부의 프로세서에도 동일한 스케줄링을 수

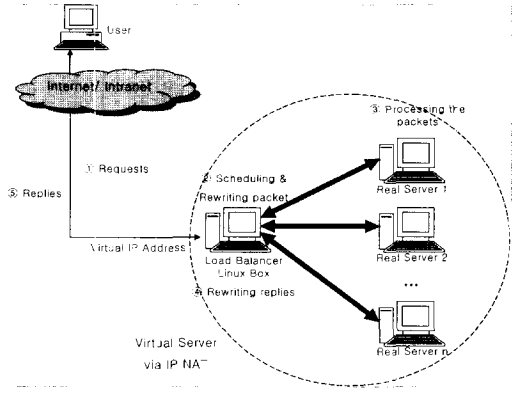
행하여 완벽한 QoS를 제공할 수 있다.

본 논문에서는 아파치 웹 서버에 분류와 스케줄링을 수행할 수 있는 모듈을 추가하고, 커널 레벨에서는 Montavista[15] 실시간 스케줄러를 이용하여 차별화 서비스를 수행할 수 있는 시스템을 제시한다.

2.2 부하 분산형 웹 서버

부하 분산형 웹 서버에 관한 기존연구는 크게 4가지로 클라이언트 측과 서버 측의 round-robin DNS, 응용프로그램 수준에서의 스케줄링, IP 수준에서의 스케줄링으로 나눌 수 있다[13]. 첫 번째 방법은 클라이언트 측 접근방법으로 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[6]가 이에 해당한다. 이 방법은 클라이언트 입장에서 보면 서버가 하나로 인식(transparent)되지 않고, 모든 클라이언트 어플리케이션 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 분산시키는 방법이다. NCSA의 scalable web server[7]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, Reverse-proxy, pWEB, sWEB 등이 여기에 속한다[8-11]. 이 방법은 분산되어 있는 서버들이 자신의 부하를 측정하여 클라이언트로부터 HTTP 요청이 들어올 경우, 자신의 부하 상태에 따라 처리 여부를 판단하여 만약 자신이 처리할 수 없으면 분산 서버 내의 다른 서버로 HTTP 요청을 전달하고, 결과를 되받아 이를 다시 클라이언트에게 최종적으로 전송한다. 이 방법의 단점은 두 번 이상의 TCP연결로 인해 전송지연이 심해지고 어플리케이션 수준에서 HTTP요청과 응답을 처리하는데 많은 오버헤드가 발생한다는 것이다. 마지막으로 서버 측에서의 IP 수준 스케줄링 방법이 있으며, 버클리 대학의 Magic router[12]와 Cisco의 Local Director[14]가 이 방식을 사용한다. 이 방법은 네트워크 주소 변환 기법(Network Address Translation : NAT)을 사용하여 서로 다른 서버 상의 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 실행서버(real server)들로 구성된다.

[그림 1]에서 NAT를 이용한 가상서버 구조를 보이고 있는데, 클라이언트가 분산 서버에서 제공하는 서비스에 접속하고자 할 경우, 클라이언트의 요구 패킷은 부하 제어기로 간다. 부하 제어기에서는 클라이언트에서 요청한 패킷의 목적지 주소와 포트 번호를 검사하게 된다. 그 내용이 가상 서버의 서비스와 일치하게 되면 스케줄링 알고리즘에 따라 부하 제어기에서 실제 서버를 선택하고, 해시테이블에 새로운 접속을 추가한다.



[그림 1] NAT를 이용한 가상서버 구성도

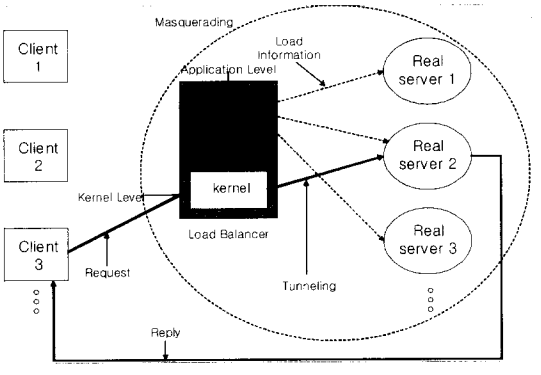
서버가 선택되면 부하 제어기에서는 패킷 변환(rewriting) 작업을 수행한 후 패킷을 서버로 전송(forward)한다. 응답 패킷에 대한 처리가 완료되면 서버에서는 부하제어기로 응답 패킷을 보내고, 부하 제어기에서는 응답패킷을 클라이언트에 전송하기 위해 다시 패킷 변환 작업을 수행한다. 접속이 해제되거나 시간을 초과하면, 해시테이블에서 연결기록을 제거한다. NAT방식은 실행 서버의 수가 20대를 넘어가게 되면 부하 제어기에 병목현상이 발생하며, 패킷 변환에 대한 오버헤드가 크다는 단점이 있다.

