리눅스 클러스터 웹 서버의 요청 스케줄링 기법 성능 평가

이 규 한[†] · 이 종 우[††] · 이 재 원[†††] · 김 성 동[††††] · 채 진 석[†††††]

요 약

클러스터 웹 서버에서 사용되는 요청 분배 기법에는 부하 균형 요청 분배 기법과 내용 기반 요청 분배 기법이 있다. 부하 균형 요청 분배 기법은 실제 서버들의 부하 균형을 목적으로 실제 서버를 선택하는 기법이다. 반면 내용 기반 요청 분배 기법은 캐쉬 친화적인 기법으로 요청 패킷의 내용에 따라서 실제 서버를 선택하게 된다. 이는 부하 균형보다는 각 실제 서버 내의 캐쉬 참조율을 높이는 것을 목적으로 한다. 하지만 현재 두 기법의 성능 비교에 관한 명확한 자료가 부족한 상태여서 본 논문에서는 두 기법에 관한 성능 평가 결과를 보이고자 한다. 이를 위해 우리는 리눅스 클러스터 웹 서버를 구현한 후 이 두 기법간의 성능을 비교, 분석한다. 이 비교/분석 결과를 활용할 경우 부하 상황에 따른 적절한 요청 분배 기법을 선택하는 데에 도움을 줄 수 있다는 것이 본 논문이 기여하는 바라 할 것이다.

Performance Evaluation of Request Scheduling Techniques in the Linux Cluster Web Server

Kyu-Han Lee[†] · Jongwoo Lee[††] · Jae Won Lee[†††]
Sung-Dong Kim[††††] · Jinseok Chae[†††††]

ABSTRACT

The request scheduling algorithms being used for the cluster web servers are mostly in two categories : load-balancing and contents-based cache affinity. The goal of the load-balancing algorithms is to balance the loads between real servers. On the other hand, contents-based scheduling algorithm exploits the cache affinity in a way that the same type of requests are to be directed to a dedicated real server allowing load imbalance. So the performance comparison of the two algorithms is necessary, nevertheless the related experiment results are not much suggested. In this paper, performance evaluations have been done to compare the performance of the two scheduling algorithms. To accomplish this, we first implement a linux cluster web server, and then present the performance measurement results. The main contribution of this paper is to help the cluster web server administrators to select an algorithm fitting in with their circumstances from the two algorithms.

키워드 : 클러스터 시스템(Cluster System), 웹 서버(Web Server), 요청 분배 알고리즘(Request Distribution Algorithm), 성능 평가(Performance Evaluation)

1. 서 론

최근 한국 인터넷 정보 센터에서 실시한 우리나라의 인터넷 이용자 수 조사 결과에 의하면 1999년 10월, 943만명(전국민의 22.4%)에 불과하던 인터넷 이용자가 2002년 6월, 2565만명(전국민의 58%)으로 증가하여 3년 사이에 약 3배의 달하는 증가율을 보이고 있다[1]. 인터넷 이용자가 우리나라 전 국민의 절반을 넘어선 셈이다. 하지만 현재에는 인터넷 사용자의 증가율보다는 사용자의 패턴이나 인터넷의 질적인 측면 등으로 그 관심이 모아지고 있다. 이는 인터넷의 사용이 많아짐에 따라 사용자에게 서비스를 제공하는 웹 서버의 부하도 그 만큼 커지고 있기 때문인데, 웹 서버의 처리량도 아울러 빠르게 증가하고 있고 아주 유명한 웹 서버의 경우는 짧은 시간에도 과부하가 걸리기 쉽다. 과부하를 극복하기 위한 방법은 두 가지가 있는데, 그 첫 번째는 서버를 좀 더 높은 성능의 서버로 교체하는 것이다. 하지만 요구가 계속 증가하면 계속해서 업그레이드를 해야하기 때문에 복잡하고 비용도 높다. 이러한 단일 서버 시스템의 가장 큰 단점은 고가의 가격과 가용성을 보장할 수 없다는 것이다. 가격이 높기 때문에 확장에도 어려움이 있고 단일 서버이기 때문에 서버장애 시에 서비스는 정지된다. 두 번째 방법은 다중 서버를 구성해서 서버 클러스터를 이용하는 것이다. 이는 가용성도 높고 확장성 면에서도 좋다. 클러스터 서버 구성에도 몇 가지 방법이 있는데, 그 중 리눅스 클러스터에서 많이 사용되는 것은 라운드 로빈 DNS 방식과 가상 서버(Linux Virtual Server, LVS) 방식이다[2-4].

