중소형 사이트를 위한 다수의 전면 서버를 갖는 고가용성 웹 서버 클러스터

문 종 배[†] · 김 명 호[††]

요 약

인터넷이 새로운 산업의 형태로 발전하면서 웹 사이트에 클러스터링 기술을 많이 적용하고 있다. 고성능 하드웨어 스위치를 사용하면 성능이 우수하지만 중소형 사이트를 구성하는 데에는 비용 부담이 많이 된다. 최근에는 무료이면서 성능이 뛰어난 리눅스 가상 서버(Linux Virtual Server)를 이용한 클러스터를 많이 구축하고 있다. 그러나 리눅스 가상 서버는 하나의 전면서버를 가지는 중앙집중식 부하분배 구조이므로 사용자 요청의 급격한 증가로 병목현상을 일으킬 수 있다. 따라서 본 논문에서는 다수의 전면서버를 두어 리눅스 가상 서버의 병목현상을 없애는 방법을 제안한다. 모든 클러스터 노드들은 전면서버와 후면서버의 역할을 모두 할 수 있도록 구성한다. 서버의 부하가 많지 않으면 자기 노드에서 직접 처리하여 사용자에게 응답할 수 있도록 하고, 부하가 많은 경우 부하가 가장 작은 노드를 선택하여 사용자 요청을 전달한다. 그리고 클러스터 서버들의 부하를 고려한 스케줄링 알고리즘을 제안하여 서버들 사이의 부하를 고르게 한다. 실험결과 기존 웹 서버 클러스터들은 서버의 수가 증가하면서 성능향상 폭이 감소하지만, 본 논문에서 제안한 웹 서버 클러스터는 선형적으로 증가하는 것을 보였다. 리눅스 가상 서버 환경에서 다수의 전면서버를 구성함으로써 병목현상을 없애고, 비용 부담이 없고 고성능의 중소형 사이트를 구축할 수 있다.

High-Availability Web Server Cluster Employing Multiple Front-Ends for Small and Middle-sized Web Sites

Jongbae Moon[†] · Myungho Kim[††]

ABSTRACT

These days, various clustering technologies have been adopted to construct web sites. High performance hardware switches have good performance, but have disadvantage of high cost for constructing small and middle-sized web sites. Now a days, many sites have been constructed with the LVS (Linux Virtual Server), which is free of charge and has good performance. Having a centralized load balancing with one front-end, the LVS causes a bottleneck when it receives all at once. In the paper, we suggest a way to remove the LVS bottleneck by providing multiple front-ends. In this architecture, all of cluster nodes act as both a front-end and a back-end. When the load of a node receiving requests is not large enough, the node responds to the client directly. When the load of a node is large enough, the node send the request to a node which is selected by a scheduling algorithm. The scheduling algorithm is discussed to balance loads between servers. While single front-end cluster raises the throughput curvedly, the multiple front-end cluster raises the throughput linearly.

키워드 : 고가용성(High Availability), 클러스터(Cluster), 스케줄링(Scheduling), 리눅스 가상 서버(Linux Virtual Server), 부하분배(Load Balancing)

1. 서 론

최근 인터넷 사용이 일반화되면서 인터넷 사용자의 수가 매년 증가하고, 인터넷은 새로운 산업의 형태로 발전하고 있다. 전자 상거래는 그 좋은 예가 될 수 있다. 현재 전자 상거래와 같은 서비스를 제공하는 사이트가 점차 늘어나고, 이 가운데 하루 평균 백만 건 이상의 접속이 이루어지고 있는 사이트도 생겨나고 있다. 이러한 사이트에서는 사용자의 요청을 처리하기 위해 많은 컴퓨팅 파워를 요구하고 있다. 또한 이러한 사이트는 시스템이 다운되었을 경우 엄청난 경제적 손실을 입게 된다. 따라서 시스템의 다운 타임 최소화도 요구하고 있다. 이러한 측면에서 고성능과 고가용성을 제공하는 웹 서비스의 필요성이 대두되었다.

단일 시스템으로 구성된 기존의 사이트들은 단일 시스템의 한계 때문에 고성능과 고가용성의 서비스를 제공하는 것이 이미 한계에 도달하였다[1]. 단일 시스템의 한계를 극복하고 더욱 강력한 컴퓨팅 파워와 시스템의 안정적 서비스를

※ 본 논문은 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2004년 3월 2일, 심사완료 : 2004년 9월 8일

제공하기 위해 클러스터링 기술이 웹사이트에 적용되고 있다[2]. 클러스터 시스템이란 고성능 시스템에 비해 성능은 뒤지지만 가격이 저렴한 시스템 여러 대를 하나의 시스템으로 묶어서 가격대 성능비를 향상시킨 시스템을 말한다.

웹 서버 클러스터는 사용자의 서비스 요청을 여러 대의 클러스터에 분배시켜주는 기능을 가지는 시스템이다. 고성능 웹 서버 클러스터 시스템을 구성하기 위해 다양한 연구가 진행되고 있다[3, 4, 7, 10-13]. 이러한 웹 서버 클러스터의 구성 방법은 크게 세 가지로 나눌 수 있다. 첫째는 L4 스위치와 같은 하드웨어 장비를 사용하는 방법이다. 이 방법은 성능과 신뢰성이 좋지만 부가적인 비용이 너무 비싸다는 단점이 있다. 둘째는 DNS(Domain Name System)를 이용하는 방법이다. DNS를 이용한 방법은 구성하기 쉽고 확장성은 좋지만 서버들의 상태를 고려하지 않기 때문에 가용성이 문제가 된다. 세 번째는 소프트웨어적으로 부하를 분산하는 디스패처 방식이다. 이 방식을 사용하는 경우에는 하드웨어 장비보다 가격이 저렴하지만 디스패처의 안정성이 큰 문제가 된다.

최근에는 무료로 사용할 수 있으면서 고성능과 가용성을 보장하는 리눅스 가상 서버(LVS, Linux Virtual Server)[5]를 사용하여 웹 서버 클러스터를 구성하는 사례가 늘고 있다. 리눅스 가상 서버는 하드웨어 L4 스위치의 역할을 값싼 서버 한 대에서 그 역할을 대신할 수 있도록 하는 소프트웨어이다. 중앙 집중형 구조인 리눅스 가상 서버는 사용자의 요청이 급격히 증가하여 전면 서버에 병목현상을 불러일으키는 상황이 발생할 수 있다. 이렇게 전면 서버가 작동을 하지 않는다면 다수의 후면 서버가 있음에도 불구하고 서비스가 전혀 이루어 질 수 없다. 전면 서버의 장애로 서비스가 지속되지 못 할 경우를 위하여 백업 서버를 두어 서비스를 계속 할 수는 있지만 병목현상에 대한 근본적인 해결책이 되지 못하고 추가적인 비용부담이 발생한다.

본 논문에서는 리눅스 가상 서버를 기반으로 추가적인 비용이 없이 중소형 사이트를 구축할 수 있도록 웹 서버 클러스터 시스템을 설계하고 구현한다. 기존 리눅스 가상 서버의 단점인 병목현상을 제거하기 위하여 다수의 전면 서버를 두어 사용자의 요청을 처리하도록 한다. 또한, 실제 서비스를 하는 후면 서버들의 부하량을 고려하여 스케줄링 할 수 있도록 새로운 스케줄링 알고리즘을 제시한다. 이 알고리즘은 후면 서버들의 부하가 한 쪽으로 집중되는 것을 막아 부하의 균형을 유지하도록 한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 여러 형태의 웹 서버 클러스터 구성에 대하여 살펴보고, 제 3장에서는 본 논문에서 제안하는 새로운 웹 서버 클러스터와 스케줄링 알고리즘에 대해 설명하고, 제 4장에서는 다양한 형태의 웹 서버 클러스터 시스템들과의 성능을 실험을 통하여 비교 분석한다. 끝으로 제 5장에서는 결론 및 앞으로 진행될 연구의 내용에 대해 설명한다.

