동적 가중치에 기반을 둔 LVS 클러스터 시스템의 부하 분산에 관한 연구

김 석 찬[†] · 이 영[††]

요 약

본 연구에서는 리눅스 가상 서버(LVS : Linux Virtual Server) 클러스터 시스템에서 실제 서버의 상태에 기초한 사용자의 요청을 분배하는 방법론을 연구하고자 한다. LVS 클러스터 시스템에서 사용자의 요청을 분배하는데 적용되는 기존 WLC(Weighted Least Connection) 방법론이 검토되었고, 실제 서버의 부하를 고려하여 각 서버에 요청을 할당하는 부하 분산 방법론을 제안하고자 한다. 부하 측정을 위한 실험은 가상의 부하를 생성하는 툴을 사용하여 서버에 부하를 부과하여 실행되었다. 본 연구에서 제시된 부하 분산 방법론이 기존의 WLC 방법론보다 실제 서버의 메모리 사용측면에서 효율을 기대할 수 있어 제안하고자 하며, 또한 서버 자원을 균형적으로 분배시키고 가중치의 변화에 대한 교정력(correction potentiality)이 어느 정도 개선됨을 확인할 수 있었다.

A study of distributing the load of the LVS clustering system based on the dynamic weight

Sukchan Kim[†] · Young Rhee[††]

ABSTRACT

In this paper, we study the methodology of distributing the requests of clients based on the state of real server in the LVS(Linux virtual server) clustering system. The WLC(weighted least connection) algorithm in the LVS cluster system is studied. The load distributing algorithm which assigns a weight into each real server is devised by considering the load of real servers. The load test is executed to estimate the load of real servers using a load generating tool. Throughout the result of the experiment, we suggest new load distributing algorithm based on the usage of physical memory of each real server. It is shown that the correction potentiality of new algorithm is somewhat better than that of the WLC algorithm.

키워드 : 클러스터 시스템(clustering system), 최소 연결 가중치(weighted least connection), 실제 서버(real server)

1. 서 론

인터넷의 대중화 추세로 일상 생활에서 인터넷이 차지하는 비중이 점차 높아지고 있으며, 인터넷 사용자 수와 인터넷 트래픽(traffic)이 급격히 증가하고 있다. 물론 이에 따른 호스트 또는 서버수 또한 급격히 증가해 1999년 8월 당시 460,974개로서, 전년대비 158%의 성장률을 보이고 있으며, 하드웨어 및 소프트웨어 기술의 발달에 따른 서버의 초당 처리능력의 향상으로는 인터넷 사용자 수와 트래픽의 급격한 증가폭을 모두 감당하기에는 조만 간에 부족할 것으로 예상된다. 2000년 8월 미디어-메트릭스(Media-Metrix) 사의 통계 발표에 의하면 접속이 빈번한 인터넷 사이트의 경우, 일일 200~270만 명이 접속하며 평균적인 접속시간이 약 30분 정도로, 동시접속자수로 환산하면 5만 명 정도라고 한다.[15] 이는 우리나라 전체 인터넷 사용 인구의 90%가 동시에 발생시키는 트래픽량에 해당하는 것으로, 서버에 엄청난 과부하로 인하여 단일 서버로 구성된 시스템이 감당하기에는 역부족임을 쉽게 예측할 수 있다. 예를 들어 접속이 빈번한 전자상거래(Electronic Commerce)사이트에서는 순간적으로 발생하는 과도한 트래픽에 의한 웹 서버의 부하는 QoS(Quality of Service)의 저하와 서비스 불능에까지 이를 수 있다. 최근의 그래픽 처리와 CGI(Common Gateway Interface) 및 관련 응용프로그램의 사용이 증가하면서 웹 서버 시스템에 미치는 부하는 기하급수적으로 증가되어 웹 서버 시스템에 대한 구조적 측면에서 재고할 필요가 있다. 서버 시스템의 과부하 문제를 해결하기 위한 노력으로서, 시스템의 하드웨어적인 관점 또는 소프트웨어 및 네트워크-미디어적인 관점 등에서 많은 연구가 계속되고 있다. 특히 서버의 확장성 연구의 한 분야로서 서버 시스템에 대한 부하 분산 방법에 관한 연구가 활발히 진행되고

† 준 회 원 : 계명대학교 대학원 산업공학과
†† 정 회 원 : 계명대학교 산업공학과 교수
논문접수 : 2001년 8월 13일, 심사완료 : 2001년 12월 24일

있는데, 현재까지는 클러스터링(clustering) 기술이 가장 효과적인 대안으로 부상하고 있다.

클러스터 시스템은 동종 또는 이기종 시스템을 서로 통신이 가능하도록 연결된 소프트웨어적으로 시스템을 확장하는 방법으로 클러스터내의 개별 시스템을 서버 또는 노드(node)라고 한다. 클러스터 시스템의 특징은 가용도(availability) 향상과 더불어 성능(performance)을 보장하는 서버 군(群)을 구성할 수 있다는 점이다. 기존의 단일 또는 다중 프로세서 시스템에 비해 거의 제한 없이 확장성(scalability)을 보장받을 수 있고, 비용 대 효율 면에서도 우수하며, 높은 신뢰성을 가지는 것도 장점이다.[14, 17] 클러스터 시스템은 그 사용 목적에 따라, 순수한 과학 계산을 위한 시스템과 부하분산(load balancing)을 목적으로 하는 시스템으로 구분될 수 있다.

Beowulf 시스템은 주로 빠른 계산을 목적으로 하는 과학기술용 클러스터 시스템이다. 노드의 수와 노드의 개별적인 성능 및 네트워크의 속도에 따라 슈퍼컴퓨터(super computer)에 버금가는 성능을 낼 수 있는 시스템이다. Beowulf 시스템은 NASA의 연구 프로젝트 수행과정에서 개발된 부산물로서, 특정 작업이 노드의 수만큼 분할되어 각 노드에 할당되며, 각 노드에서 처리된 개별 계산 결과는 마스터 노드에서 종합되어 최종적인 결과로 도출된다. Beowulf 시스템은 기존 슈퍼컴퓨터 보다 가격 대 성능비가 우수하다는 장점을 지니고 있어 과학 기술 및 오락 분야 등에서 골고루 응용되고 있다. 그러나 병렬 프로그래밍의 복잡성으로 인해 적용하기 어려운 분야가 있다는 점이 단점으로 지적되고 있다.[14]

부하 분산 클러스터 시스템은 부하를 클러스터 내의 다른 노드로 분산시키는 조정자 역할을 하는 load balancer 또는 director와, 실제 서비스를 수행하고 요청에 대해 반응하는 노드(real 서버)로 구별되며, 시스템의 구성에 따라서는 load balancer가 복수일 수도 있다. 부하 분산 클러스터 시스템에서 director는 서비스 요청이 있을 경우, 특정한 알고리즘을 바탕으로 적절한 노드를 선발하여 그 요청을 처리하게 한다. 즉, 하나의 작업을 여러 대의 노드가 나누어 처리하는 것이 아니고, 분산 알고리즘에 의해 선택된 노드가 배정받은 작업 전체를 처리한다. 그러므로 병렬 시스템과는 다르게 쇄도하는 모든 요청이 여러 대의 노드에게 시간 간격을 두고 적절히 안배되어 부하가 분산되어지는 것이다.[17]

본 연구에서는 인터넷상에서 실행되는 모든 서비스, 예를 들어 WWW, ftp, telnet 등에 적용할 수 있는 부하 분산 기능이 있는 웹 클러스터 시스템을 분석하고자 한다. 웹 클러스터 시스템은 대부분이 Linux, Unix 및 NT 계열의 NOS(Network Operating System)에서 구현되고 있다. 특히 본 연구에서는 Zhang, Jin, Wu[14] 등에 의해 연구된 Linux kernel 기반의 LVS 클러스터 시스템에 기초한 시스템을 구체적으로 대상으로 정하였다. LVS 클러스터 시스템 역시 네트워크로 연결된 클러스터 시스템 내에 하나의 director서버와 다수의 real 서버가 존재하며, 장애극복(fail-over)을 위해 다수의 director들이 존재하는 시스템도 있다. LVS 클러스터 시스템에서는 각 real 서버의 접속수에 근거하여 작업을 할당하고 있다. 접속수만을 근거하여 작업을 할당하는 경우, 실제 real 서버에 대한 부하 분산 효과가 효과적으로 실현될 수 있을지는 의문이다. 따라서 직접적으로 부하의 불균형으로 인한 LVS 클러스터 시스템 내의 일부 real 서버에서 심각한 자원의 편중현상을 초래할 수 있으며, 효율성이 저하되는 문제가 발생될 수 있기 때문이다. 본 연구에서는 클러스터 시스템을 구성하는 각 real 서버가 실제로 받고 있는 시스템 부하에 기초하여, 각 노드의 가중치(加重値)를 주기적으로 산정함으로써 클러스터 시스템 전체의 효율을 제고하는데 목적을 둔다. 본 연구에서는 각 real 서버의 자원 소모량에 기초한 몇 가지 가중치 변경 알고리즘을 제시하고, 실제 실험을 통하여 이러한 알고리즘이 LVS 클러스터 시스템 내에서 부하 분산에 기여하는 효과를 분석하고자 한다.

본 연구의 구성은 2장에서 클러스터링 방법론에 관한 관련 연구를 소개하며, 3장에서 동적 가중치를 고려한 부하 분산 방법론이 언급된다. 4장에서는 네트워크 시뮬레이션을 실행하기 위한 제반 사항이 제시되며, 5장에서는 제시된 부하 분산 방법에 의한 네트워크 시뮬레이션의 결과와 분석이 자세히 언급된다. 마지막으로 6장에서는 보다 효과적인 부하 분산 방법론을 제시하는 것으로 결론을 맺는다.