† 정 회 원 : 메조마케팅코리아 AD-Tech 팀
†† 정 회 원 : (주)아이닉스 소프트 이사
††† 정 회 원 : 성신여자대학교 컴퓨터정보공학부 교수
†††† 정 회 원 : 한성대학교 컴퓨터시스템공학과 교수
††††† 종신회원 : 인천대학교 컴퓨터공학과 교수
논문접수 : 2003년 1월 21일, 심사완료 : 2003년 7월 15일

라운드 로빈 DNS는 구현이 쉬운 반면에 서버들 사이의 부하 불균등 문제가 있다[5, 6]. LVS를 이용한 클러스터 웹 서버는 여러 대의 값 싼 PC들을 연결하여 서버로 들어오는 요청을 부하 분산 서버가 각각의 서버들에게 스케줄링 알고리즘을 통해 분산시키는 구조를 갖고 있다. 서버로 들어오는 수많은 사용자의 요청을 여러 실제 서버에 나눔으로써 서버 전체의 성능을 향상시키고자 하는 것이 클러스터 웹서버이다. 여러 대의 실제 서버가 있기 때문에 단일 서버가 보장하지 못했던 고가용성을 보장할 수도 있다[2].

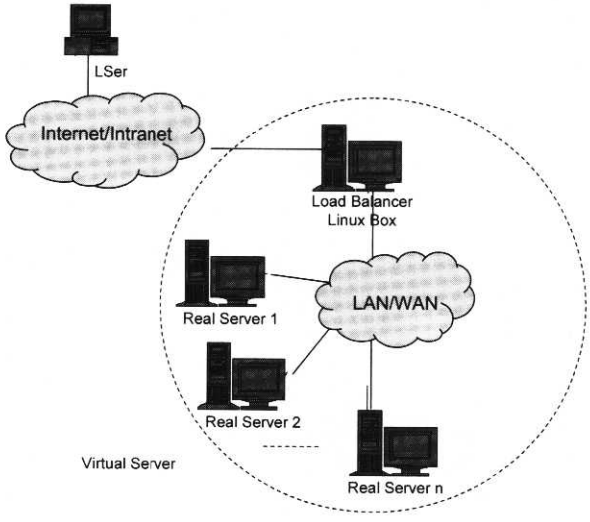
클러스터 웹 서버에서 사용되는 요청 분배 기법에는 내용 기반 요청 분배 기법과 부하 균형 요청 분배 기법이 있다. 내용 기반 요청 분배 기법은 캐쉬 친화적인 기법으로 요청 패킷의 내용에 따라서 실제 서버를 선택하게 된다[2]. 이는 부하 균형보다는 캐쉬 참조율을 높이는 것을 목적으로 한다. 반면 부하 균형 요청 분배 기법은 실제 서버들의 부하 균형을 목적으로 실제 서버를 선택하는 기법이다. 하지만 현재 두 기법의 성능 비교에 관한 명확한 자료가 부족한 상태여서 본 논문에서는 두 기법에 관한 성능 평가 결과를 보이고자 한다. 이를 위해 본 논문에서는 LVS를 구현한 후 이 두 기법간의 성능을 비교, 분석한다. 이 비교/분석 결과를 활용할 경우 부하 상황에 따른 적절한 요청 분배 기법을 선택하는 데에 도움을 줄 수 있다는 것이 본 논문이 기여하는 바라 할 것이다.

본 논문에서는 LVS를 이용하여 실제 서비스 가능한 클러스터 웹 서버를 구현하고, LVS가 사용할 수 있는 요청 패킷 분배 기법들 중 실제 서버들의 부하 균형을 위한 기법과 캐쉬 친화성을 활용한 내용 기반 기법에 대해 살펴본다. 그리고 이들 두 기법간의 성능 비교를 통해 각각의 장단점을 파악하여 상황에 따른 효율적인 방법을 제시한다. 본 연구에서 수행될 성능 평가는 다음과 같다. 첫째, 실제 서버 수를 증가시키면서 부하 균형 스케줄링 기법과 내용 기반 스케줄링 기법의 성능을 비교한다. 둘째, 요청되는 컨텐츠의 유형(type) 수에 따라 각 기법의 성능 변화를 측정한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 LVS의 개요와 요청 패킷 분배 기법에 관해 살펴보도록 하겠다. 제 3장에서는 성능 평가를 위한 클러스터 웹 서버의 구현에 관해 설명하고, 제 4장에서는 성능 평가 결과와 그에 대한 분석을 제시한다. 제 5장에서는 결론과 향후 연구 과제에 대하여 논한다.