본 논문에서의 부하 분산형 웹 서버는 네트워크 주소변환 기법과 IP 터널링 기법을 결합하여 IP 수준 스케줄링에서 발생하는 패킷 변환 오버헤드와 부하 제어기의 병목 현상을 제거하며, 서버 과부하시 중요사용자에 대한 서비스를 보장하지 못하는 문제

점을 해결한다. 또한 DLC(Differentiated Least Connection) 알고리즘을 제안하여 서비스 클래스별로 연결수를 조사함으로써 기존의 LC 알고리즘[13]의 단점을 개선한다.

Ⅲ. 제안시스템

본 논문에서 제안하는 차별화된 웹 서비스를 제공하는 부하 분산형 웹서버는 클라이언트의 요청에 우선순위를 제공하기 위하여 웹 서버에서 컨텐츠를 분류하고 스케줄링을 수행하며, 커널 수준의 실시간 스케줄러에게 프로세스를 맵핑하는 커널 수준 접근법을 지원하며, 웹 서비스의 신뢰성보장과 응답속도를 개선하기 위하여 IP 수준의 가장법과 터널링 기술을 이용하여 웹 서버의 부하를 분산하는 시스템이다. 본 논문에서 제시하는 차별화된 웹 서비스 시스템의 전반적인 구성은 [그림 2]에서 보이고 있으며, 자세한 구조 및 동작에 대한 설명은 아래의 각 절에서 주어진다.

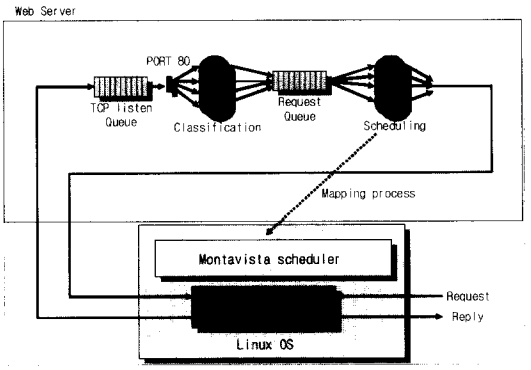


[그림 2] 시스템 전체구성도

3.1 커널 수준 접근법

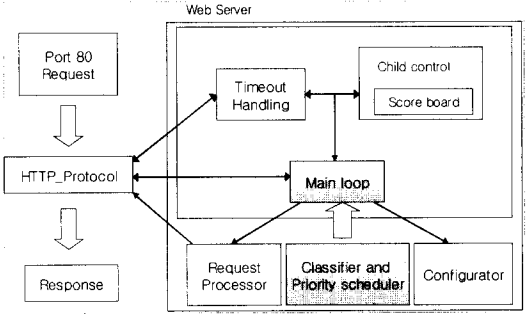
커널 수준 접근 방식은 하나의 요청에 대한 효율적인 우선순위를 보장하기 위한 것으로, 아파치 웹 서버의 스케줄러가 커널내부의 실시간 스케줄러에게 프로세스를 맵핑하는 방법으로 시스템을 구현한다. [그림 3]에서 커널 수준 접근법의 구조를, [그림 4]에서 수정된 웹 서버의 구성도를 보인다.

[그림 3]에서, 클라이언트의 요청이 NIC를 통해서 들어오면 웹 서버는 TCP listen 버퍼에서 80번 포트로 요청을 받아들이고, 연결 관리자에서 분류정책(클라이언트 IP, URL, 파일이름, 디렉토리, 사용자 인증 등)에 따라 각 연결별로 분류를 한다.



[그림 3] 커널 수준 접근법의 구조

분류된 요청은 우선순위를 부여하여 각각의 요청큐에 입력되고 DLC 스케줄링(3.3절) 정책에 따라 스케줄링 된다. 이때, 웹 서버에서 스케줄링을 수행하는 각 프로세서들은 커널 내의 실시간 프로세서와 1:1로 연결되어 수행한 후 클라이언트에게 응답 패킷을 전송한다.



[그림 4] 수정된 웹 서버 구성도

[그림 4]는 수정된 아파치 웹서버의 구성도이며, 수정된 아파치 웹 서버를 시작하게 되면 마스터 프로세서가 생성되고 이 프로세서는 prim-level, high-level, default 클래스를 가진 자식 프로세서를 생성하여 httpd를 구성한다. prim-level 클래스와 high-level 클래스는 실시간 스케줄링을 수행하도록 하며, 기본(default) 클래스는 일반 커널 스케줄러로 수행하도록 구성한다. 실시간으로 프로세서를 실행시키기 위하여 shched_setscheduler 시스템 콜 함수를 사용하며, 이 시스템 콜 함수는 프로세스 ID, policy(FIFO, Round-Robin), 우선순위와 같은 세 개의 인자를 가진다. 아파치 웹 서버의 conf 디렉토리 내의 class_ip.conf 파일은 클래스별로 파일 이름을 저장하고 있다. 그래서 아파치 웹 서버가 실행될 때 initialize_board라는

함수가 이 파일을 읽고 ctrl_levelizer 구조체에 파일 이름을 저장한다. 이 구조체는 파일 이름별로 우선순위 정보를 가지고 있다. 부모 프로세스가 생성(fork)시킨 자식프로세스가 이 구조체에 빠르고 쉬게 접근하게 하기 위해 본 논문에서는 공유 메모리 기법을 사용한다. <표 1>에서 아파치 웹 서버의 수정된 부분의 코드를, <표 2>에서 class_ip.conf 파일을 읽고, 공유메모리를 사용하기 위해 levelizer 구조체를 초기화하는 부분의 코드를 보인다.