2. 관련 연구

클러스터 시스템에서 작업 할당하여 처리하는 방식에 관해서는 다양한 연구 결과가 제시되어 있으며, 그 중 일부는 이미 상업화되어 실제로 활용되고 있다. 그리고 이와는 별도로 소극적인 의미의 부하 분산 개념이 포함된 미러링(mirroring) 방식이 실제 웹 상에서 구현된 바 있으며, 현재도 다수의 웹 사이트와 ftp 사이트가 이 방식을 채택하고 있다. 미러링 서버는 구축이 비교적 용이하지만, 컨텐츠 동기화를 위해 사용되는 NFS(Network File System) 자체에 의해서도 추가적인 부하가 발생할 수 있다. 미러링 방식은 똑같은 컨텐츠를 각기 다른 서버에 저장하고, 이를 물리적으로 떨어진 곳에 위치시켜 서비스를 원하는 클라이언트가 자신에게 가장 가까운 곳에 위치한 서버를 임의로 선택할 수 있도록 하는 방법이다. 이 방식으로는 서버의 부하를 동적으로 분산시킬 수 없기 때문에 클라이언트의 지역적 편중이 심할 경우, 이와 더불어 특정 서버에 부하가 편중되는 현상을 초래할 수 있다.

2.1 Round Robin DNS

Round Robin DNS(Domain name service)는 지역 DNS 서버를 설치하고, 서비스를 담당하는 도메인명으로 연결요청이 올 경우, 특정 IP 주소 집합 내에 있는 IP 주소로 할당시키는 방법이다. 비교적 간단한 방법으로 접속을 분산시킬 수는 있지만 몇 가지 문제점을 내포하고 있다. 특정 클라이언트에 지리적으로 가깝거나, 가장 부하가 적은 서버로 접속을 분산시키는 기능이 없고, HTTP(Hyper Text Transfer Pro-

tocol) 연결지속(persistent connection)의 문제가 있다. 예를 들어 단일 연결(session)에 여러 개의 트랜잭션이 발생하는 특정 CGI 응용프로그램의 경우 TTL(Time to Live)을 초과하면 다음 트랜잭션이 다른 서버로 연결되므로, 일관된 작업이 일어나지 않고 에러를 유발할 수 있다는 것이다. 일반적으로 Round Robin DNS 방식은 읽기 전용의 컨텐츠에 사용하는 것이 적절하다고 알려져 있다. 또한 기존에 접속한 경험이 있고, DNS cache가 남아 있거나, 프락시(proxy) cache 서버를 사용할 경우에는 특정 서버로만 접속이 편중되는 경향도 있다.[1, 9]

2.2 멀티캐스팅의 응용

멀티캐스팅(multicasting) 방법은 인터넷 상에서 데이터의 기본 단위인 패킷을 멀티캐스팅으로 전송하는 방식이다. 따라서 멀티캐스팅 네트워크는 중간 노드인 라우터가 멀티캐스팅 가능한 주소로 지정되어야 하며, 서비스를 하는 라우터는 특정 URL 클래스(class D)에서 오는 요청만을 처리하도록 하는 네트워크 기법이다. 브로드캐스팅(broadcasting) 보다는 대역폭을 절약할 수 있다는 장점이 있지만, 서버군과 클라이언트들 사이의 라우터가 멀티캐스팅 기능을 제공하지 못할 수도 있다는 것이 단점으로 지적되고 있다. Roland J. Schemers III는 하나의 DNS 이름에 대하여 실제 서비스를 하는 각각의 서버를 동적으로 바꾸어주는 perl daemon을 작성하였는데, 이때 부하가 최소인 서버가 가장 우선적으로 선택된다.[9]

2.3 LVS 클러스터 시스템

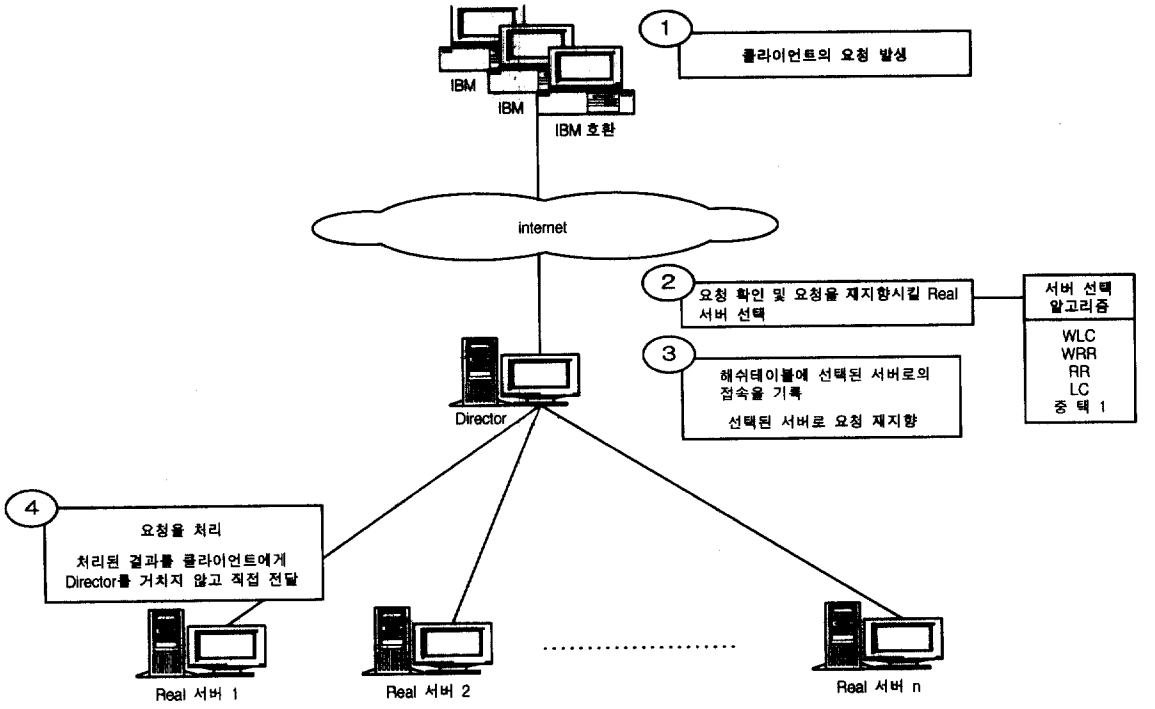
LVS 클러스터 시스템은 클라이언트들의 요청이 쇄도하고 mission critical 사이트인 것을 전제로 적용하고 있다. 이 경우 시스템에 접속하는 요청들이 서버에 미치는 영향은 서로 다를 수 있는데, 예를 들어 어떤 클라이언트는 사이트에 접속한 후 접속을 휴지(sleep) 상태로 유지할 수도 있고, 다른 클라이언트는 사이트가 제공하는 여러 가지 검색 서비스와 각종 CGI 응용프로그램을 실행시켜 하위 또는 아들 프로세스(child process)를 생성하여 서버에 부하를 추가할 수도 있다. 또 다른 경우는 다른 클라이언트들 보다 상대적으로 속도가 느려서 데이터를 처리하는 동안 패킷의 TTL(Time to Live)이 종료되어 요청에 대한 재전송을 요구하는 경우가 발생할 수도 있다. LVS 클러스터 시스템은 각 real 서버가 처리 중에 있는 접속수에 근거하여 작업 할당을 결정하기 때문에 real 서버의 자원인 메모리와 CPU time 등의 소요에 있어서 불균형을 가져올 소지가 있다. 클라이언트로부터의 요청이 발생시키는 부하의 크기가 거의 동일한 경우에는 상관없지만, 일반적으로 real 서버에 접속한 클라이언트들이 서버의 자원을 소모하는 정도는 일정하지 않기 때문에 각 real 서버별 자원 소모 정도의 차이는 더욱 심화될 가능성이 내포되어 있다. 더욱이 클라이언트의 속도와 네트워크의 전송 속도까지 고려한다면, 상대적으로 속도가 느린 클라이언트와 네트워크의 영향으로 real 서버가 재전송해야 하는 경우가 발생하게 되어 real 서버에 미치는 부하의 편중은 더욱 심화될 수 있다. 이와 같은 점을 고려할 때 LVS 클러스터 시스템은 실제 서비스를 하는 real 서버 자체에 대한 부하 분산에 적극적으로 대처하는 시스템이라고 볼 수 없다.

3. 동적 가중치를 고려한 부하 분산 시스템

LVS 클러스터 시스템의 구성을 위해서는 클러스터 시스템의 논리적인 topology와, 클라이언트의 요청을 각 real 서버로 할당하는 알고리즘이 결정되어야 한다. 현재까지는 LVS 클러스터 시스템의 구성을 위해 3가지 형태의 topology가 사용되고 있고, 특정 topology 하에서 사용될 수 있는 작업할당 알고리즘으로 4가지 방식이 존재한다. LVS 클러스터 시스템의 클러스터링 방식은 NAT(Network Address Translation) 방식, IP-Tunneling 방식, 그리고 Direct routing 방식이 있으며, 요청을 각 real 서버로 작업을 할당하는 알고리즘으로는 Round Robin, Weighted Round Robin, Least Connection, Weighted Least Connection 방식이 있다.[14, 17]

LVS 클러스터 시스템은 논리적인 topology 형태에 따라 다소 차이가 있으나 공통적으로 LVS 클러스터 시스템에서 서비스를 제공해 줄 가상 IP 주소, director와 real 서버가 가져야 할 IP 주소를 필요로 하며, 전체적인 서비스는 가상 IP 주소를 통해 이루어진다. 가상주소는 클라이언트가 서비스를 요청할 때 요청 패킷이 보내지는 주소가 되며, 이 주소를 LVS 클러스터 시스템 내의 모든 노드 즉, director와 real 서버들이 공유하게 된다. 즉, director를 포함하여 모두 n+1개의 노드가 있는 클러스터 시스템에는 n+2개의 IP 주소가 필요하게 되며 (그림 1)에 나타나 있다.