2. 관련 연구

웹 서버 클러스터는 여러 대의 서버 노드로 클러스터를 구성하여 하나의 웹 사이트 서비스를 제공할 수 있도록 구성한 시스템이다. 웹 서버 클러스터는 사용자의 요청을 어떻게 분산시키는지에 따라 크게 4가지 분류로 나눌 수 있다[4]. 본 장에서는 웹 서버 클러스터의 4가지 분류와 최근 많이 사용되고 있는 리눅스 가상 서버를 이용한 구성 방법에 대하여 살펴보고, 기존 리눅스 가상 서버 방식의 문제점을 살펴본다.

2.1 클라이언트 기반 웹 서버 클러스터

클라이언트 기반 웹 서버 클러스터는 웹 클라이언트가 클러스터로 구성된 모든 서버들을 알고 있다고 가정한다. 웹 클라이언트는 자바 애플릿(Java Applet)을 통하여 각 서버들의 상태 정보를 수집하고 이 정보를 기반으로 하여 서버를 선택하여 그 서버로 웹 클라이언트의 요청을 전달하는 방법이다[4]. 버클리 대학의 Smart Client에서 이 방식을 사용하였다[6]. 이 방법은 서버 노드들의 상태를 관찰하기 위하여 필요한 메시지의 교환이 계속되기 때문에 네트워크의 부하를 초래한다. 또한 서버측의 확장성과 가용성은 제공하지만 클라이언트 측면에서는 모든 어플리케이션 부분을 수정해주어야 하는 단점이 있다.

2.2 DNS 기반 웹 서버 클러스터

클라이언트 기반 웹 서버 클러스터의 단점을 없앤 분산 웹 서버 구조로 DNS(Domain Name Service)에서 사용자에게 서버들의 단일 가상 인터페이스를 제공하여 부하를 분산하는 웹 서버 클러스터 구조이다. 이 형태의 대표적인 방식으로는 라운드 로빈 DNS 방식이 있다. NCSA에서 처음 사용한 이 방식은 DNS에 분산된 서버들의 IP 주소를 라운드 로빈 방식을 적용하여 순차적으로 대응시켜 서로 다른 서버를 선택하는 방법이다[7, 8]. 이 방법을 이용하여 간단하게 부하를 분산시킬 수 있다. 그러나 라운드 로빈 DNS는 클라이언트 캐싱(caching)에 의해 특정 서버에 사용자의 요청이 많이 할당되어 부하의 불균형을 일으킬 수 있다. 그리고 서버들의 상태를 알 수 없기 때문에 장애가 발생한 서버에도 사용자 요청을 할당할 수 있다[4].

2.3 서버 기반 웹 서버 클러스터

서버 기반의 웹 서버 클러스터는 일반적으로 다음과 같은 2단계 분산 메커니즘을 가지고 있다. 우선 사용자의 요청은 웹 서버 시스템의 DNS에서 클러스터의 한 서버에 할당되고, 그 다음 사용자의 요청을 받은 서버는 다시 다른 서버에

게 사용자의 요청을 다시 요청하게 된다. 이 방법에서 사용자의 요청을 처음 받은 서버는 다시 다른 서버로 사용자의 요청을 분산시키기 위하여 HTTP-Redirection과 같은 응용 프로그램 수준 스케줄링 방법을 이용한다. 이 방법을 사용하는 예로는 Scalable Server World Wide Web(SWEB)이 있다[9]. 이 방법의 단점은 2단계의 분산 메커니즘을 사용함으로써 처리하는데 시간이 많이 걸리고, 응용프로그램 수준에서 HTTP 요청과 응답을 처리하는 데 많은 네트워크 오버헤드가 발생한다.

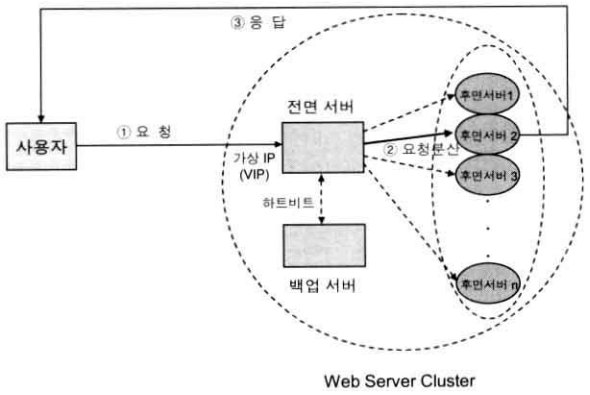
2.4 디스패쳐 기반 웹 서버 클러스터

사용자의 요청을 후면에 있는 웹 서버에 보내주는 디스패쳐를 통해 스케줄링하는 방식이다. 이 방법은 사용자에게는 하나의 가상 IP만을 보여줌으로써 웹 서버 클러스터 시스템에 대한 투명성을 제공한다. 이 방법은 다시 디스패쳐가 사용자의 요청을 처리하는 방법에 따라 패킷 변환(Packet Re-writing), 패킷 포워딩(Packet Forwarding) 방식으로 나누어진다. 패킷 변환 방식은 디스패쳐가 사용자 요청 패킷의 IP 주소를 변환하여 클러스터 서버중의 선택된 한 서버로 전달하고 처리된 응답을 받은 후 다시 사용자에게 되돌리는 방식이다. Magicrouter[10]와 Cisco의 LocalDirector가 대표적인 예이다[11]. 이 방법의 단점은 패킷의 변환에 많은 오버헤드가 발생한다는 단점이 있다. 패킷 포워딩 방식은 IP 주소를 변환하는 대신 사용자의 요청 패킷을 전달하는 방식을 사용한다. IBM의 Network Dispatcher[12]와 ONE-IP가 대표적인 예이다[13].

2.5 리눅스 가상 서버를 이용한 웹 서버 클러스터

리눅스 가상 서버를 이용한 웹 서버 클러스터는 디스패쳐 기반 웹 서버 클러스터의 일종으로 최근 웹 서버 클러스터의 고성능과 투명성, 확장성, 안정성을 확보하기 위하여 많이 사용되고 있다. 이 방법에서 클러스터를 구성하는 서버들은 사용자의 요청을 받아들여 분배하는 전면 서버 부분과 사용자 요청을 실질적으로 처리하는 후면 서버 부분으로 구분되는 것이 일반적이다. 가상 서버를 기반으로 한 웹 서버 클러스터 시스템의 전체 구성 모습은 다음 (그림 1)과 같다.

가상 서버는 디스패쳐 방식의 단점인 패킷의 변환에 따른 오버헤드를 해결하기 위하여 후면 서버가 처리한 응답을 전면 서버를 거치지 않고 사용자에게 직접 보낼 수 있도록 되어있다. 기존의 가상 서버를 이용한 일반적인 웹 서버 클러스터 시스템은 하나의 전면 서버와 다수의 후면 서버로 구성되어 있거나, 이러한 형태의 확장이기 때문에 전면 서버에 지나치게 많은 요청이 발생하여 전면 서버의 처리능력을 넘어서 병목현상이 발생 할 수 있다. 또한 여러 서버들이 처리하는 양은 불규칙하고, 서버들의 시스템 부하량은 서버들마다 모두 다르기 때문에 부하량이 많은 서버로 사용자의 요청이 계속 할당되면 서버가 처리하는 속도가 느려지게 되어 사용자에게 만족할 만한 응답시간을 보장하지 못하게 된다. 리눅스 가상 서버의 경우 기본적으로 제공하는 스케줄링 알고리즘들은 서버들의 부하량을 고려하지 않는 방법이므로 부하의 불균형을 초래할 수 있다.



