리눅스 클러스터형 웹 서버 설계†

고려대학교 권세오 · 김상식 · 김동승*

1. 서 론

클러스터 체계는 병렬/분산 처리를 이용하여 성능을 높이는 방식으로 최근 많이 활용되는 기술로서 다양한 방법으로 구현되고 있다. 즉, 네트워크로 접속된 워크스테이션들을 사용해 구성될 수도 있으며, 일반적인 PC를 네트워크로 연결해 구축할 수도 있다. 이중에서 Linux(리눅스) 네트워크 클러스터는 가격대 성능비가 탁월한 것으로 알려졌는데, 그 효시인 Beowulf[8]는 1994년 NASA의 CESDIS에서 16-node cluster(클러스터)를 리눅스와 표준 소프트웨어 패키지를 이용해 개발하였다. 이후 많은 클러스터가 제작되고 있으며, 미국 Los Alamos 국립 연구소에서는 70개의 리눅스 박스로 구성된 Avalon Beowulf 시스템을 직접 만들었으며, Top 500 supercomputer list에 315위로 등록되어 있다[11].

PC/Workstation 클러스터는 슈퍼 컴퓨팅 분야 외에도 다양한 분야에 활용되고 있는데, 그 중 하나가 인터넷과 월드 와이드 웹(World Wide Web) 분야이다. 이들에 대한 사용량은 폭발적으로 증가하고 있으나, 이런 수요 증가에 비해 제공되는 서비스들의 질은 그만큼 따라가지 못하는 것이 현실이다. 그 주요 원인은 웹 서버의 서비스 성능의 부족에서 오는 것이며 클러스터형 웹 서버가 중요한 해결수단으로 간주되고 있다.

클러스터형 웹 서버란 클라이언트(client)들로부터의 서비스 요청을 클러스터나 여러 컴퓨터들 사이로 분배시켜주는 구조를 갖는 웹 서버를 말한다. 이런 구조는 일을 처리할 컴퓨터의 개수가 많으므로 처리성능이 개선되고 높은 가용성(high availability)을 제공할 수 있게 되는 우수성을 갖는다. 하나의 컴퓨터로 웹 서버를 만들 경우, 그 컴퓨터가 이상이 생기면 전체 서비스가 불가능해지게 된다. 이에 비해 클러스터형 웹 서버는 다수의 컴퓨터로 구성되어 모든 노드가 동시에 죽지 않는 이상 계속 서비스가 이루어질 수 있어 안정적인 서비스를 요하는 분야에서 긴요하게 사용될 수 있다.

본 글에서는 PC 클러스터 구축 방법에 대한 간단한 소개와 이것을 활용한 리눅스 클러스터형 웹 서버 구조 설계 및 부하 분산 방식과 관련 실험 결과를 기술한다.

2. PC 클러스터

잘 알려진 HP, SGI, SUN, DEC 등의 워크스테이션들은 실질적으로 그 성능에 비해 가격이 비싸 쉽게 접하기 어렵다. 특히, 이러한 컴퓨터들은 다른 시스템과의 하드웨어와 소프트웨어 호환성의 부족으로, 사용자가 직접 조립하고 적당한 소프트웨어를 설치하여 사용한다는 것은 거의 불가능하다. 이러한 고가의 컴퓨팅 환경의 대안으로 NASA를 비롯한 미국내 주요 연구소에서 Beowulf 프로젝트를 통해 시장에서 누구나 살 수 있는 PC 하드웨어와 쉽게 구할 수 있는 리눅스를 통해 하이-엔드 컴퓨팅 환경 구축을 시도하여 성공하였다. Beowulf 클러스터 시스템의 가

† 이 연구는 정보통신부 대학기초(97-G-0620) 및 과학재단 국제공동연구(985-0900-003-2)의 지원에 의해 수행되었음.

* 종신회원

장 큰 성공요인은 최근 PC에서 사용하는 고성능 CPU의 지원에서 비롯된다. 최근 개인용 컴퓨터 시장이 커지면서 결국 워크스테이션에 맞먹는 컴퓨팅 능력을 갖는 PC가 등장하고 있으며, 네트워크 시스템의 대량생산에 따라 싼 가격의 고속의 네트워크 장비를 살 수 있게 됐으며, 주변에서 적절한 사양의 PC를 모아서 연결하는 것만으로 PC 클러스터를 만들 수 있다.

하드웨어의 발전과 더불어 리눅스를 운영체제로 사용함으로서 PC 클러스터는 가격대 성능비의 장점을 더 살릴 수 있다. 일반적으로 리눅스는 기본적으로 FSF(Free Software Foundation)라는 단체와 *GNU Project*에 뜻에 따라 무료로 배포되지만 성능은 상용 유닉스에 못지 않게 강력하며 안정성이 뛰어나다고 인정받고 있다. 무료로 제공되는 리눅스는 클러스터가 높은 가격대 성능비를 갖는데 큰 역할을 하고 있다. 또한, MPI[12], PVM[10]같은 표준화된 통신체계를 상용 병렬컴퓨터와 마찬가지로 자유롭게 설치, 사용할 수 있게 됨에 따라, 기존의 병렬 컴퓨터에서 만든 코드를 그대로 쓸 수 있게 되었다.