IP tunneling 방식의 LVS 클러스터 시스템은 IP datagram 안에 IP datagram을 포함하는 방법으로 IP encapsulation이라고 한다. 가상주소를 향하는 datagram을 다른 IP 주소(real 서버의 실제 IP 주소)로 재지향시키는 것이다. IP encapsulation은 Mobile-IP 또는 IP-mobile, IP-Multicast 등에 적용되며 아마추어 라디오 등에서도 많이 활용된다. Mobile-IP는 노트북에서 휴대전화를 이용하여 인터넷과 같은 데이터 통신을 이용하는 호스트의 경우, 이 호스트가 IP 주소의 변화나 연결 중단 없이 인터넷의 한 지점에서 다른 지점으로 연결을 이동시키는 메커니즘이다. 호스트의 연결 지점이 변경되는 경우, IP 주소도 바꿔야만 한다. Mail 등이 수신될 경우 IP tunneling 기술을 이용하여 datagram을 현재 mobile 호스트가 사용하는 IP 주소로 datagram을 encapsulating 한 후에 재지향하면, 지리적으로 계속해서 이동하는 mobile 호스트에게도 올바르게 데이터를 전송할 수 있게된다.[16] LVS 클러스터 시스템에서도 클러스터 시스템 내의 모든 노드들이 공유하는 Virtual 서버의 IP 주소로 WWW 또는 ftp, telnet 등의 서비스를 요청하게 된다. 이 때 director만이 가상 IP 주소로 들어오는 요청에 ARP(Address Resolution Protocol) 응답을 하여 정해진 작업할당 방식에 따라 요청 패킷을 재지향할 real 서버를 선택하고, IP tunneling을 이용하여 패킷을



(그림 1) LVS 클러스터 시스템의 요청 처리(IP tunneling 방식)

재지향하게 된다. director는 해쉬 테이블(hash table)에 각 real 서버에 대한 접속을 기록한다. 재지향된 패킷은 real 서버에서 처리된 후, director를 경유하지 않고, 바로 요청 클라이언트로 전송된다.

본 연구에서는 기존 작업 할당 방식인 WLC 방법을 간략히 소개하고, 나아가 서버 상태에 의존한 동적 가중치를 고려한 부하 분산 방법을 제시하고자 한다.

3.1 기존 WLC 방식

기존 WLC 알고리즘에서는 director로 서비스 요청이 발생할 경우, 각 real 서버마다 부여된 가중치와 각 서버 당 접속수에 근거하여 작업을 할당한다. 초기 접속은 가중치가 큰 서버로 접속이 할당되고, 다음 접속부터는 식 (1)을 만족하는 특정 서버에 할당된다.

$$Min\left(\frac{\frac{C_i}{C_{all-conn}}}{W_i}\right) = Min\left(\frac{C_i}{W_i}\right) \quad (1)$$

$$where\ i = 1 \cdots n$$

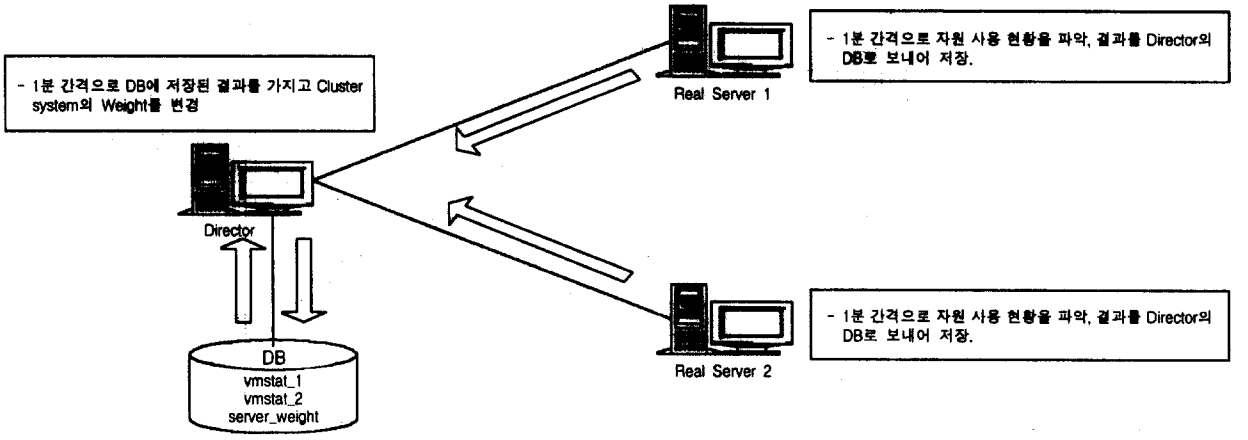
$where\ C_{all-conn} = LVS$ 클러스터 전체에 연결된 접속수
C_i = 현재 서버 i에 접속된 연결수
W_i = 서버 i의 $weight$ 값

식 (1)에서 보는 바와 같이, 서버 당 현재 접속수를 전체 클러스터 시스템에 연결된 총 접속수로 나눈 후, 다시 해당 서버의 가중치로 나누어진 값 중에서 가장 작은 값을 가지는 서버에 다음 접속을 할당한다. 여기에서 $C_{all-conn}$은 모든 서버에 대해 적용되는 같은 값을 가진 상수이므로, 서버 당 현재 접속수인 C_i를 서버 가중치 W_i로 나눈 값의 최소값으로 다음 접속을 할당할 서버를 선택해도 같은 결과를 얻는다.

3.2 동적 가중치를 고려한 부하 분산 방법 1

본 절에서는 각 real 서버의 자원 정보를 파악하고 이를 근거로 LVS 클러스터 시스템을 운용한다면 보다 효율적인 LVS 클러스터 시스템을 구축할 수 있을 것으로 판단되어 방법론을 제시하고자 한다. real 서버의 부하에 대한 고려가 없는 WLC 방법은 특정 real 서버로 부하가 편중될 수 있는 가능성이 있다는 것이다. 부하의 편중 현상을 근본적으로 차단할 수는 없겠으나, WLC 방식은 오직 접속수를 기준으로 서버를 선택하기 때문으로 추측할 수 있다. 클러스터 시스템의 성능은 시스템을 구성하는 모든 노드 중 가장 성능이 열등한 노드에 영향을 받기 때문에, 각 노드의 성능이 동일한 같은 사양의 서버 군(群)을 선택하여 LVS 클러스터 시스템을 구성하는 것이 보편 타당하다. 이러한 맥락에서 WLC 방법에서 real 서버들의 하드웨어적 성능을 나타내는 가중치는 같은 성능의 real 서버들로 구성되었으므로 가중치는 각각 1 또는 동일한 값으로 설정하고 있다. 부하 분산 방법 1은 real 서버의 부하를 주기적으로 측정함으로써 클러스터 시스템의 각 real 서버의 가중치를 주기적으로 변경시키는 방법이다. 가중치를 주기적으로 변경하도록 구축된 변형된 LVS 클러스터 시스템은 (그림 2)와 같다.

각 real 서버의 부하를 측정하기 위하여 일반적인 UNIX 계열(BSD, System V, Linux)의 vmstat를 사용하며, 적절한 값을 얻을 수 있도록 5초 간격으로 5회를 측정하고자 한다. [11] 최초의 측정은 부팅 후 데이터이므로 제외하고, 나머지



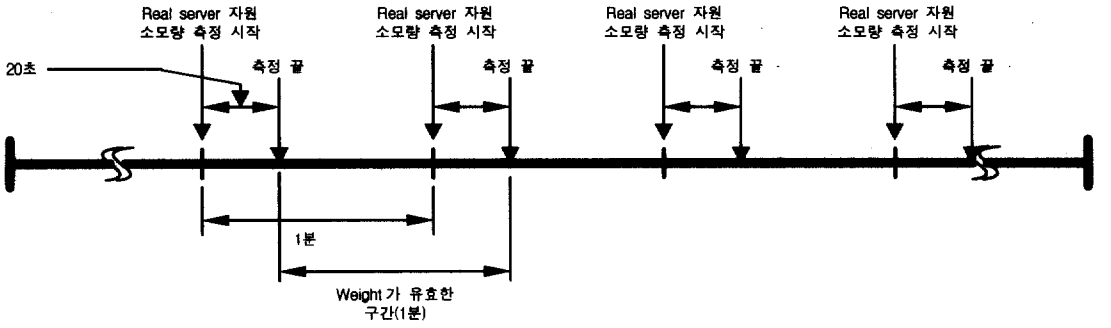
(그림 2) real 서버의 부하를 고려한 LVS 클러스터

4개의 데이터 세트를 사용하여 각 real 서버의 부하를 측정하여 평균값을 산출한 뒤, director의 DB에 저장한다. real 서버의 부하를 측정하기 시작한 시점부터 20초 동안의 성능 측정 결과로 다음 가중치가 바뀌기 전인 1분간 사용될 가중치를 결정하고자 한다. (그림 3)은 1분마다 가중치를 산정하기 위해 real 서버의 부하를 측정하는 과정을 나타낸 것이다. 가중치의 산정은 여러 개의 요청이 특정 프로세스로 연결된다고 할 때 가장 먼저 들어온 요청에 의해 프로세스가 생성되고, 이것이 처리된 후, 생성된 프로세스는 time out동안 다음 요청을 기다리므로 real 서버 내에 이러한 요청에 대기하는 휴지 상태의 프로세스들이 존재하며, 이들이 메모리를 점유하게 된다. 그러므로 각 real 서버의 사용 가능한 메모리의 크기에 따라서 2대의 서버 중 가용한 메모리의 크기가 작은 쪽에 가중치 1을 부여하고, 큰 쪽에게는 가용한 메모리들의 차를 평균 메모리로 나누어 얻어진 값의 반올림 값을 가중치로 정한다.