<표 1> levelizer구조체

```
int id_level;
struct levelizer *ctrl_levelizer;
id_level = shmget(KEY_LEVEL,
                  SEGSIZE_LEVEL, IPC_CREAT|0666);
//  공유메모리를 사용하기 위해 세마포어를 만듦

ctrl_levelizer = (struct levelizer *)shmat(id_level,0,0);
// 공유메모리를 만듦

initialize_board(ctrl_levelizer);
//class_ip.conf 파일을 읽고, levelizer 구조체에 자료 채움
```

<표 2> 수정된 아파치 핵심코드

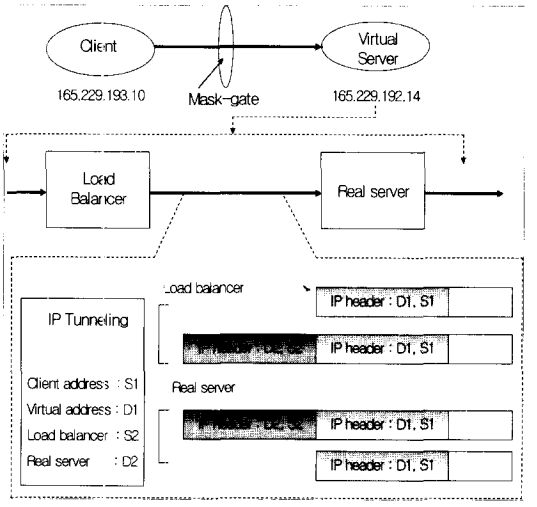
```
diff = determine_differentiated_level(r, ctrl _levelizer);
//클래스별로 분류
prio_struct.sched_priority=-1;
//우선순위 초기화
switch(diff_class) { //URI 추출하여 클래스를 정해줌
  case PRIM_LEVEL:
          policy = SCHED_FIFO;
          // FIFO 방식의 RT 스케줄러
          prio_struct.sched_priority = 90;
          //우선순위를 90
          p = getpid(); //프로세스 id를 얻음
          if(sched_setscheduler(p,policy,&prio_struct)){
            //시스템콜 함수 호출
          }
          break;
  case HIGH_LEVEL:
          policy = SCHED_FIFO;//FIFO방식 RT스케줄링
          prio_struct.sched_priority = 9; //우선순위를 9
          p = getpid();
          if(sched_setscheduler(p,policy,&prio_struct))
          /* p 프로세서에게 FIFO방식의 RT 스케줄러로
          우선순위 9를 설정하여 실행시키도록 명령*/
          break;
  default:
```

3.2 부하분산 접근법

본 논문에서 제시하는 부하 분산형 웹 서버는 IP 수준의 가장법과 터널링을 이용하여 패킷 전송 속도를 향상시키고 부하 제어기의 병목 현상을 제거함으로써 성능을 높이고 확장이 쉽다. 이는 1대의 부하 제어기가 클라이언트들로부터의 서비스 요청을 각 실행 서버로 분산시켜주는 부하 분산형 구조로 구성되고, 가장법을 이용하여 외부에서는 마치 단일 IP 주소를 가진 한대의 서버처럼 보이게 된다. 부하 제어기의 구조는 커널수준과 IP 수준으로 구성되며 각각의 구조와 동작방식은 다음의 세부 절에서 기술한다.

3.2.1 커널 수준

IP 가장법은 분산 서버들을 외부 네트워크상에서 게이트웨이 역할을 하는 오직 하나의 컴퓨터 뒤에 숨겨져 동작될 수 있도록 한다.