본 저자의 실험실에서 구축한 PC 클러스터는 동일한 PC 16대로 구성되어 있고 *Pentium* II 300(MHz) 프로세서와 64 Mbyte SDRAM을 사용했다. 각 PC를 하나의 완성된 머신으로 구현하는 것이 필요했기 때문에, 각 머신마다 4.2G Byte의 하드를 달았다. 메인보드로는 Intel BX 칩셋을 사용하는 보드를 사용했으며 네트워크 인터페이스 카드로 Intel Ethernet Express 100+을 사용했다. 각 컴퓨터를 100Mbps 네트워크로 연결하는 데 있어서 흔히 사용되는 허브가 아니라 스위칭 허브(BayStack 450 Switch)를 사용했다. 보통 일반 이더넷으로 연결된 네트워크는 shared network로서 허브는 허브내의 각 포트에 단순히 반복해서 신호를 전달해주는 역할을 해줄 뿐인 반면, 스위칭 허브는 각각의 네트워크 세그먼트 내에 있는 노드들의 이더넷 주소를 맵핑해서 단지 필요한 트래픽만 스위치를 통과하는 것을 허락한다. 따라서 이더넷 스위치는 LAN을 경합이 없고 scalable한, 공유가 아닌 배타적인 전송이 실현되도록 선로를 구축해주어서 전체 시스템의 성능 향상에 도움이 된다. 한편 값비싼 switch를 사용하지 않고도 클러스터내의 네트워크 인터페이스 카드를 서로 cross-link해 하이퍼큐브형태의 통신 구조도 만들 수도 있다. 또한 각 노드에 네트워크 인터페이스 카드를 복수로 설치하여 network channel bonding을 하여 네트워크의 대역폭을 증가시킬 수도 있다.

운영체제인 리눅스는 레드햇(Red Hat) 5.1로서 커널 버전은 2.0.34이다. 클러스터는 분산 메모리 방식의 병렬 컴퓨팅 환경을 제공하기 때문에 네트워크로 연결된 컴퓨터들은 특별한 메시지 전달방식 통신 라이브러리를 사용해서 병렬 컴퓨팅을 수행하게 된다. 이런 것들에는 MPI (Message Passing Interface)[12], PVM (Parallel Virtual Machine)[10] 등이 있으며 MPICH(MPI CHameleon)가 설치되었다 이것은 레드햇 패키지로 제공되어서 쉽게 설치가 가능하다.

3. 클러스터로 구성된 웹 서버

클러스터형 구조에서 웹 서비스를 수행하기 위해 각 서버에 http 데몬을 띄우고 동작시켜 클라이언트로부터 들어온 요청을 여러 대의 컴퓨터가 나누어 처리하도록 만든 구조가 클러스터형 웹 서버이다(그림 1). 이는 급격히 늘어나는 서비스/접속요청(*web traffic*)에 신속한 응답을 제공하고 안정적이고 유연성 있는 서비스를 제공함으로써 클라이언트들에게 신뢰를 얻기 위해 매우 필요하다. 이러한 클러스터형 웹 서버의 성능을 충분히 발휘하기 위해서는 지속적으로 들어오는 클

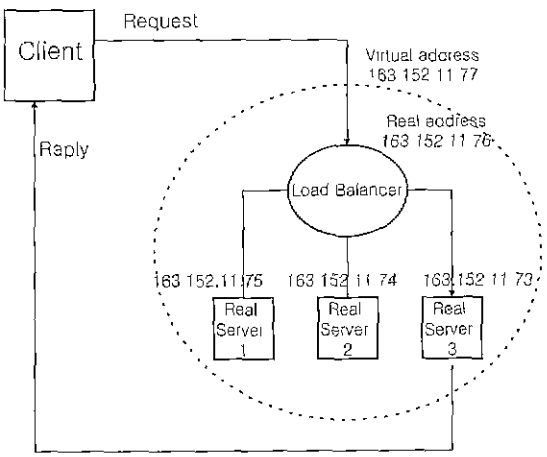

그림 1 클러스터형 웹 서버의 모습

장 큰 성공요인은 최근 PC에서 사용하는 고성능 CPU의 지원에서 비롯된다. 최근 개인용 컴퓨터 시장이 커지면서 결국 워크스테이션에 맞먹는 컴퓨팅 능력을 갖는 PC가 등장하고 있으며, 네트워크 시스템의 대량생산에 따라 싼 가격의 고속의 네트워크 장비를 살 수 있게 됐으며, 주변에서 적절한 사양의 PC를 모아서 연결하는 것만으로 PC 클러스터를 만들 수 있다.

하드웨어의 발전과 더불어 리눅스를 운영체제로 사용함으로서 PC 클러스터는 가격대 성능비의 장점을 더 살릴 수 있다. 일반적으로 리눅스는 기본적으로 FSF(Free Software Foundation)라는 단체와 *GNU Project*에 뜻에 따라 무료로 배포되지만 성능은 상용 유닉스에 못지 않게 강력하며 안정성이 뛰어나다고 인정받고 있다. 무료로 제공되는 리눅스는 클러스터가 높은 가격대 성능비를 갖는데 큰 역할을 하고 있다. 또한, MPI[12], PVM[10]같은 표준화된 통신체계를 상용 병렬컴퓨터와 마찬가지로 자유롭게 설치, 사용할 수 있게 됨에 따라, 기존의 병렬 컴퓨터에서 만든 코드를 그대로 쓸 수 있게 되었다.