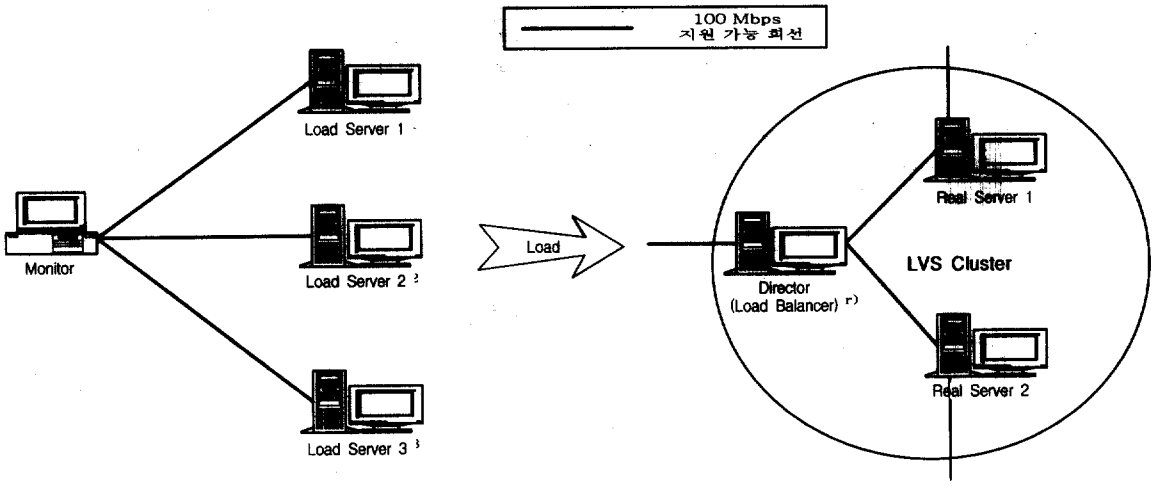
또한 (그림 4)는 LVS 클러스터 시스템의 director에서, 수집된 각 real 서버의 메모리의 소모량의 상태를 근거로 가중치를 산정하는 알고리즘이다. 가중치 산정의 기본적인 개념은 각 real 서버에서 측정된 사용 가능한 메모리 양의 차를 프로세스 당 평균 메모리로 나눈 값을 사용 가능한 메모리 양이 많은 real 서버에게 가중치로 부여하고, 적은 서버에게 1을 가중치로 부여한다.

```
push(@wei_a, swap($vs1[0], $vs1[1], $vs1[3], $vs1[5], $weights),
swap($vs2[0], $vs2[1], $vs2[3], $vs2[5], $weights)) ;
$min_mem = min_value($vs1[2], $vs2[2]) ;

if(($wei_a[0] == 0)&&($wei_a[1] == 0))
{
      $wei_a[0] = 1000000 ;
      $wei_a[1] = 1000000 ;
}

@c_mem = ([$wei_a[0], $vs1[2], $min_mem], [$wei_a[1], $vs2[2],
$min_mem]) ;
for ($i = 0 ; $i < 2 ; $i++)
{
      push(@weight, check_weight_mem($c_mem[$i][0], $c_mem
      [$i][1], $c_mem[$i][2])) ;
}

for ($i = 0 ; $i < 2 ; $i++)
{
      $weight[$i] = int($weight[$i] * 1000) ;
}
      $max_weit = max_value($weight[0], $weight[1]) ;

if ($max_weit > 65535)
{
      for ($i = 0 ; $i < 2 ; $i++)
      {
            $weight[$i] = int ($weight[$i] * dec_rate($max_weit)) ;
      }
}

system("ipvsadm -e -t 210.107.211.59 : 80 -r 210.107.211.52 -i -w
$weight[0]") ;
system("ipvsadm -e -t 210.107.211.59 : 80 -r 210.107.211.53 -i -w
$weight[1]") ;

$dblvs -> do("insert into server_weight values(null, null,'$weight
[0]','$weight[1]')") ;
```

(그림 4) 가중치 알고리즘



(그림 3) 부하의 sampling 프로세스



(그림 5) LVS 클러스터 시스템 및 load 서버 군(群)

$$weight_{\max} = \frac{\text{사용 가능한 } memory_{\max} - \text{사용 가능한 } memory_{\min}}{3,500\ KByte}$$

$$weight_{\min} = 1 \qquad (2)$$

식 (2)에서 사용한 3,500 KB는 실험 과정 중 각각의 real 서버의 HTTPD 프로세스와 perl 프로세스가 차지하는 메모리 양에 대한 평균치로 계상된 값이다.

3.3 동적 가중치를 고려한 부하 분산 방법 2

부하 분산 방법 2에서는 가중치 산정 방식을 수정하였는데 2대의 real 서버의 메모리 차이를 구한 후, 메모리 차이를 프로세스 당 평균 메모리 량인 3,500 KByte로 나눈 값을 real 서버간의 가중치 차이로 계산하였고, 분당 2,000개의 접속에 대하여 가중치의 차이로 나누어 고정된 값 K를 구한다. 그리고 2대의 real 서버 중 가용한 메모리가 큰 쪽에 'K+가중치차'를 가중치로 부여하고, 다른 real 서버에는 'K'를 가중치로 부여하는 방식을 사용하였다. 부하 분산 방법 1과 다른 점은 real 서버에 부여하는 가중치의 최소값이 1이 아닌 K로 설정하여, 부하분산의 반영도를 실험 1보다 민감도를 둔화시키고자 하는 의도이다. 이를 수식화하면 식 (3)과 같다.

$$weight_{diff} = \frac{\text{사용 가능한 } memory_{\max} - \text{사용 가능한 } memory_{\min}}{3,500 KByte}$$

$$K = \frac{2,000}{weight_{diff}}$$

$$weight_{\max} = K + weight_{diff}$$

$$weight_{\min} = K \qquad (3)$$

where $weight_{diff}$ = $weight$의 차(가중치의 차)

$weight_{\max}$ = 사용 가능한 $memory$가 큰 $real$서버의 $weight$

$weight_{\min}$ = 사용 가능한 $memory$가 작은 $real$서버의 $weight$

분당 접속수를 2,000으로 설정한 것은 load test에서 생성되는 약 4,000개의 접속 요청이 2대의 real 서버로 분산되기 때문이다.

4. 네트워크 시뮬레이션

4.1 클러스터링 시스템의 구성

본 연구에 사용된 시스템은 Wensong Zhang[17]의 LVS-Howto 및 관련문헌을 근거로 하여 구성하였다.[2-6, 8, 12-13] 실험에 바탕이 되는 기본적인 LVS 클러스터 시스템을 구성하기 위하여, 논리적인 topology로서 IP tunneling 방식을 적용하였고, 작업할당 알고리즘으로는 WLC 방법과 두가지의 부하 분산 방법을 채택하였다. 실험 환경은 일반적인 이더넷 환경 하에서 수행되었으며, 주로 심야를 이용하여 실험을 수행하였는데 이는 시스템에 영향을 줄 수 있는 외부간섭을 최대한 배제하기 위함이다. 실험에 사용된 장비들은 1대의 director와 2대의 real 서버, 그리고 LVS 클러스터 시스템에 부하를 부과할 load 서버 3대와 이를 모니터할 시스템 1대로 환경을 구성하였으며, 자세한 구성 사양은 생략하기로 한다. 이상과 같이 구성된 LVS 클러스터 시스템과 Load 서버에 다음과 같은 소프트웨어들을 설치하였으며, 시스템 및 load 서버 군(群)의 구성 형태는 (그림 5)와 같이 구성되었다.

Load 서버는 모두 NT 4.0과 Windows 2000 서버를 사용하였고, 일반적인 설정에 WWW와 ftp 서비스를 제외시킨 형태로 설치하였다. 이들 3대의 서버 각각에는 부하를 생성해내는 WebLoad 1.3과 Load 서버와 모니터의 통신을 위한 TestTalk를 설치하였다. 본 연구에서는 Platnum사에서 개발한 WebLoad 1.3 package를 이용하여 가상의 클라이언트를 생성하고, 가상의 클라이언트를 LVS 클러스터 시스템에 접속시켜 부하를 부여하는 방식으로, 생성되어진 각각의 접속은 director로 보내어지고, 작업 할당 방법에 따라 real 서버로 배분된다. real 서버는 요청에 대한 처리를 하고, 그 결과를 클라이언트로 전송하는데, 각각의 real 서버 자체 내에서는 부팅후 분 단위의 간격으로 사용 가능한 자원의 현황을 파악하여 그 결과를 director 서버 내의 MySQL 데이터베이스에 저장된다. 마지막으로 load test 과정을 감시 및 통제하는 모니터에는 TestTalk와 모니터용 소프트웨어를 설치하였다.