2. 리눅스 클러스터 웹 서버 구성 요소 분석

2.1 LVS 구성

클러스터 웹 서버의 특징에는 고가용성과 확장성을 들 수 있는데[2, 3], LVS는 여러 대의 실제 서버들을 클러스터로 구성하여 확장성과 가용성을 구현한 서비스이다. LVS는 소프트웨어적으로 H/W Layer-4 스위치를 구현한 것이라 할 수 있다[2, 3, 5]. LVS는 클러스터 전면에 부하 분산 서버(load balancer)를 두어 이로 하여금 사용자의 요청을 각 실제 서버에게 분배하도록 한다. 사용자는 부하 분산 서버와 실제 서버들이 공유하는 가상 서버 주소로 요청을 하게 되며 부하 분산 서버는 받은 요청을 분배 알고리즘에 따라 선택된 실제 서버에게 분배한다. 실제 서버는 요청 받은 서비스를 수행하여 그 결과를 사용자에게 직접 주거나 부하 분산 서버를 통해 돌려준다. 클러스터의 구조는 최종 사용자에게는 투명하며 사용자는 오직 하나의 가상 서버만을 볼 수 있다. 기본적인 구조는 (그림 1)과 같다.



(그림 1) 리눅스 가상 서버

본 논문에서의 관심 사항은 아니지만 LVS 구성의 중요 요소 중에는 패킷 포워딩 서브시스템이 있는데 이에 대해서는 [2, 7]에 자세히 기술되어 있다. 본 논문에서는 직접 라우팅 기법을 사용하여 패킷을 포워딩 한다. 다음 절에서는 요청 분배 서버가 실제 서버를 선택하는 패킷 스케줄링 기법들에 대해 살펴보도록 한다.

2.2 패킷 스케줄링 기법

리눅스 클러스터 웹 서버에서 일반적으로 사용하는 요청 분배 기법은 크게 두 가지 부류로 나눌 수 있다. 첫째는 실제 서버들의 부하 균형을 위한 기법들이고, 다른 하나는 캐쉬 친화성(cache affinity)을 활용하는 내용 기반 패킷 스케줄링 기법이다[2, 7, 8]. 첫 번째 방식인 부하 균형 스케줄링 알고리즘은 네트워크 계층의 IP 수준에서 동작하기 때문에 LVS에서는 이를 IPVS(IP Virtual Server)라고 칭하고 있다. IPVS는 패킷 필터링 모듈로 간단히 설치할 수 있으므로 사실상 리눅스 커널 자체는 전혀 수정하지 않아도 된다는 장점이 있다. 두 번째 방식인 내용 기반 스케줄링 방식은 패킷의 내용까지 검토해야 하는 관계로 네트워크 계층의 TCP 수준

에서 이루어진다[9-11]. 따라서 이는 TCPVS 또는 KTCPVS라고 불리기도 한다. K가 붙은 이유는 커널의 TCP 코드를 일부 수정해야 한다는 의미를 나타내기 위함이다. IPVS와 TCPVS에서 지원되는 패킷 스케줄링 알고리즘을 살펴보면 다음과 같다[2, 8].

2.2.1 실제 서버 부하균등을 위한 패킷 스케줄링 알고리즘

- **라운드로빈(Round-Robin, *rr*)**

부하 분산 서버가 요청된 패킷들을 실제 서버들에게 돌아가면서 한 번씩 할당한다. 이는 정확한 부하 분산은 되지만 실제 서버들의 활성 접속 수나 응답 시간을 고려하지 않는다. 예를 들어, 실제 서버가 A, B, C라면, 요청은 A B C A B C 순으로 분배된다. 실제 서버들의 현 부하 상태를 고려하지 않는다는 면에서 이 방식은 정적 스케줄링 방식이라고 할 수 있다.

- **가중치기반 라운드로빈(Weighted Round-Robin, *wrr*)**

실제 서버들의 성능 차를 극복하기 위해서 서로 다른 처리 용량을 지정하여 처리 용량이 큰 서버에 요청을 우선 배정한다. 처리 용량이 같은 서버들 간에는 RR 방식으로 할당한다. 예를 들어, 실제 서버가 각각 A(가중치 = 3), B(가중치 = 2), C(가중치 = 1)라면, A A B A B C 순으로 분배된다. 이 역시 정적 스케줄링 방식에 속한다.

- **최소 접속(Least-Connection, *lc*)**

동적 스케줄링 알고리즘의 하나로 활성 접속 수가 가장 적은 실제 서버에게 요청을 할당하는 방법이다. 하지만 서로 다른 성능의 실제 서버들로 클러스터를 구성했을 때에는 성능을 고려하지 않고 활성 접속 수만을 고려하여 요청을 분배하므로 실제로 효율적이지 못하다. 예를 들면, 실제 서버의 처리량이 각각 A(처리량 = 4), B(처리량 = 16)의 성능을 갖고 있을 때, 현재 활성 접속 수가 A는 4, B는 5라면 들어오는 요청을 포화 상태인 A로 할당하게 될 것이다.