본 저자의 실험실에서 구축한 PC 클러스터는 동일한 PC 16대로 구성되어 있고 *Pentium* II 300(MHz) 프로세서와 64 Mbyte SDRAM을 사용했다. 각 PC를 하나의 완성된 머신으로 구현하는 것이 필요했기 때문에, 각 머신마다 4.2G Byte의 하드를 달았다. 메인보드로는 Intel BX 칩셋을 사용하는 보드를 사용했으며 네트워크 인터페이스 카드로 Intel Ethernet Express 100+을 사용했다. 각 컴퓨터를 100Mbps 네트워크로 연결하는 데 있어서 흔히 사용되는 허브가 아니라 스위칭 허브(BayStack 450 Switch)를 사용했다. 보통 일반 이더넷으로 연결된 네트워크는 shared network로서 허브는 허브내의 각 포트에 단순히 반복해서 신호를 전달해주는 역할을 해줄 뿐인 반면, 스위칭 허브는 각각의 네트워크 세그먼트 내에 있는 노드들의 이더넷 주소를 맵핑해서 단지 필요한 트래픽만 스위치를 통과하는 것을 허락한다. 따라서 이더넷 스위치는 LAN을 경합이 없고 scalable한, 공유가 아닌 배타적인 전송이 실현되도록 선로를 구축해주어서 전체 시스템의 성능 향상에 도움이 된다. 한편 값비싼 switch를 사용하지 않고도 클러스터내의 네트워크 인터페이스 카드를 서로 cross-link해 하이퍼큐브형태의 통신 구조도 만들 수도 있다. 또한 각 노드에 네트워크 인터페이스 카드를 복수로 설치하여 network channel bonding을 하여 네트워크의 대역폭을 증가시킬 수도 있다.

운영체제인 리눅스는 레드햇(Red Hat) 5.1로서 커널 버전은 2.0.34이다. 클러스터는 분산 메모리 방식의 병렬 컴퓨팅 환경을 제공하기 때문에 네트워크로 연결된 컴퓨터들은 특별한 메시지 전달방식 통신 라이브러리를 사용해서 병렬 컴퓨팅을 수행하게 된다. 이런 것들에는 MPI (Message Passing Interface)[12], PVM (Parallel Virtual Machine)[10] 등이 있으며 MPICH(MPI CHameleon)가 설치되었다 이것은 레드햇 패키지로 제공되어서 쉽게 설치가 가능하다.

3. 클러스터로 구성된 웹 서버

클러스터형 구조에서 웹 서비스를 수행하기 위해 각 서버에 http 데몬을 띄우고 동작시켜 클라이언트로부터 들어온 요청을 여러 대의 컴퓨터가 나누어 처리하도록 만든 구조가 클러스터형 웹 서버이다(그림 1). 이는 급격히 늘어나는 서비스/접속요청(*web traffic*)에 신속한 응답을 제공하고 안정적이고 유연성 있는 서비스를 제공함으로써 클라이언트들에게 신뢰를 얻기 위해 매우 필요하다. 이러한 클러스터형 웹 서버의 성능을 충분히 발휘하기 위해서는 지속적으로 들어오는 클

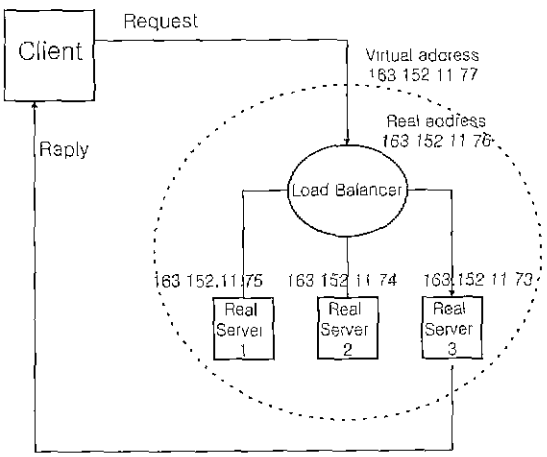

그림 1 클러스터형 웹 서버의 모습

라이언트의 요청을 내부의 모든 서버 각자에게 적절히 분배하여 모든 서버의 부하를 균등히 해야한다. 기존의 방법은 부하 분산 스케줄러가 입력된 클라이언트 요청을 고정된 스케줄링 방식으로 전체 서버에게 분배하였다. 그러나 이러한 방법은 각 클라이언트가 요구한 데이터 크기가 다르고, 각 서버가 처리하는 클라이언트의 요청 건수는 같아도 부하량은 서로 다를 수 있어 서버간 부하가 불균등하게 되고 심각한 경우 더 이상 서비스를 할 수 없게 될 수도 있다.