(그림 1) 가상 서버를 이용한 웹 서버 클러스터의 구조

2.6 웹 서버 클러스터의 비교

앞에서 다양한 형태의 웹 서버 클러스터 구성방법에 대하여 살펴보았다. 각각의 웹 서버 클러스터의 특징을 비교해 보면 다음 <표 1>과 같다.

〈표 1〉 다양한 웹 서버 클러스터의 비교

분 류	부하분배	장 점	단 점
클라이언트 기반	분산방식	◦ 서버의 부하 감소	◦ 투명성 없음 ◦ 네트워크 부하 증가
DNS 기반	중앙집중식	◦ 병목현상 없음 ◦ 근거리(LAN)뿐만 아니라 원거리(WAN) 부하분산 가능	◦ 서버들의 부하 불균형
서버 기반	분산방식	◦ 병목현상 없음	◦ 응답시간 증가 ◦ 패킷 변환 오버헤드 발생
디스패쳐 기반	중앙집중식	◦ 고른 부하분산	◦ 디스패쳐 병목현상
리눅스 가상 서버 기반	중앙집중식 (IP 수준 스케줄링)	◦ 뛰어난 성능, 확장성	◦ 전면서버 병목현상

클라이언트 기반 웹 서버 클러스터 형태는 사용자의 요청을 받아서 처리하는 부분이 따로 있지 않고 각 서버들이 직접 사용자 요청을 받아서 처리하기 때문에 사용자에게는 클러스터 시스템이 하나의 서버처럼 보이지 않는다. 각 서버 노드들의 상태를 관찰하기 위한 네트워크의 부하량이 많고 클라이언트 측면에서는 응용프로그램을 수정해야 하기 때문에 더 이상 연구되고 있지 않다.

DNS 기반 웹 서버 클러스터 형태는 단순하게 구성할 수 있고 확장성이 뛰어나다는 장점이 있지만 서버들의 상태를 알 수 없기 때문에 장애가 발생한 노드에 사용자의 요청을

계속 보낼 수 있다는 단점이 있다. 또한 클라이언트의 캐싱에 의해 특정 서버에 사용자의 요청이 많이 할당되어 서버들의 불균형을 초래할 수 있다.

서버 기반 웹 서버 클러스터 형태는 각 서버들이 부하 분산을 하기 때문에 한 노드가 장애로 서비스를 하지 못할 경우에도 전체 클러스터는 계속 서비스를 할 수 있다. 즉 SPOF(Single Point Of Failure)가 발생하지 않는 장점이 있다. 그러나 HTTP 재지향(Redirection)과 같은 응용프로그램 수준의 스케줄링 방법 사용함으로써 응답시간이 길어지게 되는 단점이 있다.

디스패쳐 기반 웹 서버 클러스터는 하나의 가상 IP(VIP)를 사용함으로써 사용자에게는 클러스터의 투명성을 쉽게 제공하고, DNS 기반 웹 서버 클러스터에서 발생했던 클라이언트 캐싱이 발생하지 않는 장점이 있다. 그러나 중앙 집중식의 디스패쳐는 사용자의 요청이 많아지면 병목현상이 발생하여 SPOF가 발생할 수 있다. 또한 패킷의 변환에 오버헤드가 발생한다는 단점이 있다.

리눅스 가상 서버를 이용한 웹 서버 클러스터는 디스패쳐 기반 웹 서버 클러스터의 일종으로 하나의 가상 IP를 사용함으로써 손쉬운 투명성을 제공하고, IP 수준의 스케줄링과 후면 서버가 직접 사용자에게 응답을 보냄으로써 빠른 응답시간을 보장할 수 있고, 후면 서버의 확장성을 손쉽게 보장하는 장점이 있다. 그러나 중앙집중식 부하 분배방법으로 병목현상과 SPOF을 피할 수 없다. 또한 후면 서버들의 부하량을 고려하지 않는 스케줄링 알고리즘을 사용하기 때문에 후면 서버들의 부하의 불균형을 초래할 수 있다. 리눅스 가상 서버는 무료 소프트웨어이면서 성능이 우수하기 때문에 중소형 사이트를 구성하는데 적합하다. 그러나 병목현상을 피할 수는 없기 때문에 병목현상을 없애고 좀 더 좋은 성능을 얻을 수 있는 방법을 연구할 필요가 있다.

3. 다수의 전면 서버를 갖는 고가용성 웹 서버 클러스터 시스템의 설계 및 구현

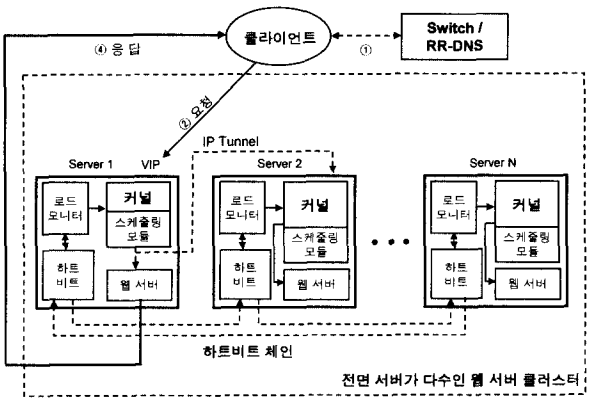
3.1 고가용성 웹 서버 클러스터 시스템을 위한 요구사항

본 논문에서는 중소형 사이트를 위한 고가용성 웹 서버 클러스터의 구현을 목표로 하고 있다. 이러한 목표를 위하여 기본적으로 다음과 같은 요구사항들을 만족해야 한다. 첫째, 비용부담이 없어야 한다. 둘째, 웹 서버 클러스터는 사용자의 요청에 빠른 응답시간을 보장할 수 있어야 한다. 넷째, 안정성을 위하여 전면 서버의 병목현상에 의한 SPOF가 없어야 한다. 다섯째, 장애가 발생한 서버에 대한 처리를 할 수 있어야 한다. 여섯째, 각 서버들의 균형적인 부하를 유지해야 한다.

본 논문에서는 비용부담을 줄이기 위하여 무료 배포되는 리눅스 운영체제와 아파치(Apache) 프로그램을 사용하여 서버를 구축한다. 사용자의 요청에 대한 빠른 응답시간을 보장하기 위하여 IP 수준의 스케줄링을 하는 리눅스 가상 서버를 기반으로 구현하도록 한다. 다수의 전면 서버를 두어 병목현상에 의한 SPOF를 제거하도록 한다. 특정 서버에 장애가 발생하여 서비스를 못할 경우에 다른 서버가 장애 서버의 역할을 대신 처리하도록 한다. 균형적인 부하 분배를 위하여 각 서버들의 부하량을 기반으로 한 스케줄링 알고리즘을 사용하도록 한다.