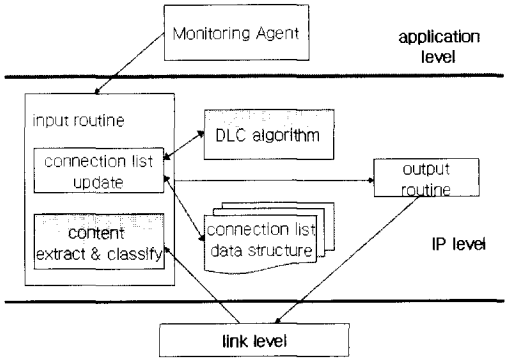


[그림 5] 커널 수준의 구조

[그림 5]에서 165.229.193.10이라는 실제 IP를 사용하여 클라이언트의 요청이 들어오면 mask-gate에서 접속정보를 변환하여 패킷을 전달함으로써 내부 네트워크상의 컴퓨터들은 인터넷을 통할 수 있으나, 클라이언트는 내부 서버들에 대한 접속정보를 알 수 없게 된다. 터널링 기법은 외부에 공개된 실제 IP 주소를 가진 IP 패킷 헤더에 내부 각 서버의 가상 IP 주소를 새로 추가하는 캡슐화(encapsulation)과정과 그 역과정인 역 캡슐화(decapsulation)과정을 수행한다. 이를 통하여 패킷을 받은 서버들이 다시 주소를 변환하지 않고도 요청 패킷의 IP 주소를 이용하여 직접 외부 네트워크로 데이터를 전송한다. 이렇게 함으로써 분산 서버 구성 시 웹 서비스의 신뢰성과 응답속도를 개선한다.

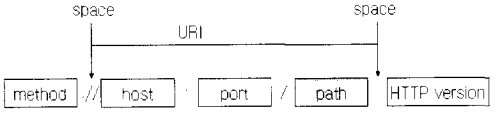
3.2.2 IP 수준

IP 수준은 컨텐츠 추출 모듈, 분류 모듈, DLC 알고리즘으로 구성된다. [그림 6]에서 IP 수준의 부하제어기의 구성도를 보인다.



[그림 6] IP 수준의 부하제어기 구성도

[그림 6]에서 클라이언트로부터의 요청이 들어오면 IP 계층의 컨텐츠 추출과 분류 모듈에서 분류와 스케줄링을 수행한다. IP 계층에서 수신된 요청 메시지를 다루기 위해서 본 논문에서는 리눅스 커널의 자료구조인 sk_buff를 이용하며, 이 자료구조의 포인터를 이용하여 HTTP 요청 데이터의 패스(path) 부분을 읽어온다. [그림 7]에서 HTTP 요청 라인(request line)의 형식을 보인다.



[그림 7] HTTP 요청 라인의 형식

HTTP 요청 라인은 요청 종류, 자원(URI), HTTP 버전 등을 정의하며, 요청 종류로는 GET, HEAD, POST등이 있다. GET은 클라이언트가 웹 서버로부터 문서를 읽어올 때 사용되는 요청이며, HEAD는 클라이언트에서 문서에 대한 정보를 원할 때 사용되는 요청이다. HEAD요청은 GET과 비슷하지만 웹 서버로부터의 응답에 본문이 제외된 점이 GET과의 차이점이다. POST는 클라이언트가 웹 서버에게 정보를 제공할 때 사용된다. URI는 웹 페이지를 액세스하기 원하는 클라이언트의 주소와 포트번호 그리고 패스를 포함하고 있다. 패스는 클라이언트에서 웹 서버로 요청하고 있는 웹 파일의 디렉토리 경로명이다. 마지막으로 HTTP 버전은 현재의 HTTP번전을 나타낸다.

컨텐츠 추출과정은 클라이언트의 요청 패킷 중 HTTP 요청 패킷을 처리하기 위한 과정으로 동작방식은 다음과 같다.

① sk_buff 자료구조를 이용하여 TCP값과 UDP값을 추출하여 프로토콜 변수에 값을 할당.
② TCP 서비스이면서 80번 포트로 입력되는 패킷을 추출(HTTP request만 처리).
③ 데이터가 있을 경우 URI에서 파일 이름을 추출.
④ 데이터 옵셋(offset)을 옮긴 다음 URI를 filename 배열에 저장.

차별화 서비스를 제공하기 위한 컨텐츠 분류과정은 서버 기반의 분류방법과 클라이언트 기반의 분류방법으로 구분할 수 있다. 서버 기반의 컨텐츠 분류방법은 서버내의 컨텐츠의 종류에 따라 차별화 서비스를 제공하는 방법으로 URL, 파일이름, 디렉토리에 따라 각기 다른 차별화 서비스를 제공하는 방식이다. 그리고 클라이언트 기반 분류방법은 접속하는 클라이언트의 IP 주소에 따라 차별화 서비스를 제공하는 방식이다. 본 논문에서의 컨텐츠 분류과정은 서버 기반의 컨텐츠 분류방법을 이용한다. 클라이언트에서 요청한 URL을 분석하여 파일 이름을 추출하고, 추출된 파일이름과 미리 정의된 파일이름별 클래스에 대한 자료구조를 비교하여 클라이언트의 요청정보를 클래스별로 분류한다.

① 클라이언트에서 요청한 파일이름을 클래스별로 분류.
② 파일에 대한 우선순위 정보가 있는 자료구조와 파일 이름을 추출하여 분류한 클래스 정보와 비교.
③ 동일 정보가 있다면 파일 이름에 대한 우선순위 리턴.
④ 동일 정보가 없다면 가장 낮은 우선순위설정.
⑤ 우선순위가 설정된 변수는 스케줄링과 패킷을 전송한 후 연결개수 데이터를 갱신.