기존에 나온 관련 연구는 크게 4가지로 클라이언트 측면과 서버측의 round-robin DNS (Domain Name Service), 응용프로그램 수준(application level)에서의 스케줄링, IP 수준(IP level)에서의 스케줄링으로 나눌 수 있다[6]. 첫째로 클라이언트 측면의 접근이 있다 이는 클라이언트 측면에서 제공하는 애플릿(applet)이 클러스터로 구성된 모든 서버들의 부하 정보를 얻기 위해 요청을 하고 수집한 서버들의 부하 정보를 기반으로 서버를 선택하여 그 서버로 클라이언트 자신의 요청을 전달한다. 버클리 대학의 Smart Client가 이에 해당한다[3, 6]. 이 방법은 클라이언트 입장에서 보면 서버가 하나로 인식되지 않고(transparent), 모든 클라이언트의 어플리케이션 부분을 수정해 주어야 한다는 단점이 있다 두 번째로 서버측에서의 round-robin DNS 방법이 있다. 이는 서버측에서만 변경되는 단순한 방법으로 DNS에 round-robin 방식을 적용하여 순차적으로 IP 주소를 대응시켜 서버를 다르게 선택하고 부하를 분산시키는 방법이다. NCSA의 scalable web server가 그 원형이 되었다[3]. 이는 클라이언트 캐싱(caching)과 계층적인 DNS 시스템으로 인해 각 서버가 과부하 상태에 도달할 때 서버를 제어하기 힘들다는 단점이 있다. 세 번째로 서버측에서의 응용프로그램 수준 스케줄링법을 이용한 것으로 EDDIE, Reverse-proxy, pWEB, SWEB 등이 여기에 속한다[1, 3]. 이는 클러스터 내의 서버들이 자신의 부하를 측정하여 클라이언트로부터 HTTP 요청이 들어올 경우, 자신의 부하 상태에 따라 처리 여부를 판단하여 만약 자신이 처리할 수 없으면 클러스터 내의 다른 서버로 HTTP 요청을 전달(forward)하고, 결과를 되받아 이를 다시 클라이언트에게 최종적으로 전송한다. 이 방법의 단점은 2번의 TCP 연결로 인해 전송지연이 심해지고 어플리케이션 수준에서 HTTP 요청과 응답을 처리하는데 많은 오버헤드가 발생한다는 것이다. 마지막으로 서버측에서의 IP 수준 스케줄링 방법이 있다. 이는 네트워크 주소 변환 기법(Network Address Translation)을 사용하여 서로 다른 서버상의 병렬 서비스를 단일 IP 주소에 의한 가상의 서비스처럼 보이게 한다. 부하를 분산시키기 위한 스케줄러 역할의 부하 제어기(load balancer)와 실제 웹 서비스를 제공하는 클러스터 웹 서버 내부의 실행 서버(real server)들로 구성되며 부하 제어기는 요청된 패킷의 주소를 변환하여 선택된 서버로 전달하고 되받은 응답 패킷의 주소를 변환하여 클라이언트에게 보낸다. 버클리 대학의 Magic router와 Cisco의 Local-Director가 이 방식을 쓴다[2, 3, 9] 이 방법은 패킷의 변환(rewriting)에 의한 오버헤드(overhead)가 증가하고 이를 구성하는데 드는 비용이 너무 크다는 단점이 있다.

웹 서비스 요청을 분산시키는 round-robin 스케줄링은 부하 제어기가 들어온 패킷을 순차적으로 각 실행 서버에게 전달하는 방식이다. 이는 기존의 DNS에서 도메인 이름을 IP 주소로 대응시킬 때, round-robin 방식을 써서 등록된 IP 주소를 순차적으로 클라이언트에게 알려주는 방법을 이용한다. 한편 최소 연결수에 의한 스케줄링 방법은 부하 제어기를 통해 각 실행 서버로 네트워크 연결이 이루어질 때, 가장 연결수가 작은(least connection) 서버로 들어온 패킷을 전달하는 방법이다[6]. 리눅스 가상 서버(linux virtual server)가 이 방식을 이용한다.

앞서 언급한 스케줄링 방법은 각 실행 서버가 어느 정도의 부하 상태로 작업을 처리하는가에 따라 들어오는 패킷을 분산시키는 것이 아니라 정해진 순서나 해당 서버들의 연결수를 조사하여 새로이 들어오는 패킷을 처리할 서버를 결정하기 때문에 정적(static) 처리 방식이다. 문자(text)정보 같은 작은 데이터만 요구하는 패킷(HTML)을 받는 경우는 위에 제시된 방법에 의해 균등한 부하가 걸리도록 할 수 있지만, 요즘과 같이 문자 정보뿐만 아니라 화상이나 음악 파일과 같은 많은 양의 멀티미디어 데이터(MPEG, MP3)를

요구하는 패킷을 받는 경우에 다양한 크기의 데이터를 요구하는 패킷을 받게 되면 앞서 소개한 스케줄링 방식은 문제가 된다. 즉 연결수는 같거나 작더라도 요구된 데이터의 크기가 크면 특정 서버의 부하는 증가하게 되고 전체 서버의 부하는 불균등하게 되어 패킷을 처리하는데 많은 시간이 걸리거나 심한 경우 서버가 다운될 수도 있다. 다음은 이런 정적 부하 분산 방식을 개선하여 클러스터형 웹 서버를 구성한 두 가지 방식을 소개한다.

4. 동적 부하조절 웹 서버

동적 부하 균등화를 위한 웹 서버는 웹 데이터를 저장한 파일서버를 단일화하여 복제하는 방식과, 분할, 복수화하여 별도로 처리하는 방식으로 설계하였다.

4.1 단일파일 복제에 의한 복수 서버

이 클러스터형 웹 서버는 IP 수준의 가장법(masquerading)과 터널링(tunneling)을 이용해 빠른 패킷 전송을 하고 부하 제어기의 병목현상을 제거함으로써 성능을 높이고 확장이 쉽다. 이는 1대의 부하 제어기와 여러 대의 실행 서버가 클러스터형으로 구성되고 외부에서는 마치 단일 IP 주소를 가진 한 대의 서버처럼 보이게 된다. 자세한 설명은 다음과 같다.

4.1.1 부하 제어기(load balancer) 설계

부하 제어기의 내부는 크게 하위부(커널 수준)와 상층부(응용프로그램 수준)로 구성되고 하위부에서는 가장법과 터널링의 방법이 사용되었다. 우선 IP 가장법은 여러 컴퓨터들을 인터넷상에서 출입로(gateway) 역할을 하는 오직 하나의 컴퓨터 뒤에 숨겨서 동작시키는 기능이다. 터널링 기법은 외부에 공개된 공적인 IP 주소를 가진 IP 패킷 헤더에 새로이 내부 각 서버의 개인 IP 주소를 추가하는 encapsulation, 또 그 역과정인 decapsulation을 행한다. 이를 통하여 패킷을 받은 서버들이 다시 주소를 변환하지 않고도 원래 IP 주소를 가진 헤더 정보를 이용해 직접 인터넷상으로 데이터를 전송한다. 이를 개념적으로 나타낸 것이 그림 2, 3에 있다.