- **가중치기반 최소 접속(Weighted Least-Connection, *wlc*)**

실제 서버들에게 가중치를 지정하여 가중치에 따라 활성 접속 수가 적은 서버에 요청을 할당한다. 기본 가중치는 1이며 *lc*에 비해 가중치를 계산하는 부가적인 배분작업이 필요하다. 실제 서버의 활성 접속 수를 가중치로 나눈 값이 최소인 실제 서버에게 요청이 할당된다.

- **Locality-Based Least-Connection(*lblc*)**

요청은 한 실제 서버가 포화될 때까지 계속 한 서버로만 전달된다. 실제 서버가 포화되면 그렇지 않은 실제 서버들을 *wlc* 방식으로 스케줄링 하여 또 다시 선택된 서버로만 요청을 할당한다. 한 실제 서버내의 캐쉬를 최대한 활용할 수 있다는 장점이 있다. 하지만 요청 내용을 고려하지 않기 때문에 여러 유형의 요청이 한 서버로 몰릴 경우 그 서버 내의 캐쉬 활용도는 기대만큼 높지 못할 가능성이 크다고 할 수 있다.

- **Locality-Based Least-Connection with Replication(*lblcr*)**

실제 서버들을 그룹으로 묶은 후, 현재 실제 서버 그룹 내의 서버가 모두 포화 상태가 되면 *lc* 방식으로 다른 실제 서버를 선택하여 요청을 할당한 후, 그 실제 서버를 현재 모두 포화된 실제 서버 그룹으로 포함시킨다. 포화 그룹의 상태가 일정기간 이상 안정되면 재그룹하여 원상 복구한다. 실제 서버의 수가 매우 많은 클러스터 웹 서버에서 유용하다.

- **목적지 해싱(Destination Hashing)**

패킷들의 목적지 IP 주소들로 할당된 해시 테이블의 통계 분석을 통하여 실제 서버로 할당된다. 하지만 이 방식은 현재 중앙집중식 요청 분배 서버를 두고 있는 LVS에서는 아무 의미가 없다. 왜냐하면 사용자들이 보낸 패킷들의 목적지는 모두 이 요청 분배 서버를 향하고 있을 것이기 때문이다.

- **출발지 해싱(Source Hashing)**

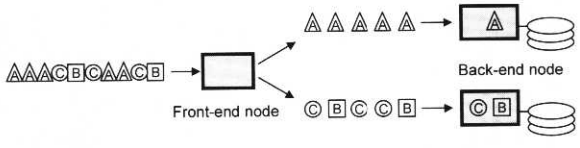
패킷들의 출발지 IP 주소들로 할당된 해시 테이블의 통계 분석을 통하여 실제 서버로 할당된다. 이론적으로 사용자들의 출발지 IP 주소가 무한 개수만큼 있다고 가정하면 이 방법은 *rr*과 유사한 알고리즘이 된다. 따라서, 이 방식은 서로 다른 IP를 가진 사용자들의 수가 제한적일 때 유용하다고 할 수 있다.

2.2.2 캐쉬 친화성 활용을 위한 패킷 스케줄링 알고리즘

클러스터 웹 서버를 구현하여 패킷을 분배하는 또 다른 방법으로는 내용 기반 스케줄링 알고리즘을 들 수 있다. 이는 실제 서버들의 부하 여부와는 별도로 실제 서버의 주 메모리 캐쉬 참조율을 높여 응답 속도를 빠르게 하고자 각 실제 서버별로 미리 지정된 유형의 요청만을 할당하도록 하는 방법이다. 이 기법의 장점은 캐쉬 참조율의 향상뿐만 아니라 서버의 데이터베이스를 분할 유지함에 따라 2차 저장장치의 확장성을 높일 수 있다는 장점도 갖고 있다. 또한 특정 서비스 형태를 위한 실제 서버의 특성화로 각 서버의 성능이 좋아진다[9-11].