3.3 DLC 알고리즘

부하제어기에서 사용되는 스케줄링 알고리즘에는 RR(round robin), WRR(weight round

robin), LL(least load), LC(least connect- ion) 등이 있다[13]. LC알고리즘에서는 클라이언트로부터 패킷이 입력되면 각 실행서버의 전체 연결 개수에 따라 연결 개수가 가장 적은 실행 서버로 패킷을 전송하는 알고리즘이다. 본 논문의 DLC 알고리즘은 실행서버의 전체 연결 개수 중 가장 작은 연결 개수를 선택하는 것이 아니라, 각 실행서버내의 클래스별 연결수를 비교하여 실행서버내의 클래스별 연결 개수가 가장 작은 실행서버로 클라이언트 요청을 전송하는 방식이다. DLC 스케줄링 알고리즘은 순차검색을 통하여 각 실행서버들의 최소 연결 개수를 구하는 방식으로 이루어진다. 부하제어기에서 각 실행서버로 부하를 분산시키기 위한 DLC 알고리즘의 처리과정은 다음과 같다.

① 분류된 클래스 정보를 연결 리스트(linked list) 형태로 입력받아 리스트 헤드부터 검색하면서 실행서버의 전체 연결 개수를 계산.
② 분류된 클래스에 대해서 최상위 클래스로부터 현재 클래스까지의 실행 서버별 연결 개수를 계산.(분류에서 제외된 클래스에 대해서도 같은 계산을 수행)
③ 각 실행 서버에 대해서 전체 연결개수에 대한 비율을 계산하여 스케줄링 수행.
④ 가장 적은 연결 개수를 가진 실행 서버 리턴.

DLC 알고리즘을 적용하는 본 논문의 부하제어기에서는 클라이언트에서 요청한 파일에 대한 우선순위를 처리하고, 각 실행 서버에 클래스별로 부하를 분산하기 위해 그에 따른 별도의 연결 개수에 대한 자료구조의 정의가 필요하다. 본 논문에서는 DLC 알고리즘에 적합한 자료구조를 다음과 같이 새로 정의한다.

○ 클라이언트 요청패킷에 대한 클래스정보 정의

클라이언트 요청 패킷은 부하제어기에서 ip_vs_conn이라는 자료 구조체로 저장된다. <표 3>은 클라이언트 요청 패킷에 대한 우선순위를 처리하는 부분코드로 순차 검색을 통하여 각 실행서버로의 부하 분산산을 위해 연결 리스트 헤더, 참조 개수, 남은 시간 등을 정의하고, 나머지 하부 항목들은 IP 터널링을 위한 항목들이다.

<표 3> 입력 패킷에 대한 클래스 정보

```
struct ip_vs_conn {
        struct list_head  c_list //linked list의 헤더
        atomic_t          refcnt  //레퍼런스 개수
        struct timer_list  timer;  //남은 시간

        __u32             caddr;  //클라이언트 주소
        __u32             vaddr;  //가상 주소
        __u32             daddr;  //실행 서버주소
        __u16             cport   //클라이언트 포트
        __u16             vport;  //가상 포트
        __u16             dport;   //실행 서버주소
        __u16             protocol;  //프로토콜
        .....
        int (*packet_xmit)(struct sk_buff *skb,
                           struct ip_vs_conn *cp);
        //패킷을 전송할 때 사용되는 함수
        int class_level; //입력 패킷을 분류한 클래스
        .....
};
```

○ 실행 서버 당 연결 개수를 저장하는 구조체

<표 4>는 실행 서버 당 연결 개수를 저장하는 구조체로서, 부하제어기가 각 실행 서버로의 부하 분산을 수행하기 위해 서버의 IP 주소, 포트번호 등의 정보를 포함하고 있다. 실행 서버내의 연결 개수는 서버와의 연결이 성립될 때와 해제될 때 증가, 감소되며 연결이 성립되는 시기는 최초 SYNC 패킷이 입력될 때이며, 연결의 해제는 FIN 패킷이 입력되거나 타임아웃 되었을 때 이루어진다.

<표 4> 서버 당 연결 수 저장

```
struct ip_vs_dest {
   struct list_head       n_list;  //linked list 헤더
   struct list_head       d_list;  //linked list 헤더
                          .....
   atomic_t activeconns;    //현재 활동 전체 연결수
   atomic_t inactconns;     //현재 비활동 전체 연결수

   //실행 서버내의 클래스 별 연결수
   struct __class_connection *count[class_num];
          .....
};
```

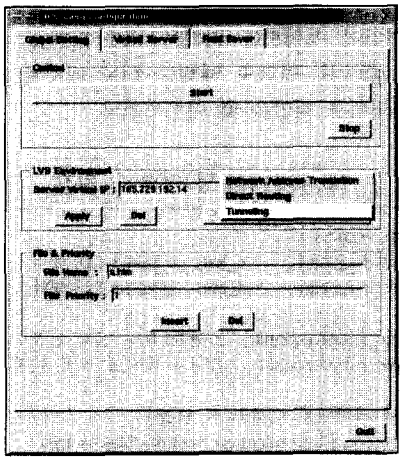
본 논문에서는 실행서버 내의 웹 문서에 우선순위를 부여하여 우선순위에 따라 클라이언트의 요청을 처리하기 때문에, 각 실행 서버내의 클래스별 연결 개수에 대한 정보도 가지고 있어야 한다. 따라서 '__class_connection' 자료구조를 ip_vs_dest 구조체내에 포함시킴으로써 DLC 스케줄링 시 실행서버 내의 클래스별 스케줄링이 가능하도록 한다.