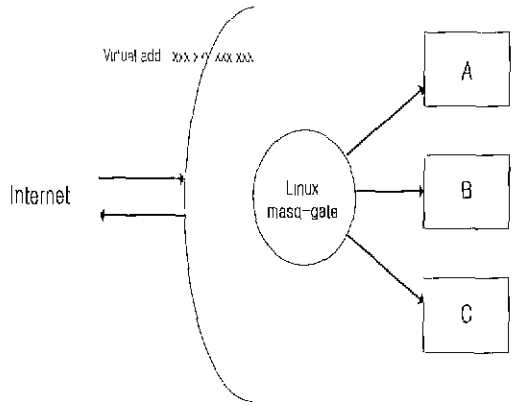

그림 2 가장법(Masquerading)

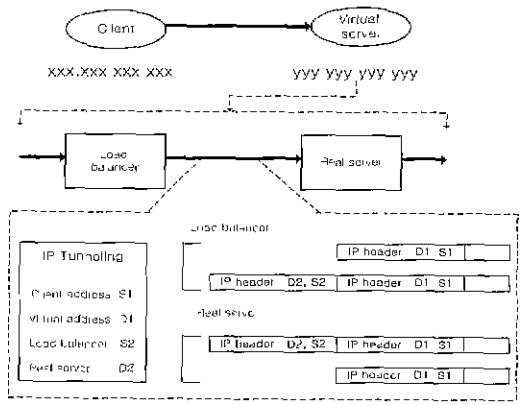

그림 3 IP 터널링(IP Tunneling)

그림 2에서 xxx.xxx.xxx.xxx라는 공적인 IP 주소를 가지고 외부에서 접근하게되면 내부에서는 masq-gate가 A, B, C의 접속을 변환해서 패킷을 전달함으로써 내부 네트워크상의 컴퓨터들은 인터넷으로 통할 수 있으나 클라이언트는 내부 서버들에 대한 접속정보는 알 수 없게 된다. 그림 3은 클러스터형 구조의 서버가 source IP xxx.xxx.xxx.xxx와 destination IP yyy.yyy.yyy.yyy를 터널링을 이용해 헤더를 덧붙여 실행서버에게 전달함으로써 실행서버가 직접 응답하는 과정을 보여주고 있다.

상층부에서는 각 서버의 부하 정보를 비교하기 위한 load monitor가 존재하며 이는 크게 세 부분으로 구성된다. 첫 단계는 각 실행 서버의 부하 정보를 요구하고 이를 load monitor로 수집

하는 단계이고 두 번째는 수집된 부하 정보를 비교하여 가장 작은 부하를 택하고 해당 서버의 IP 주소를 선택하는 단계이다. 마지막으로 이 선택된 주소를 이용해 입력된 기존 서버의 주소를 수정하는 단계이다. 이후에 클라이언트로부터 입력된 패킷은 수정된 IP 주소를 갖는 서버로 전달된다.

4.1.2 실행 서버(real server) 설계

실행 서버에서는 자신의 부하 정보를 조사하여 특정 포트로 부하 제어기에게 전달하는 역할과 부하 제어기를 통해 전달된 패킷을 받아 요청된 서비스를 처리하여 직접 클라이언트에게 전달하는 역할을 한다(그림 4 참조).

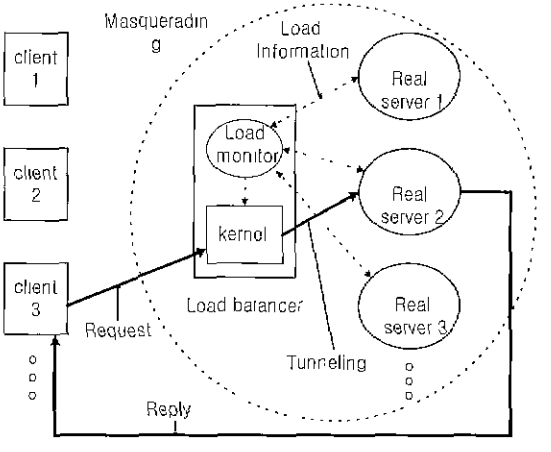

그림 4 load balancer와 real server로 이루어진 클러스터형 웹 서버

동적 부하 분산법을 이용한 클러스터형 웹 서버의 장점은 가장법을 이용함으로써 1개의 공적인 IP 주소로 서비스가 가능하게 하여 실행 서버의 교체, 확장, 그리고 변경이 용이하다. 또한 터널링을 이용하여 부하 제어기의 병목 현상을 제거하며 실행 서버와 클라이언트가 직접 통신함으로써 데이터 전송속도가 빠르다 또 load monitor가 일정한 주기마다 서버의 리스트를 갱신하기 때문에 실행 서버의 부하 정보를 바탕으로 스케줄링하여 더 정확하게 부하를 분산시킬 수 있다.

단점으로는 부하 제어기뿐만 아니라 실행 서버에도 프로그램을 설치해야 한다. 그리고 서버수가 증가하면 모든 서버의 부하 정보를 수집하는데 걸리는 시간이 증가하고 그 사이에 부하의 변동이 있어 오차가 발생할 수 있다. 하지만 적정한 개수의 노드로 된 클러스터인 경우 내부 네트워크를 통해 부하를 전달하는 시간이 매우 짧고 실행 서버의 부하의 측정 오차가 크지 않아 크게 문제되지 않는다.