3.2 고가용성 웹 서버 클러스터 시스템의 전체 구조

본 논문에서 설계 및 구현한 고가용성 웹 서버 클러스터 시스템은 (그림 2)에서 보는 것처럼 전면 서버와 후면 서버의 역할을 동시에 하는 다수의 노드들로 구성되어 있다. 각 노드들은 리눅스 가상 서버 패키지, 하트비트 데몬, 로드 모니터, 부하량에 따른 부하분배 방식의 스케줄링 모듈 등의 컴포넌트들로 구성되어 있다. 각 모듈간의 통신이 서비스에 영향을 미치지 않도록 2개의 네트워크 카드를 이용한다.



(그림 2) 전면 서버와 후면 서버를 동시에 할 수 있는 고가용성 리눅스 가상 서버를 이용한 웹 서버 클러스터

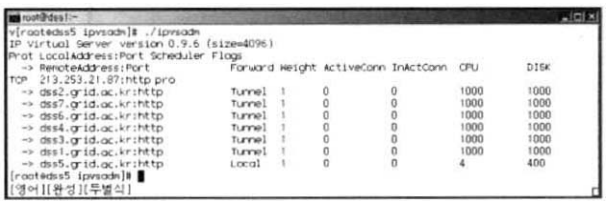
본 논문에서는 추가적인 비용 부담을 고려하여 라운드 로빈 DNS를 사용하여 고가용성 웹 서버 클러스터 시스템에 대한 투명성을 제공하도록 한다. 사용자의 요청은 일차적으로 라운드 로빈 DNS에 의해 선택된 클러스터의 임의의 서버에 전달된다. 사용자 요청을 받은 서버는 자기 노드의 부하량이 임계값보다 작으면 자기 노드에서 사용자 요청을 처리하여 응답을 직접 사용자에게 보내도록 한다. 그 값이 임계값보다 크면 로드 모니터와 스케줄링 모듈에 의해서 부하량이 가장 작은 노드를 선택하고, 사용자 요청을 IP 터널을 통해 선택된 서버로 사용자 요청을 전달한다.

3.2.1 다수의 전면 서버를 위한 리눅스 가상 서버 설정

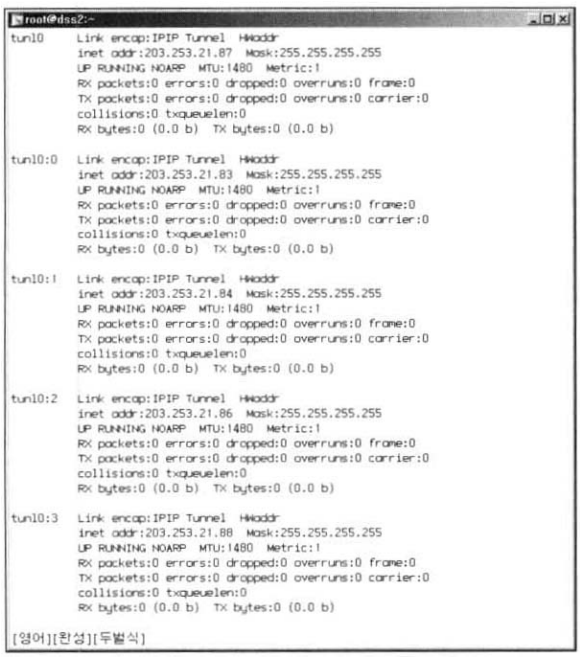
본 논문에서는 리눅스 가상 서버의 형태로 IP 터널(Tunnel) 방식을 사용한다. IP 터널 방식은 터널 장치를 이용하여 캡슐화(encapsulation)된 사용자 요청 패킷을 후면 서버에

전송하는 방식이다. 각 노드는 IP 터널을 사용할 수 있도록 커널 패치가 되어있어야 한다. 모든 노드들이 전면 서버의 역할을 하려면 터널 장치로 패킷을 전송할 수 있도록 서비스를 등록해야 한다. 리눅스 가상 서버에서 제공하는 ipvsadm 툴을 사용하여 서비스를 등록할 수 있다. (그림 3)은 한 노드에서의 서비스 설정을 보여준다. 가상 IP(VIP) 213.253.21.87의 80번 포트로 들어오는 사용자 요청을 pro 스케줄링 알고리즘에 의해 밑에 나열된 서버들로 분산한다는 설정이다. pro 스케줄링 알고리즘은 본 논문에서 제안한 부하기반 스케줄링 알고리즘이다. CPU와 DISK의 값은 각 노드의 부하량을 나타낸 것이다.

또한, 모든 노드가 후면 서버의 역할을 하려면 전면 서버와 연결된 터널 장치가 설정되어 있어야한다. 자기 노드를 제외한 모든 전면 서버에 대한 터널 장치를 등록해야 한다. 이것은 터널 장치의 앨리어스(alias)를 사용함으로써 가능하다. (그림 4)는 한 노드에서의 네트워크 설정을 나타낸 그림이다. 각 노드는 전면 서버를 위한 가상 IP와 모듈간의 통신을 위한 사설 IP를 가지고 있으며, 터널 장치의 앨리어스를 사용하여 각 전면 서버와의 IP 터널을 구성하도록 한다.



(그림 3) 각 노드들의 ipvsadm를 이용한 LVS 설정



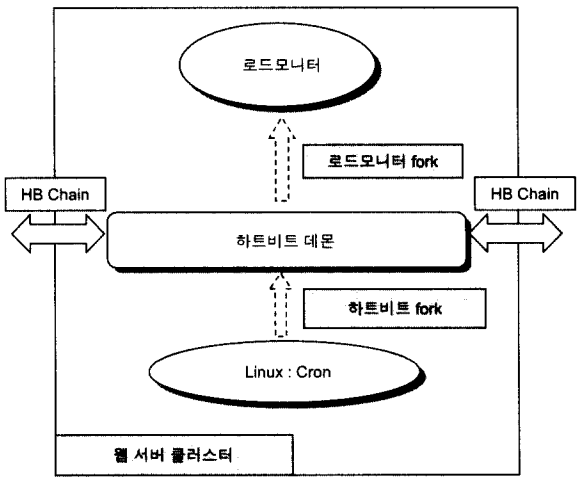
(그림 4) 노드의 네트워크 설정

3.2.2 하트비트 데몬

하트비트(Heartbeat) 데몬은 웹 서버 클러스터 시스템의 고가용성을 위한 컴포넌트이다. 하트비트 데몬은 웹 서버 클러스터 시스템에서 다음과 같은 중요한 두 가지 역할을 담당한다.

첫째는 결함이 발생했던 노드가 재실행 될 때 로드 모니터를 구동하는 것이다. 또한 하트비트 데몬은 주기적으로 자기 노드의 로드 모니터가 동작중인지 검사한다. 만약 로드 모니터 데몬이 동작하지 않는다면 하트비트가 로드 모니터를 실행시킨다. 하트비트가 자기 노드의 로드 모니터의 상태를 검사함으로써 컴포넌트에 대한 안정성을 제공한다. 하트비트의 안정성은 시스템에서 보장하도록 하였다. 리눅스 시스템 명령인 Cron을 사용 주기적으로 하트비트가 동작중인지 검사한다. 이때 하트비트가 동작하지 않는다면 Cron 데몬은 자동으로 하트비트를 수행시킨다.

둘째는 클러스터 시스템의 안정성과 고가용성을 위해 하트비트 체인을 구성하고 유지하는 것이다. 클러스터 시스템의 임의의 노드가 다운된 경우 결함노드는 하트 비트 체인에 의해서 발견된다. 결함된 노드가 수행 중이던 작업은 나머지 노드 중 임의의 노드에 의해서 대신 수행된다. 이와 같은 방식으로 하트비트 데몬은 한 노드의 구성 컴포넌트에 대한 안정성과 시스템의 안정성을 제공하고 있다. 하트비트 데몬과 웹 서버 클러스터 시스템의 안정성 유지 구조는 (그림 5)와 같다.