(그림 2)는 내용 기반 요청 분배 알고리즘의 요청 분배 과정의 예를 보이고 있다. 이 그림에서 전위 노드(front-end node)는 요청을 분배하는 부하 분배 서버 역할을 하며 후위 노드(back-end nodes)들은 실제 서버 역할을 한다. 그림에서 볼 수 있듯이 부하 분산 서버는 여러 내용의 요청 스트림을 받았을 때 요청 패킷의 내용을 검사하여 정해진 규칙에 따라서 각 실제 서버에게 요청을 분배하고 있음을 알 수 있다. 전위 노드는 유형 A에 속한 모든 요청을 후위

서버 1에게 보내주고, 요청 유형 B와 유형 C는 모두 후위 서버 2에게 보내준다. 이렇게 하면 후위 서버들의 캐쉬에서 요청된 내용을 찾을 가능성이 증가된다. 만약 들어오는 요청을 부하 균형을 위한 방식으로 분산시키면 모든 요청은 그 유형과 상관없이 두 후위 서버들에게 섞여서 돌아갈 것이다. 그러면 각 후위 서버들 내에서 캐쉬 미스를 유발할 것이므로 캐쉬 오염도가 증가되어 결국엔 작업 부하가 불필요하게 늘어날 것이다. 부하 균형을 위한 기법들 중에서도 캐쉬 참조율의 향상을 위한 *lblc, lblcr* 같은 방법이 있지만 이들은 실제적으로 요청 패킷의 내용을 검사하지 않는다는 점에서 내용 기반 요청 분배 기법과 구별된다.



(그림 2) 내용 기반 요청 분배 알고리즘

(그림 2)의 내용 기반 요청 분배 방법은 후위 서버들 사이에 부하 불균형을 초래하여 좋지 않은 성능을 나타낼 수도 있는데, 이 때문에 TCP hand-off 등의 방법으로 후위 서버들 사이의 부하 균등을 모색하는 방법들도 있다[9, 12].

부하 분산을 담당하는 전위 노드에서 각 요청의 내용을 확인하는 방법은 요청 패킷의 내용을 직접 검사하는 것 외에는 달리 방법이 없다. HTTP 요청 패킷의 GET 메소드 뒤에 있는 요청 내용의 유형과 각 실제 서버별로 미리 정해놓은 서비스 유형을 비교하여 일치하는 실제 서버를 선택하는 것이다[8]. 이는 IP 단계에서 즉, IP 주소(가상 서버 주소)만으로 패킷을 분배하는 방법에 비하면 당연히 내용 비교에 따른 오버헤드가 존재한다. 하지만 캐쉬 참조율의 향상으로 인한 성능 향상이 검사시간에 따른 오버헤드보다 높다면 이는 문제가 되지 않을 것이다.

본 논문에서는 TCPVS를 이용해 클러스터를 구성하여 내용 기반 요청 분배 알고리즘을 구현할 것이다. 이를 이용하여 IPVS와 같은 부하 균등 요청 분배 기법과 내용 기반 요청 분배 기법 간의 성능을 비교할 것이며, 클러스터 시스템의 상황에 따라 어느 알고리즘이 최적의 성능을 보이는지를 규명하고자 한다.

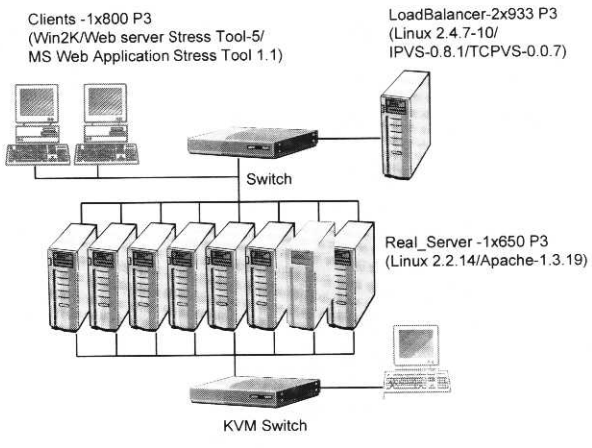
3. 리눅스 클러스터 웹 서버 구현

3.1 하드웨어 설정

본 실험에서 사용된 부하 분산 서버는 933MHz 인텔 펜티엄 III dual 프로세서, 384MB RAM과 30GB 하드디스크를 장착하고 있다. 이와 함께, 100Mbps 이더넷 카드가 1개 장착되어 있다. 이는 스위치를 통해서 실제 서버들과 연결된다. 실험에 사용된 실제 서버는 총 8대이며, 650MHz 인텔 펜티엄 III 프로세서, 128MB RAM과 10GB 하드디스크를 장착하고 있다. 부하 분산 서버와 마찬가지로 100Mbps 이더넷 카드가 1개씩 장착되어 있으며 실제 서버들은 스위치를 통해 연결된다.