4.1.3 성능 실험 및 평가

실험용 클러스터는 16대의 펜티엄2 - 300MHz의 PC를 100Mbps의 이더넷 스위치로 연결하였고, 리눅스 운영체제상에서 아파치를 웹 데몬으로 사용하였다. 모든 실행 서버는 동일한 컨텐트(content)를 가지고 있다. 실험에서는 크기가 다른 HTML 파일을 인위적으로 생성하여 서비스 대상으로 하였고 실행 서버수를 2대, 4대, 8대로 증가시키면서 서버의 성능을 측정하고, 동적 부하 분산 방법과 기존의 스케줄링 방법을 각각 적용하였고 서비스 요청 파일은 크기를 불규칙(random)하게 만들어 실험하였다.

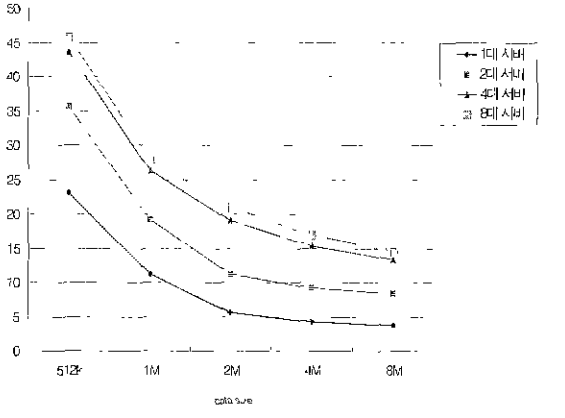

그림 5 동적 부하 분산법에 의한 성능실험 (0.5M ~8M)

그림 5는 데이터의 크기를 0.5MB에서 8MB까지 같은 파일 크기의 서비스를 요청하는 경우 실행 서버수를 2대, 4대, 8대로 했을 때, 클러스터형 웹 서버에서의 초당 처리 가능한 요청(request)수를 측정한 것이다. 클러스터를 썼을 때 1대의 서버보다 1.5 ~ 4.9 배의 성능 향상을

보여준다. 여기서 서버수를 증가시킴에 따라 비례해서 서버의 성능이 나타나지 않는 것은 하드디스크의 접근속도가 느려 서버수를 증가하여도 이에 비례해서 성능이 나타나지 않는 것으로 추정된다.

데이터를 불규칙하게 요구할 경우에도 나머지 두가지 방법에 비해 1.2 ~ 1.6배의 성능 향상이 있었다. 이는 일정 주기로 각 실행 서버의 실제 부하를 측정해서 부하 분산을 정밀하게 행하기 때문에 그렇지 못한 다른 스케줄링 알고리즘에 비해 성능이 향상된 것으로 보인다.

4.2 분산 파일 서버에 기반한 웹 서버

클러스터형 웹 서버가 제공하는 정보, 즉 컨텐트(content)는 일반적으로 하나의 공유된 파일 서버를 통해 제공되거나 실제 웹 서비스를 해주는 각각의 실행서버들에 복사되어 제공된다. 공유 파일 서버로 구성된 웹 서버는 클라이언트의 요청이 많아질수록 병목현상(bottle-neck)이 발생할 확률이 높아져 전체적인 시스템의 성능이 떨어질 수 있으며, 동일 파일 서버로 구성된 웹 서버는 응답속도를 향상시키기 위해 데이터 캐싱(caching)을 각 노드가 행하는 경우 동적 웹 페이지 작성, 데이터 베이스 변경 등으로 인해 전체 컨텐트의 일관성(consistency)이 깨지는 문제가 발생할 수 있다. 이것들을 해결하기 위해 분산 파일 서버로 구성된 웹 서버를 구축한다. 하나의 파일 서버에 저장될 컨텐트를 정해진 인덱스(index)에 맞춰서 각각의 실행서버에 나누어서 저장하는 구조의 웹 서버를 말한다 전체적으로 하나의 파일 서버의 저장 용량으로만 서비스를 해주므로 저가로 구현되고 동적 웹 페이지의 생성이나 데이터 베이스의 추가·변경시에도 해당 파일은 주어진 index 그룹에서만 존재하므로 일관성 유지가 용이하며, 인덱스와 파일 서버를 추가해 쉽게 저장공간을 늘릴 수 있다. 전체 시스템의 성능 향상과 신뢰성 향상을 위해서는 여러 대의 실행서버를 하나의 인덱스 그룹에 할당하도록 구현한다. 이 구조는 사용자들의 요청이 분산된 컨텐트 그룹 중에 일부에 집중될 때 전체 시스템의 성능이 매우 나빠질 수 있다는 점이다. 이런 그룹간의 부하의 불균형은 서버의 이주(migration)를 통해 해결하였다. 다음은 웹 서버의 설계 과정과 실험 결과이다

4.2.1 웹 서버의 구조

웹 서버는 dispatcher와 부하관리노드의 두 부분으로 구성된다. 사용자의 요청을 실행 서버로 전달해주는 dispatcher는 우선 요청이 어느 인덱스 그룹에 속해 있는지 분석을 하며, 선택된 그룹내에서 라운드 르빈(round-robin) 방식으로 서비스를 해줄 실행 서버를 고르게 된다. 실험 환경에서 인덱스를 웹 서버의 디렉토리로 했을 경우 dispatcher가 사용자의 요청을 분석하는 데 걸리는 시간은 평균 94μsec로서 웹 서비스 시간을 고려한다면 무시할 만하다. 그 후 dispatcher는 HTTP redirection을 통해 요청을 선택된 실행서버로 분배시키게 된다. HTTP redirection이란 HTTP protocol 자체에서 지원해주는 기능으로 응답 메시지의 헤더에 redirection됨을 나타내주는 상태 코드(status code) 307(HTTP 1.1[4], HTTP 1.0에서는 302[5])과 서비스를 해줄 서버의 URL을 적어서 다시 사용자측의 웹 브라우저에게 보내주고, 사용자측의 웹 브라우저는 새로운 URL로 다시 요청하게 되는 기능이다.