Ⅳ. 구현 및 실험

본 논문에서 제시한 차별화된 웹 서비스 시스템은 Pentium-III 800MHz, 256MB 사양의 PC와 Linux Kernel 2.4.7 운영체제 상에서 구현되었으며, 시스템의 동작을 실험하기 위해서 클라이언트 3대, 부하 제어기 1대, 서버 2대, 모니터링 서버 1대를 네트워크로 연결한 실험환경을 구성하였다. 웹 서버는 Apache Web Server 2.4.17을 개조하여 이용하고 있으며, Linux 커널에는 Montavista 실시간 스케줄러를 접목하였다.

4.1 부하제어기 인터페이스

부하제어기는 일반 설정(global setting), 가상 서버 설정, 실행서버 설정 부분으로 구성되며 GUI(Graphic User Interface) 형태의 설정을 위하여 Python과 Tkinter모듈[18]을 이용하여 구현한다. [그림 8]에서 본 논문의 부하제어기 인터페이스를 보인다.

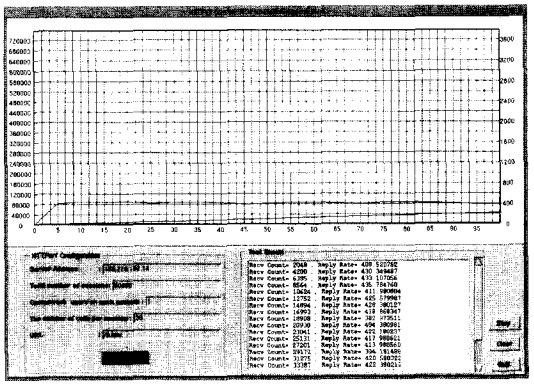


[그림 8] 부하제어기 인터페이스

일반 설정에서는 실행서버의 가상주소를 설정하고 클래스별 파일이름과 우선순위를 설정한다. 가상서버 설정 부분에서는 가상서버의 이름, 사용 어플리케이션, 프로토콜, 포트번호, 서버 주소, 네트워크 카드, 사용 알고리즘 등을 설정할 수 있다. 웹 서비스의 품질 보장을 위하여 사용 어플리케이션은 HTTP, 프로토콜은 TCP, 포트번호는 80번으로 설정하며, 사용 알고리즘은 DLC로 설정한다. 실행 서버 설정 부분은 실제로 웹 서비스를 수행하는 웹 서버와 각 서버의 가중치를 추가하는 부분이다. 본 논문의 실험을 위하여 사용한 웹 서버는 2대이며 실제 주소는 165.229.192.19, 165.229.192.20을 사용한다.

4.2 클라이언트 인터페이스

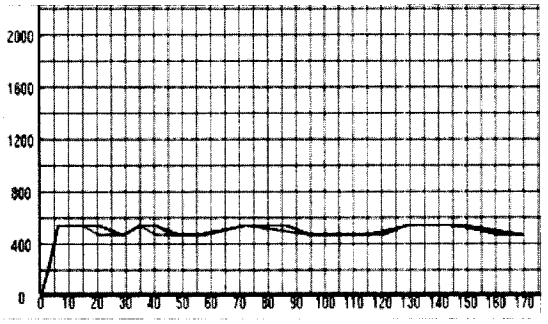
클라이언트 인터페이스는 사용자가 특정 웹 서버의 웹 페이지에 시간 단위와 전송량을 설정하여 요청을 보냄으로써 웹 서버의 수행 능력을 테스트할 수 있는 모듈이다. 각 클라이언트에는 이러한 모듈이 있으며, 이를 이용하여 각 클라이언트는 부하제어기를 통하여 실행서버에 요청을 보낸다. 본 논문에서는 클라이언트 요청에 대한 서버의 응답률을 GUI형태로 제공하기 위하여 각 클라이언트와 모니터링 에이전트 부분에 클라이언트 인터페이스 모듈을 설치한다. [그림 9]에서 본 논문의 클라이언트 인터페이스 화면을 보이고 있는데, 왼쪽 하단의 창은 테스트 환경을 설정할 수 있는 부분으로서, 'Server Address'는 요청을 보내고자 하는 가상 서버 주소를, 'Total number of sessions'은 연결하는 총 연결 개수를 나타낸다. 'Concurrent users on each session'은 한번에 동시 접속할 수 있는 접속자 수를 나타내며, 'The number of calls per session'은 한 연결 당 요청하는 세션 수를 나타낸다. 오른쪽 하단은 시간 단위로 들어오는 응답 카운트와 응답 비율을 실시간 텍스트 형태로 표현해주는 Test Result창이 있다. 최 상단의 그래프 창은 응답 비율과 응답 카운트의 데이터를 좌표로 변환하여 시간단위의 그래프를 화면상에 보여준다. 왼쪽 단위는 응답 카운트의 단위며, 오른쪽은 응답 비율의 단위를 나타낸다.