클라이언트는 총 2대이며, 800MHz 인텔 펜티엄 III 프로세서를 갖고 있다. 클라이언트 1은 128MB RAM, 30GB 하드디스크를 장착했으며 10Mbps 이더넷 카드를 장착하고 있다. 클라이언트 2는 10GB 하드디스크를 장착했으며 64MB의 RAM이 장착되어 있다. 클라이언트 또한 스위치를 통해 부하 분산 서버와 실제들에게 연결되어 있다. 이처럼 낮은 수준의 PC들로 클러스터를 구성한 이유는 현실적으로 싼값의 PC들을 활용하여 높은 성능을 내고자 하는 것이 클러스터 시스템을 구축하는 이유이기 때문이다.

본 실험에서 사용된 모든 서버들은 물리적으로 동일한 네트워크 세그먼트 안에 포함되어 있다. 이는 선택된 실제 서버로 요청 패킷을 포워딩하는 방법으로 IPVS의 직접 라우팅(direct routing) 기법을 사용했기 때문이다. (그림 3)은 실험에 사용된 클러스터 시스템의 전체 구조를 보이고 있다.



(그림 3) 실험 환경

3.2 소프트웨어 설정

리눅스를 이용해 클러스터 웹 서버를 구축하는 데에 필요한 소프트웨어 설정들은 대부분 부하 분산 서버에 편중되어 있다. 실제 서버들은 분배된 요청을 처리하는 역할만 하기 때문에 웹 서버 외엔 필요한 것이 없다. 우선 클러스터 웹 서버를 구축하는데 있어 중요한 설정 사항은 사용할 요청 분배 알고리즘을 지정하는 일이다. 앞서 언급한대로 설정 가능한 요청 분배 알고리즘에는 부하 균형을 위한 IPVS와 내용 기반 요청 분배를 위한 TCPVS가 있다. 각각의 자세한 설정에 관해서는 다음절에서 설명하도록 하고 여기서는 전반적인 소프트웨어 환경에 대해서 설명한다.

부하 분산 서버에는 커널 버전-2.4.7인 레드햇 7.2를 설치했으며 요청 패킷 분배를 위한 모듈로는 IPVS-0.8.1과 TCPVS-0.0.7를 사용한다. 실제 서버는 ARP를 숨기기 위한 히든 디

바이스를 갖고 있는 커널 버전-2.2.14인 레드햇 6.2를 설치했으며 Apache-1.3.19를 웹 서버로 갖고 있다. 클라이언트 1에는 윈도 2000 위에 두 개의 부하 생성기가 설치되어 있다. 부하 생성기인 Webserver Stress Tool-5[15]와 MS Web Application Stress Tool 1.1[14, 16]이 설치되어 있는데 이는 동시 사용자 수를 최대 100명, 150명까지 설정할 수 있다. 클라이언트 2에는 윈도 2000 위에 Webserver Stress Tool-5만 설치되어 있다. <표 1>은 이와 같은 소프트웨어 설정 사항들을 요약하고 있다.

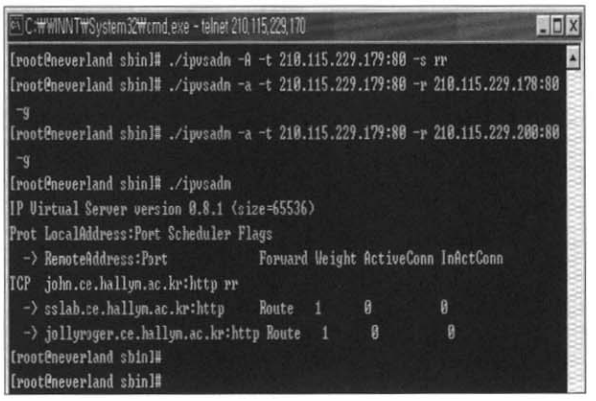
〈표 1〉 소프트웨어 구성 요소

구성 요소	버 전
운영체제	Windows 2000, Linux7.2, Linux6.2
리눅스 커널	linux-2.4.7, linux-2.2.14
LVS	ipvs-0.8.1, tcpvs-0.0.7
웹 서버	Apache-1.3.19
부하 생성기	Webserver Stress Tool-5 Professional, MS Web Application Stress Tool 1.1