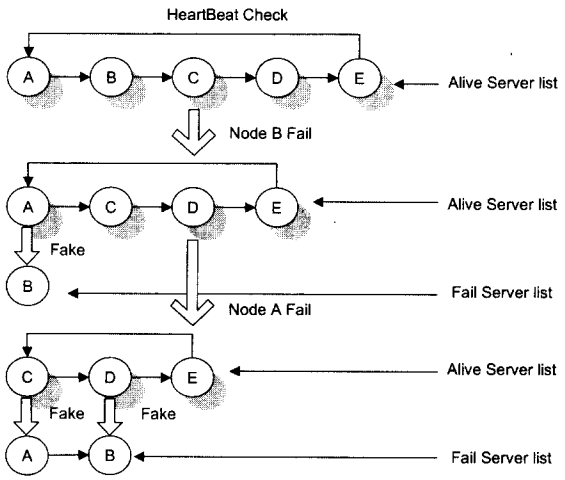


(그림 5) 웹 서버 클러스터 시스템의 고가용성

각 노드의 하트비트는 이웃노드의 하트비트를 주기적으로 검사한다. 하트비트는 단방향으로 이루어져 링(ring) 형태의 하트비트 체인이 만들어진다. 하트비트는 이웃 노드의 시스템 상태를 주기적으로 점검하게 된다. 결함을 발견한 노드의 하트비트는 클러스터의 다른 노드에게 결함노드에 대한 정보를 방송(broadcast)하게 된다. 또 하트비트는 하트비트 체인을 유지하는데 필요한 정보들, 예를 들어 장애 노드에

대한 정보 혹은 복구 노드에 대한 정보를 받아 하트비트 체인을 재구성한다.

클러스터를 구성한 노드 중 한 노드에 장애가 발생한 경우 임의의 한 노드가 결함 노드의 작업을 대신하게 된다. 본 논문에서는 결함노드의 작업을 대신하기 위한 방법으로 Fake를 사용한다. Fake는 결함 노드의 IP를 다른 노드가 대신하는 방식으로 ARP 프로토콜과 리눅스의 논리적 네트워크 인터페이스 지원을 이용하여 한 노드가 다른 노드의 IP를 마치 자신의 IP인 것처럼 속이는 것이다. Fake를 수행하기 위해서 각 노드는 하트비트 체인 정보이외에 서비스를 할 수 있는 서버들의 리스트인 alive_server 리스트와 현재 장애가 발생한 노드의 리스트인 fail_server 리스트를 유지한다. alive_server 리스트와 fail_server 리스트의 인덱스가 같은 노드에서 결함 노드의 Fake를 수행한다. 이 때 하트비트 체인과 alive_server, fail_server 리스트들은 분산된 각 노드간의 일관성을 유지하기 위해서 정렬된 상태로 구성되게 된다. 장애 노드에 대해 Fake 수행 구조는 (그림 6)과 같다. 이 때 Fake 수행 특성상 클러스터 노드가 n개일 경우 최소한 n/2개의 노드는 결함이 발생하지 않는다는 것을 전제로 한다.



(그림 6) 결함 노드에 대한 Fake 수행

3.2.3 로드 모니터

로드 모니터(Load Monitor)는 주기적으로 자기 자신의 부하 정보를 측정하고 이 값을 커널에 보내어 유지하고, 부하의 변화가 있으면 다른 노드와 자신 노드에게 부하 정보를 알린다. 부하의 변화가 크지 않을 경우에도 부하 정보를 교환하는 것은 오히려 로드 모니터간의 네트워크 오버헤드를 늘릴 수 있다. 따라서 로드 모니터는 임계값을 설정해서 부하의 변화량이 임계값을 초과하는 경우에만 자신의 부하 정보를 다른 노드들에 보내도록 하도록 한다.

로드 모니터는 부하량을 산출하기 위해 리눅스의 /proc/stat 파일에 저장되는 CPU와 디스크에 대한 정보를 사용한다. /proc/stat 파일은 리눅스 커널에서 유지하고 있는 여러 정보들의 통계치를 제공하는 파일이다. 산출된 부하량과 임계값을 커널에 저장하기 커널 내의 자료구조에 변수를 다음 (그림 7)과 같이 추가한다.

```
struct ip_vs_rule_user { /* 사용자 모드에서 커널의 자료구조를 제
어하기 위해 사용 */
                ...
  int    cpu ;          /* CPU 사용률을 위해 추가된 변수 */
  int    disk ;         /* DISK 사용률을 위해 추가된 변수 */
  int    threshold ;    /* DISK 사용률을 위해 추가된 변수 */
} ;
struct ip_vs_dest {     /* 실제 노드를 가리키는 자료구조이다 */
                ....
  int    cpu ;          /* CPU 사용률을 위해 추가된 변수 */
  int    disk ;         /* DISK 사용률을 위해 추가된 변수 */
  int    threshold ;    /* DISK 사용률을 위해 추가된 변수 */
} ;
```

(그림 7) 수정된 커널 내의 자료구조

3.2.4 서버들의 부하를 고려한 스케줄링 모듈

부하량을 기반으로 한 스케줄링을 하기 위해 각 노드의 부하 정보가 필요하다. 로드 모니터에 의해 수집된 부하 정보는 커널에 유지된다. (그림 8)은 부하를 고려한 스케줄링의 알고리즘을 나타내는 의사코드(pseudocode)이다. 스케줄링 모듈은 가상 IP로 사용자의 요청이 도착하면 우선 자기 노드의 부하량이 임계값보다 작은지 먼저 판단한다. 부하량이 임계값보다 작으면 자기 노드에서 처리를 할 수 있도록 하고, 임계값보다 크면 부하량이 가장 작은 서버를 선택한다. 커널에서는 선택된 노드의 IP를 캡슐화하여 후면 서버로 사용자 요청패킷을 IP 터널을 통해 전송하게 된다. 각 서버들의 부하량을 기반으로 한 스케줄링을 함으로써 부하가 많은 노드에 사용자의 요청을 계속 분배하는 것을 방지하고, 특정 노드의 과부하를 방지할 수 있도록 한다.

```
Procedure ip_vs_pro_scheduling(service)
{
        list = service → destinations ;
        least = select_local_entry(list) ;
        local_load = least → cpu + least → disk ;
        if (local_load > least → threshold) {
                for all entry of list
                {
                        dest = select_dest_entry(list) ;
                        dest_load = dest → cpu + dest → disk ;
                        if (dest_load < local_load) {
                                least = dest ; local_load = dest_load ;
                        }
                }
        }
        return least ;
}
```

(그림 8) 부하 기반 스케줄링 알고리즘 의사코드

4. 실험 및 분석

단일 가상 서버와 같이 고가용성 기능이 없는 클러스터 환경에서는 후면서버가 장애로 인해 서비스를 하지 못할 경우 장애 서버로 할당되는 사용자 요청에 대해서는 서비스가 불가능하다. 따라서 클러스터의 성능에 큰 영향을 미친다. 반면에 본 논문에서 구현한 다중 가상 서버에서는 하트비트와 Fake 메커니즘을 이용하여 장애 서버에 할당되는 사용자 요청을 다른 후면서버가 대신 받아서 처리한다. 따라서 다중 가상 서버에서는 장애 서버가 발생하여도 클러스터의 서비스나 성능에는 크게 영향을 미치지 않는다. 장애 서버가 발생했을 경우에는 고가용성이 있는 다중 가상 서버의 성능이 좋을 수밖에 없다. 따라서 본 논문에서는 다양한 클러스터 환경에서 장애 서버가 발생하지 않을 경우의 성능비교에 중점을 두었다.