[그림 9] 클라이언트 인터페이스

4.3 실험

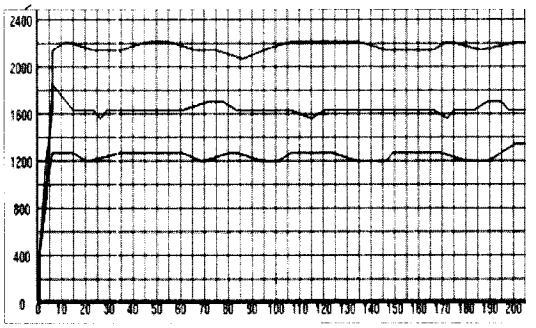
본 논문에서 수행한 실험은 서버에 과부하가 발생하지 않을 경우와 서버에 과부하가 발생했을 경우 그리고 서버의 과부하 시 클래스별 요청이 중단되었을 경우로 나누어서 수행한다. 실험 1은 서버에 과부하가 발생하지 않았을 경우를 위한 실험이다. 가상 IP는 165.229.192.14이고 전체 연결 개수는 50000, 동시 접속자는 1명, 한 연결 당 요청 개수는 50으로 실험한다.



[그림 10] 서버에 과부하가 발생하지 않을 때

[그림 10]에서 세 개의 클라이언트로부터 결과값을 모아놓은 모니터링 에이전트의 실행화면이다. 그림에 보이는 것처럼 부하가 적기 때문에 세 개의 그래프가 거의 비슷하다. 이와 같은 상황에서는 세 개의 클래스가 모두 서비스를 완벽하게 잘 받고 있기 때문에 차별화된 서비스가 필요하지 않은 상황이다. 즉, 웹 서버가 과부하 상태가 아니라면 모든 클래스의 요청이 제대로 처리가 되기 때문에 서비스를 하는 데는 문제가 되지 않는다. 그러나 웹 서버에 과부하 상태가 된다면 높은 우선순위를 가진 요청도 서비스를 받지 못하는 상황이 생길 수 있다. 따라서 과부하 상태인 서버의 작업을 부하 분산 시켜줄 필요가 있다.

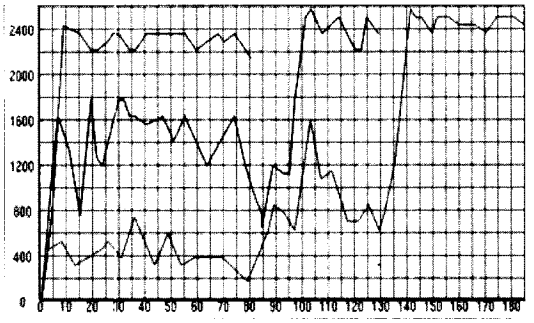
다음은 서버에 과부하가 발생한 경우를 위한 실험이다. 가상 IP는 165.229.192.14이고 전체 연결 개수는 50000, 동시 접속자는 30명, 한 연결 당 요청 개수는 50으로 실험한다. [그림 11]에서 가장 상단선이 클라이언트 2가 요구한 a.html의 응답률이고, 가운데 선이 클라이언트 3이 요구한 b.html의 응답률이며 그리고 제일 하단에 있는 선이 클라이언트 1이 요구한 c.html의 응답률의 결과 값을 보여주고 있다.



[그림 11] 서버에 과부하가 발생할 때

컨텐츠에 대한 우선순위 설정에서 a.html을 가장 높은 우선순위로 그 다음 b.html, c.html 순으로 우선순위를 설정해 두고 각 클라이언트에서 응답률을 테스트 한 결과 가장 우선순위가 높은 컨텐츠를 요청한 클라이언트 2가 응답률이 가장 높고 가장 우선순위가 낮은 컨텐츠를 요청한 클라이언트 1의 응답률이 가장 낮음을 볼 수 있다.

마지막으로, 서버에 과부하가 발생하고 클래스별 요청이 중단 되었을 경우를 위한 실험이다. 가상 IP는 165.229.192.14이고 전체 연결 개수는 50000, 동시 접속자는 30명, 한 연결 당 요청 개수는 50으로 실험한다.