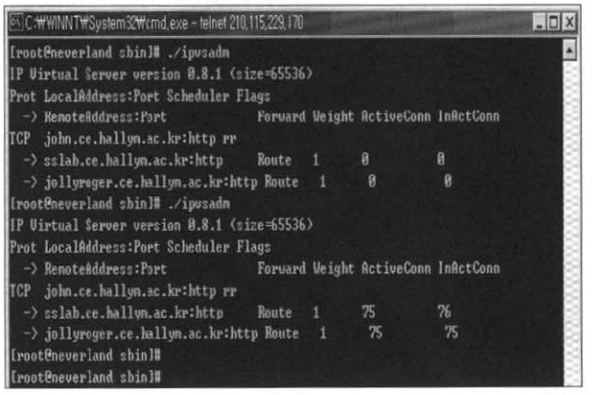
3.3 클러스터 웹 서버 구현

3.3.1 IPVS

부하 분산 서버에 IPVS를 설치하는 과정은 [2, 7]을 참조하면 간단히 끝낼 수 있다. IPVS를 이용한 클러스터 웹 서버를 설정하는 과정에는 ipvsadm이라는 도구가 주로 사용된다. 이는 가상 서버 설정과 실제 서버들의 추가, 삭제 등을 위한 도구인데 IPVS를 설치하면 같이 설치된다. 가장 먼저 할 일은 가상 서버 주소를 설정하는 일인데, 이는 부하 분산 서버의 네트워크 인터페이스에 대해 ifconfig 명령어로 에일리어싱(aliasing)을 설정함으로써 이루어진다. 그런 다음 네트워크 라우팅 테이블에 추가한다. 그리고 ipvsadm을 이용해서 가상서버를 위한 서비스를 설정하고 실제 서비스를 하게 될 실제 서버들을 추가하면 된다. (그림 4)는 IPVS 설정 과정의 예를 보이고 있는데, 첫 번째 줄은 가상 서버 주소 210.115.229.179로 HTTP 서비스를 하고 스케줄링 알고리즘으로는 라운드 로빈을 선택한 것이다. 두 번째 줄과 세 번째 줄은 179번으로 들어오는 HTTP 요청을 178번과 200번의 실제 서버들에게 보내도록 하는 실제 서버 추가 설정이다. -g 옵션은 직접 라우팅 기법을 사용하겠다는 의미이다. 서비스를 시작하는 별도의 명령어는 없으므로 (그림 4)의 위 세 명령어만으로 서비스는 시작된다. 서비스가 계속되는 상황에서도 실제 서버들의 추가, 삭제 등의 설정 변경이 가능하다. 스케줄링 알고리즘 변경도 가능하다. 가상 서버를 등록하고 시작하게 되면 ipvsadm의 명령어로 현재 설정된 가상 서버 정보와 실제 서버 각각의 부하 정도를 알 수 있는 활성 접속 수, 비활성 접속 수 등을 확인할 수 있다. (그림 5)는 현재 각 실제 서버들의 접속 수를 보여주는 한 예이다.



(그림 4) ipvsadm을 이용한 가상 서버 설정

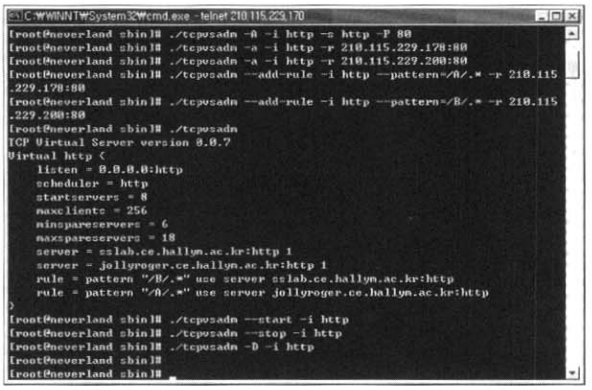


(그림 5) 실제 서버들의 부하 상태 정보

3.3.2 TCPVS

내용에 기반한 요청 분배 기법을 구현하기 위해 사용한 모듈인 TCPVS는 [8]에서 내려받아 요청 분배 가상 서버에 설치하면 된다. 가상 서버 설정에 사용되는 도구는 tcpvsadm인데, 이를 이용하면 가상 서버를 설정하고 실제 서버들을 추가하고, 내용 유형에 기반하여 요청을 분배하도록 내용 패턴(pattern)을 지정해 줄 수 있다. 가상 서버 주소를 네트워크 인터페이스에 설정하거나 실제 서버에서 'lo' 디바이스 설정은 IPVS와 동일하다. IPVS와 다른 점은 서비스의 시작과 종료를 사용자가 직접 명령어 입력을 통해 실행한다는 것이다. --start와 --stop는 서비스 시작과 종료를 지시하는 옵션을 의미한다. -D 옵션은 현재 저장되어 있는 가상 서버 설정을 삭제할 때 사용한다. (그림 6)은 가상 서버 설정에서 실제 서버를 추가하고 전체 설정을 검사한 후, 서비스를 시작하고 종료, 삭제하는 과정을 보여주고 있다. 내용을 기반으로 실제 서버를 선택하는 방법은 앞서 말했듯이, HTTP 요청패킷의 GET 메소드 뒷부분과 패턴(--pattern) 옵션 뒤에 나오는 부분을 비교하는 것이다. (그림 6)의 예에서 보면, /A/ 디렉터리 이하의 모든 파일 요청에 관해서는 실제 서버 178번으로 패킷을 보내도록 하고 요청 패킷의 GET 메소드 다음에 /B/ 디렉터리로 시작되는 모든 파일의