다음은 사용자들의 요청이 하나의 인덱스 그룹에만 집중됨으로써 그룹 사이의 부하의 불균형이 생기는 것을 피하도록 부하 정보를 수집하는 부하 관리 노드(load management node)와 이주를 실현하는 부분이다. 특정한 인덱스 그룹으로 사용자의 요청이 집중되면 기존의 인덱스 그룹을 서비스해주는 실행 서버의 개수로는 처리하기 부족할 것이다. 이것을 해결하기 위해서 다른 그룹내의 실행 서버들 중 일부를 과부하가 걸린 그룹으로 이주시킨다. 부하 관리 노드는 각 인덱스 그룹내의 실행 서버들의 부하의 양을 CPU와 네트웍, 하드디스크 등 여러 자원 등을 고려하여 측정해 각 그룹의 부하 평균들이 임계 조건을 만족했을 때 이주를 수행시킨다. 그림 6은 부하 관리 노드에 의해 수행되어지는 각 실행 서버들의 부하량 검사와 이주 과정의 모습을 보여준다.

여기서 임계 조건은 최소 부하 그룹의 1.5배가 넘는 상황을 썼으나 1.5란 값은 경험적으로 선정한 값이다.

이주 과정은 dispatcher쪽과 선택된 실행 서버의 파일 서버쪽에서 진행된다. 이주된 서버는 새

로운 인덱스 그룹에 속하게 되는 것이므로 dispatcher에서 이 서버로 사용자의 요청이 분배되도록 dispatching 테이블내의 서버의 위치를 변경한다. 마찬가지로 이주된 서버도 새로운 인덱스 그룹에서 해당하는 컨텐트를 서비스할 수 있도록 해당 파일 서버를 마운트한다 자신의 파일 시스템으로 복사하는 방법은 과부하 상태에 있는 서버들의 부하를 더 가중시키게 되므로 쓰지 않았다.

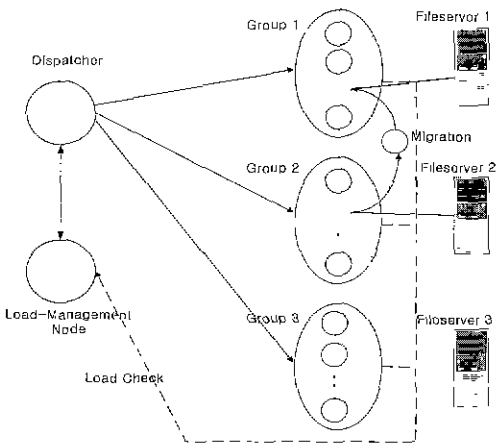

그림 6 부하 관리 노드와 이주(migration) 과정

4.2.2 실험 결과 및 검토

앞에서 언급한 클러스터에서 14대의 PC를 써서 실험하였다. 이중 1대는 dispatcher 노드가 되며, 다른 1대는 부하 관리 노드, 그리고 8대는 실제 서비스를 해주는 실행 서버 노드로, 나머지 4대는 파일 서버로 사용하였다. 각 PC에는 64MB의 RAM과 4.3GB의 EIDE방식의 하드디스크가 장착되어 있고, 설치된 실제 웹 서버 프로그램은 아파치(Apache) 버전 1.2.6으로 성능 향상을 위한 특별한 조율(tuning)을 하지는 않았다. 인덱스는 웹 서버의 디렉토리명으로 구축하였다.

웹 서버의 성능 평가 지수는 초당 처리 가능한 request의 수로 평가하였고 실험 중에 처리된 request의 총 개수를 전체 실험 시간으로 나눈 값으로 계산된다. 실험을 통해 웹 서버의 성능을 평가하기 위해서는 실제 웹의 작업 부하 특성을 파악하고 이에 맞는 부하 생성기(load-generator), 즉 웹 클라이언트에서 웹 서버로의 접속을 주어진 시간 안에 반복적으로 빨리 만들어주는 프로그램이 필요하다. 본 실험에서는 ACME Labs. Software에서 만든 http_load[7]를 HTTP redirection이 가능하도록 수정하여 한번에 수백개의 클라이언트 프로세스를 생성할 수 있도록 만들었다. 실험에 사용한 작업 부하(work-load)는 평균 21K 바이트 정도로 분포시킨 단순한 텍스트 기반의 정적 html 문서와 C 언어로 만든 간단한 CGI 동적 문서이다 HTTP 프로토콜을 수행하는데 있어서 서버에서는 단지 GET operation만을 사용하는 것을 원칙으로 했다.