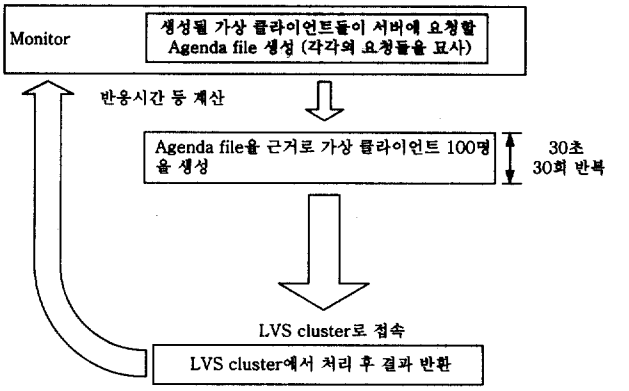
4.2 네트워크 시뮬레이션 변수 선정

Real 서버의 부하 정도를 파악하는데 사용된 자원은 사용 가능한 물리적인 메모리의 양과, uninterruptable sleep 상태에 있는 프로세스의 수, swap out된 프로세스의 수 등이 있다. 사용 가능한 물리적인 메모리의 양을 주요 파라메터로 선정하였는데, 클라이언트로부터 요청이 오면 Apache와 같은 preforking 방식의 서버는 처음 서비스를 시작할 때 임의의 수(본 실험은 default로 10개)의 서버 프로세스를 실행해두고 요청이 들어올 때마다, 이를 처리하며, 처리할 여분의 서버 프로세스가 없는 경우에는 서버 프로세스 수를 증가시켜 요청을 처리하게 된다. 이 때 서버 프로세스는 보통 2.7~3.5Mbyte의 메모리와 별도의 swap 메모리를 차지하게 된다.[1, 11] 웹서버는 uninterruptable한 프로세스의 값이 non-zero 값을 가지는 경우, I/O 또는 또 다른 어떤 이벤트를 프로세스가 기다리고 있는 상황이 발생한 것으로 간주할 수 있다. 계속적으로 이 값이 non-zero를 가지면 디스크 throughput문제가 있다고 판단되기 때문이다.[11] 마지막으로 swap out된 프로세스 수가 non-zero인 경우, 시스템의 물리적인 메모리의 양이 부족하여 swap out이 발생되는 것이다. page-in 또는 page-out과정에서 메모리가 충분하다면 page-in과 page-out 만이 발생하겠지만, 메모리가 부족하게 되면 현재 메모리에서 불필요한 프로세스를 swap-out 시키고, 이전에 swap out된 프로세스의 데이터를 disk에서 swap-in 시킨다. 이때 상대적으로 swapping을 하는 데에 CPU time이 모두 소모되기 때문에 시스템의 throughput이 현저히 감소된다.[11] Swap out된 프로세스 수가 non-zero 값이 계속되면 메모리 부족 현상을 나타내는 것이기 때문에 시스템 성능에 영향을 준다고 할 수 있다.[10, 11] 각 real 서버의 부하를 측정하기 위하여 일반적인 UNIX 계열(BSD, System V, Linux)의 vmstat를 사용하여 이상과 같이 선정된 파라메터들에 관하여 측정하였다.

본 연구에서는 다음과 같은 가정 하에서 진행되었는데, 첫째는 웹서버의 사용 가능한 메모리 양을 균등하게 배분한다는 관점 하에서 CPU 성능에 대한 고려는 하지 않았다. 이는 일반적으로 웹서버의 성능은 CPU의 성능보다는 사용 가능한 메모리의 양에 더 영향을 받기 때문이다.[1] 둘째는 director에 대한 성능도 고려하지 않았다. 즉, 클라이언트의 요청이 많이 발생한다 하더라도, director에서 클라이언트의 요청을 real 서버로 재지향시키는 일련의 과정들이 큰 부하를 주지 않는다고 하였기 때문이다.[14]

4.3 시스템 부하 생성

본 연구에서 가상으로 LVS 클러스터 시스템으로 향하는 다수의 접속을 생성하기 위해 (그림 6)과 같이 가상의 클라이언트들이 LVS 클러스터 시스템으로 보낼 요청을 생성해낸 후, LVS 클러스터 시스템으로 전송한다. 생성되는 부하는 15분간 Load 서버에서 30초마다 100명의 가상 클라이언트가 생성된다. 본 연구에서는 가상의 클라이언트를 생성하는 tool로써 WebLoad를 사용하였는데, WebLoad의 실행절차는 다음과 같다.



(그림 6) 부하의 생성 절차

- 실제 클라이언트가 대상 사이트에 접속하여 실험에서 요청할 내용을 포함하는 Agenda file을 script 문을 이용하여 작성한다.
- 작성된 Agenda file의 내용을 그대로 반복하는 가상 클라이언트를 단위시간 당, 다수를 생성하여 LVS 클러스터 시스템에 접속시킨다.
- Agenda file의 선택된 컨텐츠들을 요청하여 대상 사이트에 부하를 부과하게 되고, 요청이 이루어지고 결과가 반환되면 기록하고, 가상 클라이언트는 소멸된다.

가상 클라이언트가 가지고 있는 Agenda 파일에는 실험에 사용된 대상 사이트의 html 문서들과 그래픽 파일 및 CGI 게시판에 약 1Kbyte 정도의 메시지를 쓰도록 작성되었으며, 본 연구에서는 실험 대상 사이트의 html 문서 및 그래픽 파일과 CGI 응용프로그램 등 40개의 컨텐츠들이 요청된다. 대상 사이트에 가상 클라이언트가 접속하면, HTTPD 프로세스에 의하여 약 3.6Mbyte의 메모리가 소비되고, 이어서 CGI 게시판을 요청하게 될 경우 약 1.7 Mbyte의 메모리를 차지하는 아들 프로세스가 생성된다. 물론 물리적인 메모리 공간을 차지하는 동시에 swap 공간 또한 일정량을 차지하게 된다.

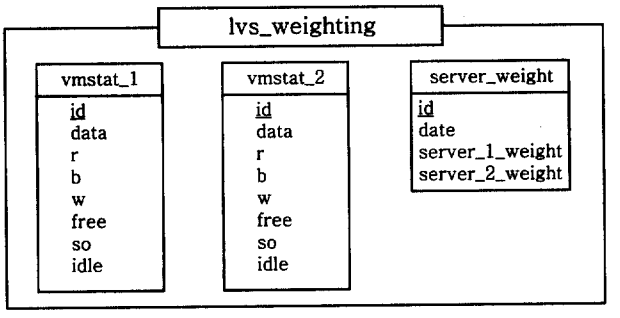
4.4 웹 사이트의 구성 및 데이터베이스

본 연구의 네트워크 시뮬레이션에 사용된 웹사이트의 컨텐츠는 일반적인 html로 된 문서와 gif 포맷 형식의 아이콘(icon) 및 배너(banner)와 데이터베이스를 사용하지 않고, 파일 수준의 입출력을 처리하는 인터프리터(Interpreter)방식의 Perl로 프로그래밍 된 CGI 게시판으로 구성되었다.

LVS 클러스터 시스템의 각각의 real 서버의 측정 결과는 director내의 데이터베이스에 저장되고, 데이터베이스의 구조는 (그림 7)과 같다. (그림 7)의 데이터베이스에는 3개의 테이블이 존재하며, 각 테이블 내의 속성들은 이름은 같지만, 2개의 테이블 vmstat_1과 vmstat_2는 각 서버 별로 측정된 성능 측정 결과의 시간적인 차이를 살펴보기 위해, 각기 다른 테이블의 같은 속성들은 어떤 관계도 갖지 않도록 구성했다. 전체 실험을 걸쳐 vmstat_1과 vmstat_2 테이블에 데이터들이 작성되는 시간의 차이는 약 1~2초 정도여

서, 실험 동안의 각 real 서버 별 성능 측정은 거의 동시에 측정되었다고 할 수 있다. vmstat_1과 vmstat_2는 real 서버 1과 real 서버 2의 성능 결과를 저장하는 테이블들이며 각 속성들은 다음과 같다.

id : 각 데이터들이 저장되는 순서(auto_increment) ;
date : 해당 데이터, 즉, r, b, w, free, idle 값이 저장된 시간 ;
r : 현재 생성되어 running 중인 프로세스 의 수 ;
b : uninterruptable 프로세스 의 수 ;
w : swap out 된 프로세스의 수 ;
free : 현재 가용한 물리적 메모리의 량 ;
so : 초당 swap out 되는 데이터의 양(KByte/s) ;
idle : 전체 CPU 휴지시간에 대한 percentage.