본 논문에서 구현한 웹 서버 클러스터의 효율성을 평가하기 위하여 다음과 같이 다양한 웹 서버 클러스터를 구성하고 성능을 측정하여 비교 분석하였다. 첫째, 단일서버(SS)는 웹 서버 한 대가 웹 서비스를 하는 일반적인 형태이다. 둘째, 단일 가상 서버(S-LVS)는 한 대의 전면서버가 사용자의 요청을 분산하는 리눅스 가상 서버 형태이다. 셋째, 단일 재지향(S-R)은 한 대의 서버가 사용자의 요청을 HTTP 재지향 방식을 이용하여 실제 서비스를 하는 서버에 라운드 로빈 방식으로 차례로 분배하는 형태이다. 넷째, 부하기반 재지향(LB-R)은 S-R 방법과 유사하다. 사용자 요청을 받은 첫 번째 서버는 클러스터 서버들 중 부하가 가장 적은 서버에 사용자 요청을 재지향하도록 하는 형태이다. 다섯째, 다중 가상 서버(M-LVS)는 본 논문에서 구현한 방법으로 다수의 전면서버가 있는 리눅스 가상 서버형태이다. 여섯째, 다중 재지향(M-R)은 S-R 방법과 비슷한 방법이고, 사용자 요청을 분산하는 전면서버가 다수인 형태이다.

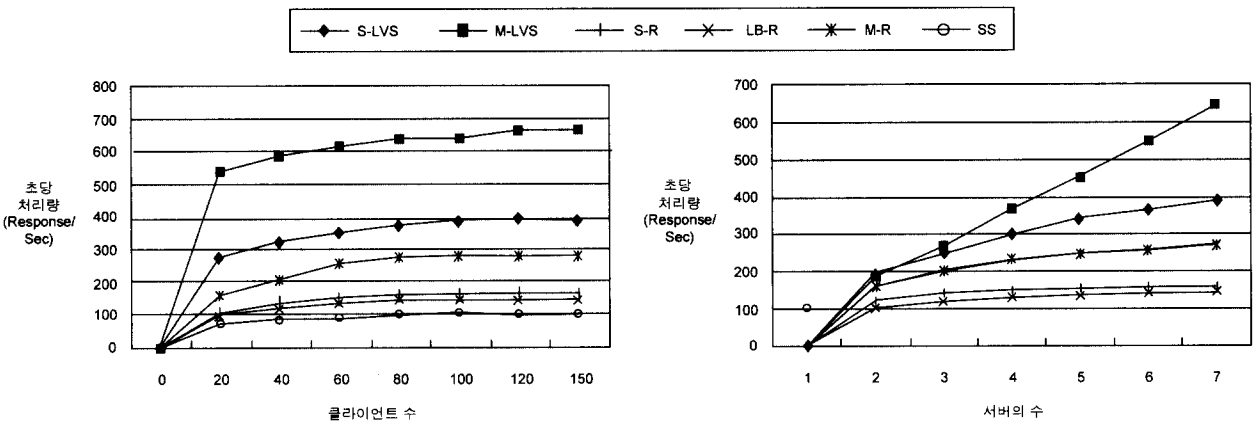
실험에 사용한 클러스터는 P-Ⅲ 800MHz CPU, 256MB RAM, 100Mbps의 이더넷 카드 2개로 구성되어 있는 7대의 개인용 컴퓨터(PC)를 사용하였다. 운영체제는 리눅스(커널 버전 2.4.13)를 사용하였고 리눅스 가상 서버를 위해서 ipvs-0.9.6 커널 패치를 하였다. 웹 서버는 아파치(Apache) 웹 서버를 사용하였다. 웹 서버 클러스터의 성능을 평가하기 위해서 SURGE라는 공개소스 프로그램을 수정하여 사용하였다[14]. SURGE는 실제 사용자의 행동을 대신하는 쓰레드를 발생해서 웹 서버에 접속하여 주어진 테스트 시간동안 사용자 요청을 발생한다. 본 논문에서는 SURGE를 일정시간 동안 수행한 후 결과로 얻은 로그파일에서 서버들의 초당 처리량을 구하여 클러스터의 성능을 평가하였다.

각 웹 서버 클러스터의 성능을 측정하기 위하여 우선 서버의 수를 고정시키고 쓰레드의 수를 증가시키면서 최대 성능을 보이는 클라이언트를 수를 구하였다. 그 다음에는 최대 성능을 보이는 클라이언트 수를 고정시키고 서버들의 수를 증가시키면서 웹 서버 클러스터의 성능을 측정하였다.

4.1 다양한 형태의 웹 서버 클러스터 성능 평가

(그림 9)(a)는 서버의 수를 7대로 고정시키고 클라이언트의 증가에 따른 클러스터의 성능을 측정한 결과를 보여준다. S-R과 LB-R, M-R은 HTTP 재지향 메커니즘에 의한 2단계 스케줄링을 사용한다. 사용자의 요청을 받은 전면서버는 새로운 서버로의 재지향 코드를 포함한 응답을 사용자에게 다시 보낸다. 응답을 받은 사용자는 실제 처리를 하게 될 서버로 다시 사용자 요청을 보낸다. 따라서 이와 같은 응용프로그램 수준의 스케줄링 방법을 사용하는 클러스터는 사용자에게 빠른 응답시간을 보장하지 못하므로 초당 처리량은 많이 증가하지는 않는 것을 볼 수 있다. S-R이 SS 방법보다 처리량이 좀 더 많은 것은 실제 사용자 요청을 처리하는 후면서버가 6대로 구성되어 있기 때문이다. LB-R은 재지향할 서버를 선택하는 부분에서 시간이 지연되기 때문에 S-R보다 처리량이 조금 적다. M-R은 S-R보다 거의 1.5배의 처리량을 보여주고 있다. 이것은 사용자 요청을 재전송하는 서버를 다수로 설정하여 더 많은 사용자 요청을 받아들였기 때문이다. 리눅스 가상 서버를 이용한 웹 서버 클러스터는 전면서버가 사용자 요청 패킷만 보고 스케줄링 해주기 때문에 빠른 응답시간과 많은 처리를 할 수 있다. 따라서 응용프로그램 수준의 스케줄링을 하는 클러스터 형태보다 훨씬 많은 처리량을 보여주고 있다. M-LVS는 S-LVS보다 1.5배 이상의 성능 향상을 보여주고 있다. 이것은 사용자 요청을 분산하는 전면서버를 다수로 두었기 때문이다.