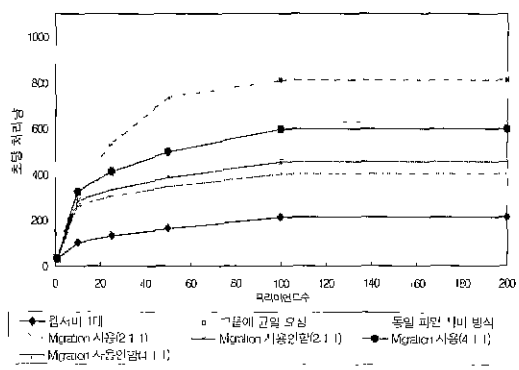

그림 7 정적 웹 페이지에 대한 성능(실행 서버 6대, 그룹 3개)

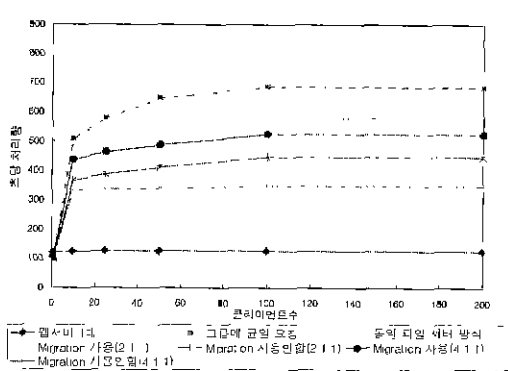

그림 8 동적 웹 페이지에 대한 성능(실행 서버 6대, 그룹 3개)

실험은 사용자의 요청이 인덱스 그룹들 사이에

균일하게 분포되는 경우와 불균일하게 분포되는 경우로 나누어서 수행되었으며, 이 때 이주를 사용해 성능이 얼마나 향상되는지 알아보았다. 그림 7, 8은 인덱스 그룹의 수를 3개로 하고 6대의 실행서버를 사용하였을 때 정적 웹 페이지와 동적 웹 페이지(CGI)의 경우에 대한 실험 결과이다.

실험 결과에서 동일 파일 서버 방식의 값은 그 조건하에 얻을 수 있는 최대 성능치, 즉 상한값(upper bound)으로 사용된다. 이것은 분산 파일 서버 방식의 경우 dispatcher가 사용자의 요청을 분석해야하는 오버 헤드 등을 더 포함하기 때문이다. 전체 실험 결과에서 볼 수 있듯이 사용자의 요청이 각 그룹에 균일하게 요청됐을 경우의 값과 크게 차이나지 않았다. 그림 7에서 괄호내에 표시된 값은 각 그룹에 요청된 비율로서 2:1:1이라면 3개의 그룹 중 하나에 다른 것들보다 2배 더 요청됨을 뜻한다. 클라이언트의 수가 증가할수록 초당 처리량은 증가하다가 프로세스의 개수가 약 100개 정도가 되었을 때 포화되는 현상을 보여주는데, 이것은 메모리나 프로세스 등의 자원이 고갈되고 있기 때문이다

이주를 사용함으로서 확실히 성능이 향상됨을 확인했다. 특히 한쪽 그룹으로의 과부하가 심할 경우 성능 향상이 더 뚜렷했다 정적 웹 페이지에 대한 실험에서 2:1:1 분포보다는 4:1:1 분포에서 성능향상은 38%에서 49%로 증가하였다(그림 7). CGI에 대한 실험에서도 마찬가지로 2:1:1 분포시 29%의 성능 향상이 4:1:1의 분포시 50%의 성능향상이 있었다(그림 8)

5. 결 론

PC 클러스터를 활용해 웹 서버를 만드는 과정을 소개하였다. 클러스터형 웹 서버는 저비용으로 사용자에게 신뢰성 있고 안정된 서비스를 제공해준다는 점에서 이미 많은 연구가 진행되고 있고 특히 클라이언트의 요청을 내부의 모든 실행 서버에게 적절히 분배하여 서버의 부하를 균등히 하는 방안이 중요시되고 있다. 본 연구에서는 모든 실행 서버의 부하를 동적으로 모니터하여 패킷을 분산시키는 방법을 적용한 클러스터형 웹 서버와 서버마다 서로 다른 컨텐트를 갖는 분산된 웹 서버에서 부하관리노드를 이용해 이주를 하는 분산 파일서버로 구성된 웹 서버를 설계, 구현하여 그 성능이 향상됨을 확인하였다. 향후 서버 규모를 늘릴 때 더 큰 성능의 향상을 얻도록 개선되어야 하며, 간단한 인위적 데이터가 아닌 실제의 웹 서버로 동작시켜 클러스터형 웹 서버의 성능향상과 부하 균등화를 완성하는 연구를 계획하고 있다.

참고문헌

[1] Daniel Anderson, et al, Towards a Scalable Distributed WWW Server on Workstation Clusters, Proc. of 10th IEEE Intl. Symp. of Parallel Processing IPPS' 96, pp. 850-856. April, 1996.

[2] Eric Anderson, et al, The Magic router : an Application of Fast Interposing, http://www.cs.berkeley.edu/~eanders/magicrouter/, May 1996.

[3] Michele Colajanni, et al, Dynamic Load Balancing on Web Servers systems, http://computer.org, May 1999.

[4] Fielding, et al, Hypertext Transfer Protocol-HTTP/1.1,http://www.w3.org/hypertext/WWW/Protocols/, June 1999.

[5] Berners-Lee, et al, Hypertext Transfer Protocol-HTTP/1.0,http://www.w3.org/hyper text/WWW/Protocols/, May 1996.

[6] Wensong Zhang, et al, Creating Linux Virtual Servers, Linux Conference, http://proxy.iinchina.net/~wensong, 1999.

[7] ACME Labs Software, http_load-multiprocessing http test client, http://www.acme.com/software/http_load/

[8] Beowulf Project at CESDIS, http://beowulf.gsfc.nasa.gov

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

[10] LAM / MPI Parallel Computing, http://www.mpi.nd.edu/lam/

[11] Top500 supercomputers list, http://www.top500.org/top500.list.html

[12] The Message Passing Interface (MPI),