(그림 7) 시뮬레이션 저장용 데이터베이스

5. 시뮬레이션 결과 및 분석

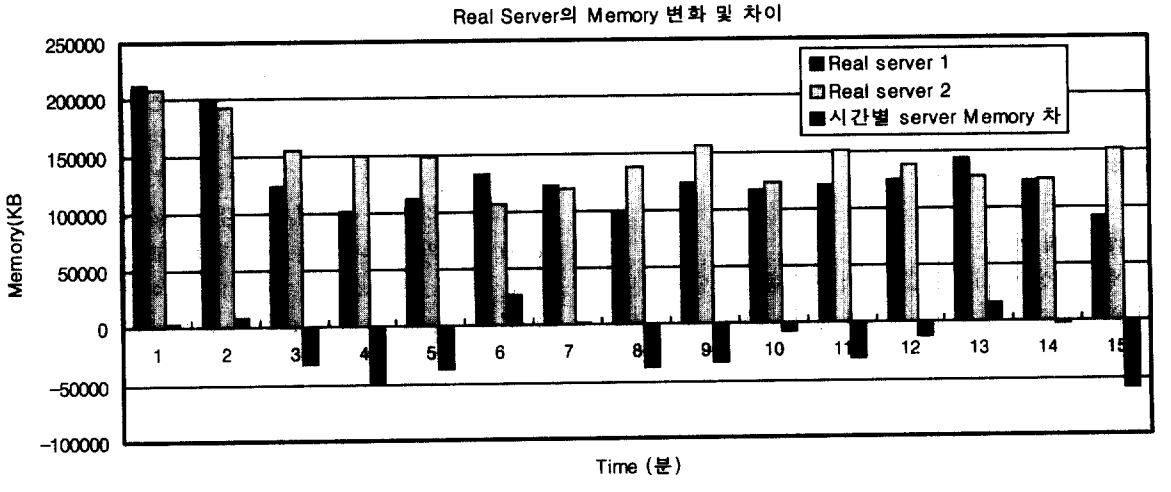
5.1 WLC 방법

기존 LVS 클러스터 시스템은 real 서버의 자원 손실을 고려하지 않으며 관리자가 임의로 가중치를 선정한다. LVS 클러스터 시스템은 가중치와 real 서버 당 접속수로 식 (1)의 값을 가지는 서버로 다음 접속을 할당한다. 그러나 real 서버가 작은 접속수 또는 동일한 접속수를 각각 할당된 경우라도, 총 부하량이 접속이 보다 많은 다른 real 서버 쪽의 부하보다 더 클 수 있다는 것을 분석하기 위함이다. (그림 8)은 작업 할당 알고리즘으로 WLC를 이용하는 LVS 클러스터 시스템에 대해 일정한 부하를 부과하였을 때, LVS 클러스터 시스템을 구성하는 2대의 real 서버의 메모리 소모량을 표시한 것이다. 실험 방법은 매 30초에 4,000개의 요청을 LVS 클러스터 시스템에 접속시켰을 때, 각 real 서버의 메모리 변화량을 나타낸 것이다. 실험에서는 각 real 서버를 동시에 부팅하였고, 같은 양의 사용 가능한 메모리를 확보한 상태에서 실험이 시행되었다. 각 real 서버의 초기 가중치를 1로 설정하였다.

실험 결과 값은 (그림 8)과 같이 real 서버 2로 더 많은 부하가 편중되고 있음을 알 수 있다. real 서버 1로는 반환되는 결과의 크기가 작은 요청들이 많이 할당되었고, real 서버 2로 보내진 요청들은 real 서버 1보다는 반환되는 결과의 크기가 상대적으로 큰 CGI 등에 대한 요청이 많이 할당된 결과라고 할 수 있다. 본 실험에서 가상 클라이언트가 수행하는 Agenda 파일은 모두 동일한 Agenda 이지만 Agenda 내의 요청들은 반환되는 크기가 각각 다르다.

각각 분산된 요청들은 html 문서, 이미지 파일 등 대개 1~2KByte 정도의 작은 것들 일 수도 있고, CGI 또는 java 애플릿 등 서버 측 응용프로그램으로써 서버 측에서 실행되어 아들 프로세스를 생성하거나, html 또는 이미지파일 보다 최소 몇 배에서 수십 배 이상 큰 용량을 가지는 컨텐츠 등으로 다양하다. (그림 9)는 본 실험에서 사용된 Agenda file을 나타낸 것이다.

본 실험에서 사용된 CGI 게시판은 한번의 요청으로 여러 번의 트랜잭션을 발생시키지 않는다. 즉 한번의 요청으로 2대의 real 서버로 트랜잭션이 분할되지 않는다는 의미이다. CGI가 실행 요청이 접수되면 HTTPD 프로세스는 아들 프로세스인 perl 프로세스를 fork( )/exec( ) 등의 시스템-콜을 사용하여 생성하고, 프로세스가 요구하는 만큼의 물리적 메모리를 소비하게 된다. 오직 한 클라이언트가 시스템에 접속하여, real 서버 1로 보내졌다고 할 때, 클라이언트가 index.html page를 요청하게 되면 real 서버 1은 index.html page를 클



(그림 8) WLC 방법의 메모리 소모량

라이언트에게 보낸다. 그 후 클라이언트 브라우저가 index.html을 파싱(parsing)하는 과정에서 이 문서에 링크된 컨텐츠들을 다시 요청한다. 이때의 요청들은 real 서버 1과 2로 분산되어 보내지게 되는 것이다. 이렇게 반복되는 과정에서 CGI 응용프로그램의 요청이 발생하고, 이 요청이 접속이 작은 real 서버로 전송되어진다.

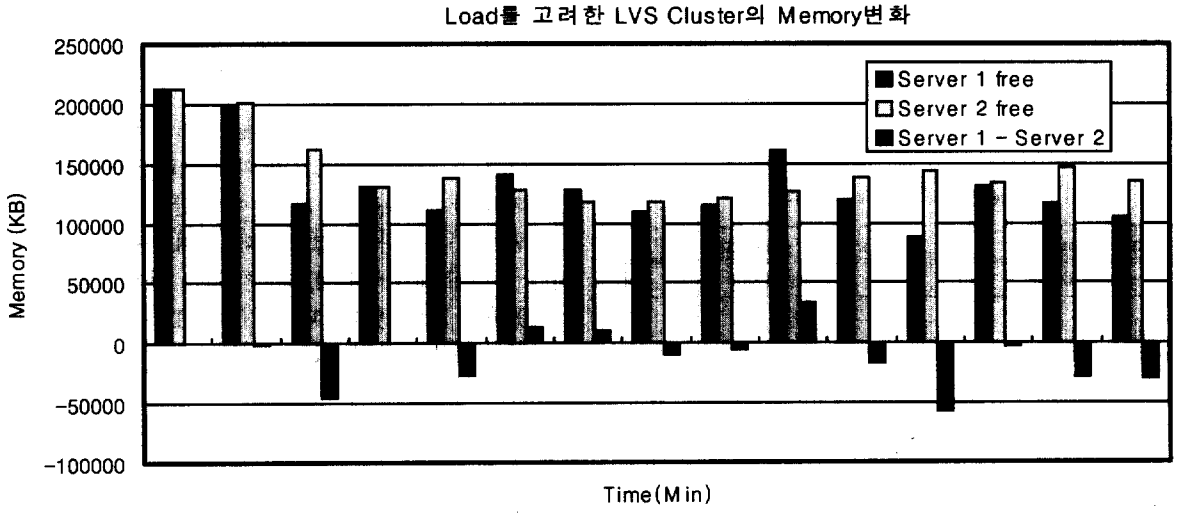
```
do GET("http://210.107.211.59/책표지/learning_perl.gif") {
}
do GET("http://210.107.211.59/cgi-bin/wowboard/board.cgi") {
}
do GET("http://210.107.211.59/image/write.gif") {
}
do GET("http://210.107.211.59/image/fix0.gif") {
}
do GET("http://210.107.211.59/image/delete0.gif") {
}
do GET("http://210.107.211.59/image/list0.gif") {
}
do GET("http://210.107.211.59/image/back0.gif") {
}
do GET("http://210.107.211.59/image/next0.gif") {
}
do GET("http://210.107.211.59/image/admin.gif") {
}
do GET("http://210.107.211.59/cgi-bin/wowboard/board.cgi?bd
=&list=0&j=form") {
}
do POST("http://210.107.211.59/cgi-bin/wowboard/board.cgi") {
    FORMDATA {
        j = "write" ;
        u = "" ;
        bd = "" ;
        id = "kalsman" ;
        passwd = "kalsman" ;
        email = "kalsman@keobuksun.kmu.ac.kr" ;
        upfile = "" ;
        title = "2000" ;
        content = "2000년11월4일오후7시테스트를 하고 있습니다." ;
    }
}
```

(그림 9) 네트워크 시뮬레이션용 Agenda file

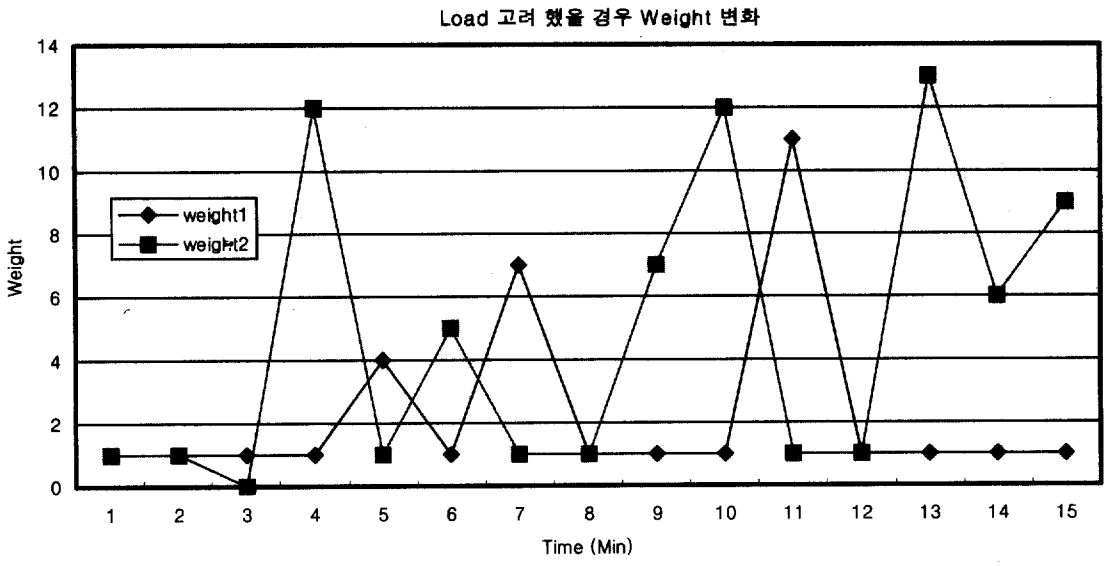
접속이 작은 서버로 보내진 CGI 요청은 지금까지 보내진 html 또는 작은 이미지파일 등을 모두 합한 것보다 더 많은 메모리를 요구하게 되어 접속수 기준으로는 균등한 분산이지만, 메모리 자원의 소비 면에서는 공평한 분산이라고 예측할 수 없기 때문이다. 이와 같은 과정들이 수많은 클라이언트에 의해서 반복될 경우, 특정 서버로 메모리의 소비가 편중될 수 있다.