(그림 9)(b)는 클라이언트 수를 고정시키고 사용자 요청을 처리하는 서버를 7대까지 증가시키면서 성능을 측정한 결과를 나타낸다. S-LVS, S-R, LB-R은 사용자의 요청을 분산하는 전면서버가 1대로 고정되어 있고 후면에서 실제로 웹 서비스를 하는 서버의 수를 증가시키면서 실험하였다. M-LVS와 M-R의 경우에는 부하를 분산하는 전면서버를 증가시키면서 실험하였다. S-R과 LB-R은 후면에 서비스를 처리하는 서버들이 증가하면서 성능향상이 조금 있지만 서버의 수가 증가해도 지속적인 성능향상은 보이지 않고 있다. 이것은 사용자 요청을 재지향시키는 서버가 과부하가 발생하여 더 이상의 성능은 보이지 않는 것이다. M-R은 사용자의 요청을 받는 전면서버들이 증가함에 따라 S-R보다 성능향상이 있다. 그러나 HTTP 재지향에 따른 네트워크의 오버헤드와 시간 지연 때문에 많은 성능향상을 보이지 않았고 지속적인 성능의 향상도 보이지 않는다는 것을 볼 수 있다. 리눅스 가상 서버를 이용하여 구성한 웹 서버 클러스터는 서버의 수가 증가하면서 HTTP 재지향을 이용한 방법보다 많은 성능의 향상을 보이고 있다. 리눅스 가상 서버는 전면서버가 IP 수준의 스케줄링을 하기 때문에 다른 형태의 웹



(a) 클라이언트 증가에 따른 성능 변화 (b) 서버의 증가에 따른 성능 변화

(그림 9) 다양한 형태의 웹 서버 클러스터 시스템의 성능 변화

서버 클러스터보다 성능이 좋다. 그러나 서버의 수가 증가할수록 증가하는 비율이 줄어드는 이유는 전면서버가 한 대이므로 부하가 많이 발생하여 응답시간이 늦어지면서 지속적인 성능의 향상은 보이지 않는다. M-LVS는 서버의 수가 증가함에 따라 처리량이 거의 선형적으로 증가하는 것을 볼 수 있다. M-LVS는 모든 노드들이 전면서버와 후면서버의 역할을 동시에 수행하고 부하량이 많지 않으면 자기 노드에서 처리하기 때문에 HTTP 재지향 방법보다 시간 지연이 발생하지 않는다. 또한 전면서버들이 다수이므로 병목현상에 의한 SPOF를 없앨 수 있다. 단일 서버의 처리량에 스칼라 곱으로 증가하지 않은 것은 전면서버와 후면서버의 역할을 동시에 할 수 있도록 했기 때문에 서버의 오버헤드가 발생했고, 사용자가 라운드 로빈 DNS를 한 번 더 거쳐서 처리가 되기 때문이다.

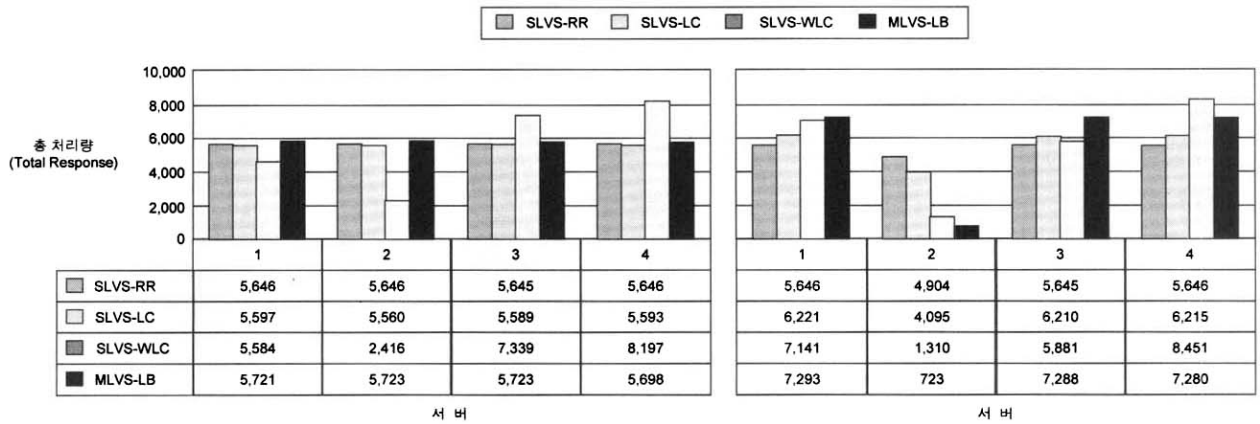
본 실험에서는 각 웹 서버 클러스터들은 사용자 요청을 처리하는 서버들의 수가 증가하면서 성능의 향상을 보인다는 것을 알 수 있고, 사용자 요청을 분산하는 서버가 처리량의 한계로 지속적인 성능의 향상을 가져올 수 없다는 것을 확인했다. 또한 사용자 요청을 분산하는 서버를 다수로 설정하여 많은 성능 향상을 가져올 수 있다는 것을 볼 수 있다. 또 응용프로그램 수준의 스케줄링 보다는 IP 수준의 스케줄링을 하는 리눅스 가상 서버 형태가 더 많은 처리를 한다는 것을 볼 수 있다. 따라서 본 논문에서 구현한 웹 서버 클러스터의 형태가 성능이 우수한 것을 알 수 있다.

4.2 부하량에 따른 부하분배 방법의 평가

본 논문에서 제안한 M-LVS의 부하량에 따른 부하분배 방법(MLVS-LB)의 성능을 측정하기 위해 S-LVS에서 제공하는 스케줄링 알고리즘 중 라운드 로빈 스케줄링(SLVS-RR)과 최소 연결 스케줄링(SLVS-LC), 가중치 최소 연결 스케줄링(SLVS-WLC) 방법과 비교하였다. SLVS-WLC 방식에서 가중치는 서버 순서대로 2 : 1 : 3 : 4으로 설정하였다. 각 서버들에 CPU의 사용량이 0.1%로 거의 부하가 없고 부하 비율이 비슷한 상태와 두 번째 노드에 CPU 사용량이 90%로 특정 서버에 부하가 많을 때의 처리량을 알아보는 실험을 하였다. 이 때 특정 노드의 부하는 CPU와 디스크를 사용하도록 하는 프로그램을 작성하여 발생시켰다.

(그림 10)(a)에서 보듯이 부하 비율이 비슷할 때 SLVS-RR, SLVS-LC, MLVS-LB 방법은 비슷한 결과를 나타내었고, SLVS-WLC 방법은 4번 서버에 더 많은 사용자 요청을 할당하였다. SLVS-RR 방법은 등록된 서버에 차례로 사용자의 요청을 할당하는 방법이기 때문에 모든 서버에 고른 사용자 요청을 할당한다. SLVS-LC 방법은 현재 등록되어 있는 서버들 중 사용자 요청이 가장 적게 연결된 서버에 사용자 요청을 할당하는 방식이다. 본 논문에서 실험한 서버들의 경우 시스템의 성능이 같고 평균 부하 비율이 거의 같기 때문에 SLVS-LC 방법은 모든 서버에 사용자 요청을 비슷하게 할당하였다. MLVS-LB 방법은 각 서버들의 부하량이 많지 않기 때문에 거의 모든 사용자 요청을 자기 노드에서 처리를 해서 모든 서버들의 총 처리량이 비슷하게 나왔다. SLVS-WLC 방법은 SLVS-LC 방식에 가중치를 두어 가중치가 큰 서버에 많은 사용자 요청을 할당한다. 본 실험에서 SLVS-WLC 방법은 가중치가 가장 큰 4번 서버에 가장 많은 사용자 요청을 할당하였기 때문에 4번 서버의 처리량이 가장 많은 것을 볼 수 있다.