[그림 12] 클래스별 요청이 중단되었을 때

[그림 12]를 보면 모든 클래스의 요구가 지속적으로 발생하다가 80초의 시간이 흐른 뒤 a.html을 요구하는 클라이언트 2의 요구가 중단 되었을 때 b.html과 c.html을 요구한 클라이언트의 응답률이 증가하는 것을 볼 수 있다. [그림 12]에서와 같이 a.html 파일에 대한 요청이 중단되었기 때문에 b.html의 요구가 가장 우선순위가 높다. 이때 130초의 시간이 흐른 뒤 b.html을 요구한 클라이언트 3의 요구가 중단 되었을 때 다시 c.html의 응답률이 증가하는 것을 그래프로 볼 수 있다.

Ⅵ. 결 론

본 논문에서는 정보의 중요도나 정보 사용자의 중요도에 따라 서비스를 제공할 수 있는 차별화된 웹 서비스 시스템의 구현을 위한 두 가지 접근 방법, 즉, 커널 수준 접근 방법과 부하 분산 접근 방법을 제시하였다. 커널 수준 접근방법에서는 커널 내부에 실시간 스케줄링 프로세서를 접목하였으며, 부하 분산 접근방법에서는 IP 수준의 가장법과 터널링 기법을 이용한 부하제어기를 구현하였다. 본 논문의 부하제어기는 기존의 LC 알고리즘을 개선한 DLC 알고리즘을 제시하여 서비스 요청의 우선순위에 따라 차별화 된 웹 서비스를 제공할 수 있도록 하였다. 본 논문에서 구현한 부하 분산 시스템의 동작을 세 가지 경우에 대해서 실험하였으며, 이 실험에서 차별된 웹 서비스를 지원하고 있다는 것을 보였다. 본 논문의 DLC 알고리즘은 LC 알고리즘과 마찬가지로 정적으로 동작됨으로 각 실행 서버의 동적인 부하변화를 반영할 수는 없다. 따라서 각 실행 서버의 CPU 모니터링 및 서버 상태 분석 등을 통한 서버 부하 정도를 고려한 동적인 분산 서비스 시스템에 대한 연구는 계속 진행 중이다.

참 고 문 헌

[1] R.Fielding, J. Getys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hyper text Transfer Protocol - HTTP/1.1," IETF, January 1997.

[2] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User Perceived Quality into Web Server Design", Proc. Of the 9th International World Wide Web Conference, Amsterdam, Netherlands, May 2000, pp.92-115.

[3] N. Vasiliou and H. Lutfiyya, "Providing a Differentiated Quality of Service in a World Wide Web Server", Proc. Of the Performance and Architecture of Web Servers Workshop, Santa Clara, California USA, June 2000, pp. 14-20.

[4] Apache Group, http://www.apache.org/.

[5] R. Bhatti and R. Friedrich., "Web Server Support for Tiered Services.", IEEE Network, September 1999, pp.64-71.

[6] Chad Yoshikawa, et al., "Using Smart Clients to Build Scalable Services", USENIX'97, 1997, http://now.cs.berkeley.edu/.

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

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

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

[10] Edward Walker, "pWEB : A Para-llel Web Server Harness", April, 1997.

[11] Daniel Andresen, et al., "Towards a Scalable Distributed WWW Ser- ver on Workstation Clusters", Proc. of 10th IEEE Intl. Symp. Of Parallel Processing (IPPS'96), pp. 850-856, April 1996.

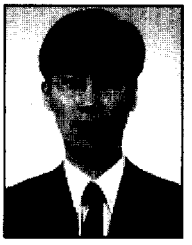
[12] Eric Anderson, Dave Patterson, and Eric Brewer, "The magicroute Magicrouter : an Application of Fast Packet Interposing", http://www.cs.berkeley.edu/~eanders/ma-gicrouter/, May 1996.

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

[14] Cisco System, "Cisco Local Director",http://www.cisco,com/warp/public/751/ lodir/index.html. 1998.

[15] Montavista Software, http:// www. montavista.com
[16] Barden, R., Clark, D. and Shen- ker, "Integrated Services in the Internet Architecture: an Overview", Internet RFC1633, June, 1994
[17] S.Blake et al., "An Architecture for Differentiated Services", RFC-2475. December 1998.
[18] Python, http://www.python.org

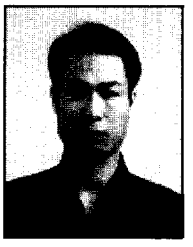
이 명 섭(Myung-Sub Lee) 정회원



e-mail:skydream@cse.yu.ac.kr
1998년 2월:경일대학교 컴퓨터공학 졸업 (공학사)
2000년 2월:영남대학교 대학원 컴퓨터공학과 졸업(공학석사)
2002년 8월:영남대학교대학원 컴퓨터공학과 수료(박사수료)

<주관심분야> 망관리 시스템, 데이타 마인닝, 에이전트, QoS(Quality of Service)

박 창 현(Chang-Hyeon Park) 정회원



e-mil: park@cse.yu.ac.kr
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
1993-현재: 영남대학교 컴퓨터공학과 부교수

<주관심분야> 인공지능, 데이타 마인닝, 에이전트, 지능형 망관리 시스템