5.2 부하 분산 방법 1

부하 분산 방법 1에서도 5.1절에서 제시된 동일한 Agenda file을 사용하였는데, 시뮬레이션 결과는 (그림 10)과 (그림 11)에 각각 메모리 변화와 가중치의 변화를 보여주고 있다. 네트워크 시뮬레이션은 부하를 고려하지 않은 LVS 클러스터 시스템과 같은 환경 및 조건에서 실시하였으며, 실험 초기의 가중치는 real 서버 1과 real 서버 2 동일하게 1이었다. 또한 director에서 1분 간격으로 각 real 서버의 가중치를 변경시키는 일련의 과정이 포함되었다. 시뮬레이션 결과는 부분적으로 각 real 서버의 메모리 소비가 한 쪽으로 치중되는 것도 있었지만, 전체적으로 메모리 소모량이 real 서버 1과 real 서버 2에서 비교적 균형을 이루었다. 단지 초기 부분에 메모리의 소비가 한쪽 real 서버로 편중되는 경향을 보이는데, 이것은 실험 시작 3분 정도 후에 uninterruptable한 프로세스가 발생했기 때문이다. 즉, 클라이언트들로부터 초기 요청이 쇄도하면서, 디스크로부터 데이터를 읽어들이는 과정에서 빠른 CPU 또는 메모리의 속도에 비해 850배 이상 느린 real 서버 2의 disk에서 병목현상이 발생하여 가중치가 0이 되었고, 가중치가 0이 되면, LVS 클러스터 시스템의 director는 real 서버 2로 클라이언트의 요청을 할당하지 않는다. 그 결과 실험 3분 경과 후 real 서버 1의 메모리는 (그림 10)에서 보는 바와 같이 많은 소비를 보이게 되었다. 실험이 어느 정도 경과한 후 CPU의 idle time percentage가 0이 되어 CPU는 더 이상 가중치의 산정에 포함되지 않게 됨을 알 수 있다. 실험의 마지막 부분에서는 다시 CGI보다는 html의 요청이 더 많아



(그림 10) 평균 메모리 양을 3,500 KByte / 프로세스로 한 경우의 메모리 변화



(그림 11) 평균 메모리 양을 3,500 KByte/프로세스로 한 경우의 가중치 변화

지므로 이와 같은 결과가 발생했다고 판단된다. 또한 메모리의 소비량과 가중치의 변화 추이를 보면 정확히 일치하지 않는다. 이것은 실험의 처음과 마지막에는 CPU의 활용도가 100%, 즉 CPU idle time percentage 가 0이 되지 않아. CPU의 idle time percentage의 10% 정도가 가중치에 영향을 주게 되어, 그 차이로 인하여 발생했다고 판단된다. 전체적으로 실험 결과는 가중치의 변화가 일어나 각 real 서버의 메모리 소비가 한쪽으로 치우는 것을 교정하고 있으며, 각 real 서버의 메모리 소비량에서 감소되었다고 할 수 있다.

시뮬레이션 결과가 WLC 방법보다 향상은 되었으나, 예상보다 만족스럽지는 못한 원인은 LVS 클러스터 시스템의 향상은 WLC 작업할당 방법의 문제로 분석된다. WLC 방식에서 다음 접속을 할당할 real 서버를 선택할 때에는 앞에서 언급한 식 (1)이 사용된다. 각 real 서버의 해쉬테이블에 남아있는 접속수를 active connection과 inactive connection 수를 합한 것을 근거로 계산한다. <표 1>은 전체 실험 기간 동안 30초 간격으로 4,000개씩의 connection이 클러스터 시스템으로 접속될 때, 어느 시점의 real 서버 별로 active connection 수와 inactive connection 수에 대해 무작위로 측정한 결과를 나타낸 것이다.

<표 1> Active, Inactive connection의 수

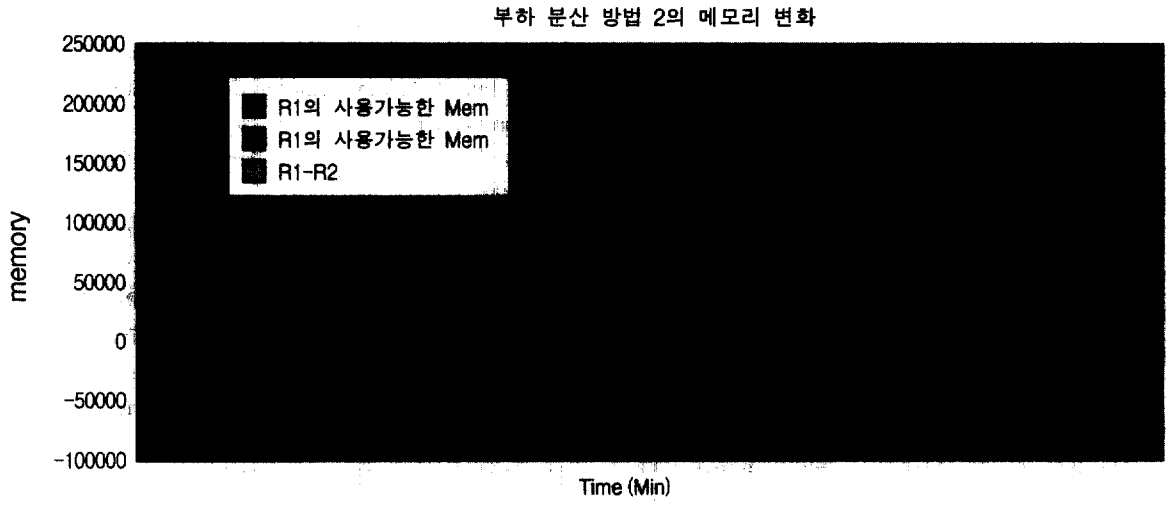
active connection	inactive connection
30~100	1,900~1,970

LVS 클러스터 시스템은 real 서버에 기록된 접속을 active connection과 inactive connection 수로 판단하는데, inactive connection은 이미 이전에 요청을 처리한 후 다음 접속을 위해 time-out 시간을 두어 다음 접속을 기다리고 있는 것이며, 실제 현재 접속은 active connection이다. 실제 real 서버의 메모리에 가장 큰 영향을 주는 접속인데 가중치를 계산하는 과정에서 접속수에 inactive connection과 active connection이 모두 포함된다. 어느 시점에서 가중치의 변화가 발생하여, real 서버 1의 가중치가 real 서버 2의 가중치 보다 크다면, real 서버 1로 많은 접속이 할당될 것이고, 그 결과 real 서버 1의 메모리가 더 많이 소모될 것이며, 또 접속수도 많아지게 된다. 그리고 대부분의 접속은 inactive connection으로 유지되게된다. 이러한 상황이 반복되면, 어느 한 real 서버가 가중치도 크고, inactive connection 수도 더 큰 상황이 발생할 수 있다. 실제 실험 기간 중 실제 가중치 차이가 크고, 접속수가 많기 때문에 가중치가 작은 real 서버로 할당된 결과로 메모리의 소비가 다소 편중되는 경향을 보인 것으로 판단된다.

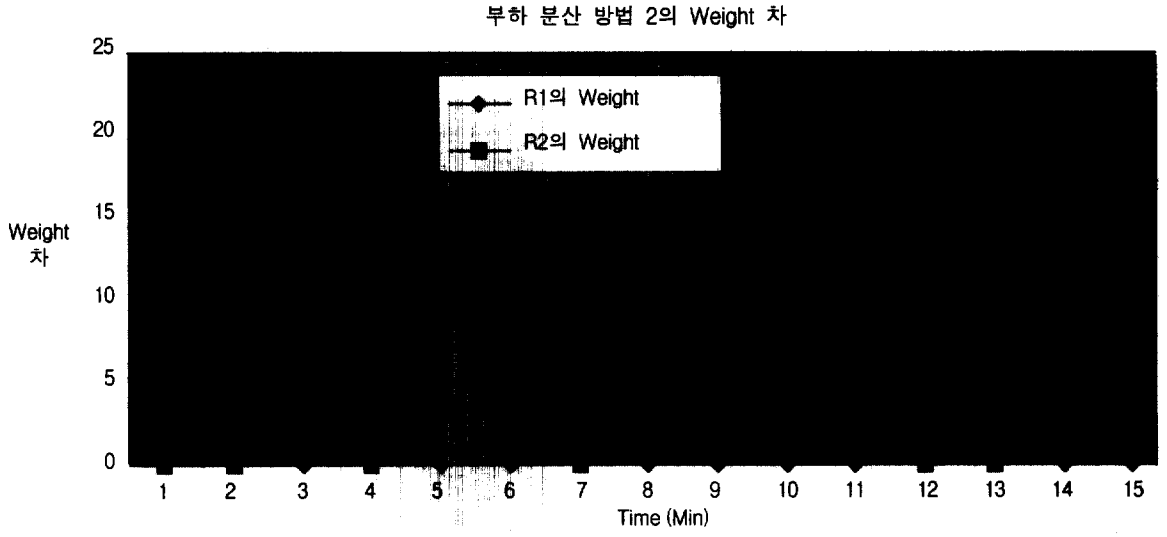
5.3 부하 분산 방법 2

부하 분산 방법 2는 기본적인 모든 과정은 부하 분산 방법 1과 동일하며, 가중치 산정 방법을 식 (3)을 적용하여 각 real 서버의 가중치에 의한 민감도를 둔화시키고자 하였다. 부하 분산 방법 2에 의하여 (그림 12)와 (그림 13)과 같은 real 서버의 사용 가능한 메모리 양의 변화와 가중치 차이에 대한 결과를 얻었다.