(그림 10)(b)에서 보듯이 부하가 두 번째 서버에 많은 경우 SLVS-RR 방법은 부하량에 상관없이 비슷한 요청을 할당하고 있는 것을 볼 수 있다. 부하량이 많은 서버에 사용자 요청이 조금 적게 할당된 것처럼 보이는 것은 서버의 부하량이 커서 응답시간이 길어졌기 때문이다. SLVS-RR 방법은 부하량이 많은 서버에도 사용자 요청을 상대적으로 많이 분배함으로써 서버의 부하를 가중시키고 응답시간이 길어지게 한다. 따라서 서버들의 부하량이 불균형해지게 되고, 더 심해질 경우에는 서버의 장애를 초래하게 되어 클러스터 시



	1	2	3	4
SLVS-RR	5,646	5,646	5,645	5,646
SLVS-LC	5,597	5,560	5,589	5,593
SLVS-WLC	5,584	2,416	7,339	8,197
MLVS-LB	5,721	5,723	5,723	5,698

	1	2	3	4
SLVS-RR	5,646	4,904	5,645	5,646
SLVS-LC	6,221	4,095	6,210	6,215
SLVS-WLC	7,141	1,310	5,881	8,451
MLVS-LB	7,293	723	7,288	7,280

(a) 부하 비율이 비슷할 때의 비교 (b) 특정 서버에 부하가 많을 때의 비교

(그림 10) 부하 비율이 비슷할 때와 특정 서버에 부하가 많을 때의 스케줄링 비교

스템의 성능을 떨어뜨리게 된다. SLVS-LC 방법도 부하 비율이 비슷할 때와 비교하여 크게 변화는 없었다. 부하량이 많은 서버에 사용자 요청이 조금 적게 할당된 것처럼 보이는 것은 SLVS-RR 방법과 마찬가지로 서버의 부하량이 커서 응답시간이 길어졌기 때문이다. SLVS-WLC 방법은 부하가 많은 서버에는 사용자 요청이 적게 할당되었다. 그러나 본 실험에서 설정한 가중치가 가장 낮은 곳에 부하가 많이 발생한 것일 뿐, 각 서버들의 부하가 동적으로 부하가 변화하는 상황에서는 가중치를 동적으로 변경하기가 어렵기 때문에 부하의 불균형을 일으키게 된다.

MLVS-LB 방법은 임계값을 정해서 부하량이 그 값보다 작으면 자기 노드가 처리를 하기 때문에 SLVS-RR 방법과 비슷하게 동작하는 것처럼 보이지만 서버의 부하량이 임계값을 넘을 때에는 부하량이 가장 작은 서버를 선택하여 IP 터널을 통해 요청을 보낸다. 부하량이 많은 노드에 사용자 요청을 적게 할당함으로써 서버들의 부하량을 어느 정도 고르게 할 수 있고, 특정 서버가 과부하로 장애를 일으키는 것을 막을 수 있다.

5. 결론 및 향후 과제

인터넷이 새로운 산업의 형태로 발전하면서 웹 사이트에 클러스터링 기술을 많이 사용하고 있다. 하드웨어 장비를 사용하면 성능이 우수하지만 중소형 사이트를 구성하기에는 가격 부담이 많이 된다. 본 논문에서는 중소형 사이트에서 비용부담 없이 구성할 수 있는 고가용성 웹 서버 클러스터를 설계 및 구현하였다. 전면 서버와 후면 서버의 구분을 없애 다수의 전면 서버를 구성하여 전면 서버에 대한 병목현상을 제거하였고, 또한 손쉬운 확장성을 보장하였다. 그리고 하트비트와 로드 모니터를 이용하여 클러스터 시스템의 고가용성을 보장하도록 하였다. 또한 서버들의 성능과 부하를 고려한 동적인 부하 분배방법을 적용하여 부하가 많은 서버에 사용자 요청을 적게 할당하도록 하였다. 그 결과로 각 서버들 간의 부하 불균형 문제를 좀 더 효율적으로 해결하였다. 실험을 통하여 부하를 분산하는 전면 서버가 하나일 때보다 다수일 때가 1.5배 이상의 성능향상을 가져오는 것을 볼 수 있었다. 또한 응용프로그램 수준의 스케줄링 보다 IP 수준의 스케줄링을 하는 것이 약 1.5배 정도의 성능이 더 좋은 것을 볼 수 있다.

웹 서버 클러스터는 고가용성을 위한 하트비트 기능에서 장애 노드의 판단과 Fake 할 때의 시간적 지연 동안에 사용자 요청 패킷들이 손실될 수 있다. 향후에는 장애 노드와 서비스 가능 서버들에 대한 판단 지연과 Fake에 따른 사용자 요청 패킷들의 손실을 줄이는 연구가 좀 더 필요하다.

참 고 문 헌

[1] Huican Zhu, Ben Smith, Tao Yang, "A Scheduling Framework for Web Server Clusters with Intensive Dynamic Content Processing," *Proceeding of the 1999 ACM SIGMETRICS Conference*, May, 1999.

[2] Raykumar Buyya, "High Performance Cluster Computing vol1," *Chap 36. A Scalable and Highly Available Clustered Web Server*, Prentice Hall, 1999.

[3] RADWARE Web Server Director Pro, Available online at, http://www.radware.com/.

[4] Valera Cardellini, Michele Colajanni, Philip S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing*, Vol.3, No.3, pp.28-39, May/June, 1999.

[5] Wensong Zhang, Shiyao Jin and Quanyuan Wu, "Creating Linux Virtual Server," *The 5th Annual Linux Expo Conference*, May, 1999.

[6] Chad Yoshikawa et al., "Using Smart Clients to Build

Scalable Services," *Proceedings of the USENIX 1997 Annual Technical Conference*, January, 1997.

[7] Thomas T. Kwan, Robert E. McGrath and Daniel A. Reed, "NCSA's World Wide Web Server : Design and Performance," *IEEE Computer*, Vol.28, No.11, pp.68-74, November, 1995.

[8] Thomas T. Brisco, "DNS Support for Load Balancing," *Network Working Group RFC 1794*, April, 1995.

[9] Daniel Andresen et al., "SWEB : Toward a Scalable World Wide Web-Server on Multicomputers," *Proceedings 10th IEEE International Symposium Parallel Processing*, pp. 850-856, 1996.

[10] Eric Anderson, Dave Patterson and Eric Brewer, "The Magicrouter, An Application of Fast Packet Interposing," *Class Report*, University of California, Berkely, May, 1996.

[11] Cisco's Cisco LocalDirector, Available online at, http://cio.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml.

[12] Guerney D. H. Hunt et al., "Network Dispatcher : A Connection Router for Scalable Internet Services," *J. Computer Networks and ISDN Systems*, Elsevier Science, Amsterdam, Netherlands, Vol.30, 1998.

[13] Om P. Damani et al., "ONE-IP : Techniques for Hosting a Service on a Cluster of Machines," *J. Computer Networks and ISDN Systems*, Vol.29, Elsevier Science, Amsterdam, Netherlands, pp.1019-1027, September, 1997.

[14] Paul Barford and Mark Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proceeding of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp.151-160, July, 1998.

문 종 배

e-mail : comdoct@ss.ssu.ac.kr

2000년 숭실대학교 컴퓨터학과(학사)

2002년 숭실대학교 컴퓨터학과(공학석사)

2003년~현재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 분산/병렬 컴퓨팅, 그리드, 웹 서비스, 리눅스

김 명 호

e-mail : kmh@ssu.ac.kr

1989년 숭실대학교 전자계산학과(학사)

1991년 포항공과대학교 전자계산학과 (공학석사)

1995년 포항공과대학교 전자계산학과 (공학박사)

1995년 한국전자통신연구소 선임연구원

1998년~1999년 미국 테네시주립대 교환교수

1995년~현재 숭실대학교 정보과학대학 컴퓨터학부 교수

관심분야 : 분산/병렬 컴퓨팅, 그리드, 웹 서비스, BI, 보안