부하 분산 방법 1보다 각 real 서버의 메모리의 소비를 균등하게 하고 있으며, 메모리 차에 대한 평균값을 보면 절반 가까이 감소되었음을 <표 2>에서 알 수 있다. (그림 13)은 가중치의 차를 나타내고 있는데, 실제 적용 알고리즘에 의하여 각 real 서버에 대한 가중치 값은 적게는 87부터 많게는 1000까지 계산되었다. 따라서 (그림 13)의 가중치 차는 상대적인 변화라고 할 수 있는데, 예를 들어 처음 1분 후에 계산된 가중치는 real 서버 1과 2에 405와 400이며, 8분 후의 가중치는 각각 667과 670으로 계산된 값이다. 가중치의 변화가 처음 얼마간은 real 서버 1이 크나, 점차 real 서버 2의 가중치와 real 서버 1의 가중치가 교차하는 이른바 파형(波形)을 보이고 있다. 이 현상은 각 real 서버의 메모리를 번갈아 가



(그림 12) 부하 분산 방법 2의 메모리 변화



(그림 13) 부하 분산 방법 2의 가중치 차

며 소비하여 메모리의 소비를 양쪽으로 분산시키려는 결과를 보여주고 있는 것이라 할 수 있다. 실제 데이터에서도 메모리 차의 값이 이전의 실험들 보다 더 작은 값으로 나타나 교정력(correction potentiality)이 향상되었음을 알 수 있다.

〈표 2〉 부하 분산 방법의 메모리 변화

LVS cluster의 평균 메모리의 변화 (KByte)			
	Real 서버1	Real 서버2	Real서버1 - Real서버2
WLC 방법	129680.1	145873.9	-16193.1
부하 분산 방법 1	132279.8	143608.1	-11328.3
부하 분산 방법 2	134530.9	141352.6	-6821.7

전체 시뮬레이션을 진행하는 동안 부하를 고려하지 않은 경우와 부하를 고려한 경우에 대하여 다양한 실험을 실시하였는데 메모리의 소비가 어느 한쪽으로 편중되는 것을 완전히 수정 할 수는 없었다. 그러나 <표 2>와 같이 부하를 고려한 경우가 각 real 서버의 자원이 좀더 효율적으로 사용할 수 있음을 볼 수 있었다. 본 연구의 실험에서 가장 중요한 요소는 LVS 클러스터 시스템을 이용하는 클라이언트의 분당 평균 접속수와 가중치를 산정하는 것이라 할 수 있다. 접속수와 가중치를 산정하기 위하여 각 real 서버의 'access.log' file을 분석하고, 일반적으로 UNIX 시스템의 ps 및 vmstat, uptime 등의 tool들을 이용하여 주기적이고, 긴 시간동안 웹서비스와 관련된 HTTPD 프로세스의 수와 HTTPD 프로세스가 차지하는 메모리 양, 측정 시점에서 사용 가능한 메모리 양, 그리고 웹사이트에서 사용하는 CGI 또는 서버 API등에 대한 프로세스 정보 등을 근거로 하여 판단하여야 할 것이다.

6. 결 론

본 연구는 LVS 클러스터 시스템내의 각 real 서버의 부하량에 관한 정보를 서버 자원의 배분에 활용하는 방안을 제시

하여 서버의 메모리 편중 현상을 줄이는 LVS 클러스터 시스템을 구성하는 방안을 제시하는데 목적이 있었다. 제시한 방법에서는 각 real 서버의 부하를 주기적으로 측정하고, 그 결과를 바탕으로 LVS 클러스터 시스템 내의 각 real 서버의 가중치를 산정하고 이를 주기적으로 변경시켜 주었다. 그 결과 real 서버의 부하를 실질적으로 반영할 수 있게 되어 LVS 클러스터 시스템의 특정 real 서버의 메모리가 지속적으로 소비되는 현상을 보완할 수 있음을 보였다. 실제 실험을 통하여 real 서버의 부하에 관한 상태 정보를 고려하지 않은 것 보다 본 연구에서 제시하는 방법이 시스템의 자원을 보다 효율적으로 사용할 수 있음을 입증하였다.

본 연구의 결과는 클러스터 시스템을 구성하는 방법론으로 서버의 사양은 항상 동일한 제품이거나 동일한 용량의 서버로만 구성해야된다는 IT 사회의 고정적인 관념을 초월하여 이기종의 서버로 클러스터 시스템을 구성할 수 있는 방법론을 제시하고 있다. 즉 서버의 자원을 적절히 배분한다면 효율적인 이기종으로 구성되는 클러스터 시스템을 구성할 수 있다는 것이다. 본 연구에서는 언급되지 않았지만, 네트워크의 대역폭과 웹사이트를 구성하는 컨텐츠들의 배치와 크기, 적절한 CGI 응용프로그램용 언어의 선택과 데이터베이스의 활용 문제, 서비스의 한계 등에 대해서도 고려해 볼 필요가 있다. 각 real 서버에서 데이터를 수집하는 것 자체가 일시적으로 부하를 줄 수 있는데, real 서버의 자체 부하로 인해 수집된 데이터의 정확도가 낮기 때문이다.

참 고 문 헌

[1] 권원상, 역, "웹 성능 최적화 by Patrick Killelea", 한빛미디어, 2000.

[2] 김영식, 강윤석, 역, "perl 제대로배우기 by Randal L. Schwartz & Tom Christiansen", 한빛미디어, 1999.

[3] 김영식 외 2인, 역, "perl 프로그래밍 by Larry Wall", 한빛미디어, 1998.

[4] 문정훈, 역, "MySQL과 mSQL by Randy Jay Yarger, George Reese & Tim King", 한빛미디어, 2000.

[5] 박창민, 역, "TCP/IP 네트워크관리 by Craig Hunt", 한빛미디어, 1999.

[6] 윤석범, "CGI와 PHP", 도서출판 대림, 1999.

[7] 장훈, 역, "시스템 관리의 핵심 by AEleen Frisch", 한빛미디어, 1998.

[8] 한동훈, 이만용, 역, "Linux programing by Neil Matthew, Richard Stones", 도서출판 대림, 1998.

[9] Daniel A. Menasce, virgilio A.F. Almeida, "Capacity Planning for WEB PERFORMANCE," Prentice Hall PTR, 1998.

[10] J Purcell, "Linux Complete Command Reference," Redhat press, 1997.

[11] Mike Loukides, "System Performance Tuning," O'Reilly, 1990.

[12] Olaf Kirch, "The Linux Network Administrator's Guide," O'Reilly, 1994.

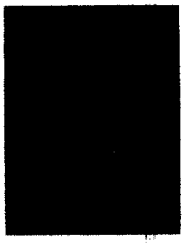
[13] PLATINUM, "WebLoad User Guide," PLATINUM, 1997.

[14] Wensong Zhang, Shiyao Jin, Quanyuan Wu. "Creating Linux Virtual servers," Ottawa Linux symposium 2000, 2000.

[15] 인터넷 통계 서비스, http : //www.i-biznet.co.kr/inet/inet-19990628190801.htm.

[16] 정하녕, 역, "NET3-4-HOWTO by Terry Dawson," http : //kldp.org/HOWTO/html/NET3-4-HOWTO.html.

[17] Joseph Mack, Wensong Zhang, "The Linux Virtual Server HOWTO," http : //www.linuxvirtualserver.org/Joseph.Mack/LVS-HOWTO-991205.gz, 1999.



김 석 찬

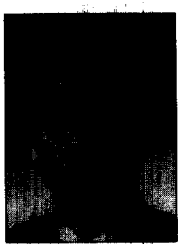
e-mail : kalsman@hanmail.net

1999년 계명대학교 산업공학과 학사

2001년 계명대학교 산업공학과 석사

2001년~현재 계명대학교 산업공학과 박사 과정

관심분야 : Network 모델링 및 성능 분석, Web 서버 용량 계획 클러스터 시스템, 네트워크 관리 시스템



이 영

email : yrhee@kmu.ac.kr

1983년 고려대학교 산업공학과 학사

1985년 고려대학교 산업공학과 대학원 석사

1990년 오클라호마대학교 산업공학과 석사 (전공 : IE)

1994년 노스캐로라이나 주립대학교 박사 (전공 : OR & CS)

1990년~1995년 연구원, Center for Comm. Signal Processing at NCSU

1995년~1998년 수석연구원, 삼성 SDS, 정보처리기술 연구소

1998년~현재 계명대학교 산업공학과 조교수

관심분야 : Network 모델링 및 성능 분석, Web 서버 용량 계획, 전자 상거래 지불시스템, 전자결재시스템