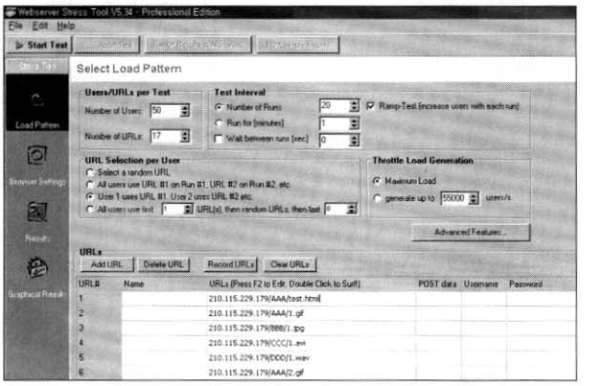
요청은 200번으로 보내진다. 하나의 실제 서버에는 오직 한 가지의 패턴만을 지정할 수 있다.



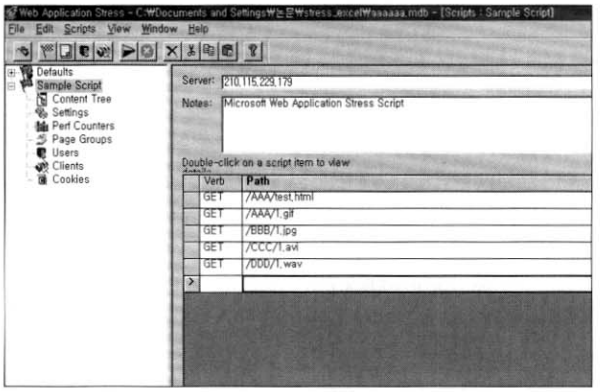
(그림 6) tcpvsadm을 이용한 가상 서버 설정

3.3.3 부하 발생기

본 논문에서는 두 가지 종류의 부하 발생기를 사용하였다. PAESSLER 사의 Webserver Stress Tool 5.0-Professional Edition[15]과 MS Web Application Stress Tool 1.1 [16] 이다. Webserver Stress Tool은 최대 동시 사용자가 100명으로 제한되어 있다. 100명이 동시에 접속을 할 수도 있고, 처음엔 1명으로 시작해서 마지막엔 100명으로 끝나는 Ramp-test도 가능하다. 이 두 도구는 모두 사용자가 요청을 한 시점부터 응답 파일을 받을 때까지의 사용자 평균 대기 시간을 결과로 보여준다. 클라이언트 1에서는 이 두 성능 평가 도구를 모두 실행시키도록 하였다. (그림 7)과 (그림 8)은 실행화면의 예를 보이고 있다. MS사의 제품은 150명의 사용자가 계속해서 요청을 발생시키도록 하였고, PAESSLER 사 제품에서는 총 50명이 20단계에 거쳐서(Ramp-test) 요청을 발생시키도록 하였다. (그림 7)에서 볼 수 있듯이 여러 파일을 차례로 혹은 무작위로 요청할 수 있다. 클라이언트 2에서는 Webserver Stress Tool를 100명의 동시 사용자가 계속해서 요청을 발생시키도록 하였다.



(그림 7) Webserver Stress Tool-5



(그림 8) MS Web Application Stress Tool 1.1

3.3.4 실제 서버

실제 서버에서는 클라이언트에게 응답을 할 때, 송신자 주소를 가상 서버 주소로 하기 위해 'lo' device에 aliasing을 이용해서 가상 서버 주소를 설정한다[7]. 또한 실제 서버 자신의 물리 주소를 숨기기 위해서 히든 디바이스의 플래그 값 '0'을 '1'로 변경해야만 한다[7]. 변경하지 않을 경우에는, 한번 응답받은 클라이언트는 자신에게 남아있는 ARP를 통해서 같은 실제 서버로만 계속해서 접속할 것이다. 각 실제 서버들은 모두 동일한 서비스 대상 파일들을 소유하고 있는데, AAA, BBB, CCC, DDD, EEE, FFF, GGG, HHH 라는 총 8개의 디렉터리에 나뉘어 저장되어 있다. 이는 내용 기반 알고리즘의 경우, 실제 서버 수에 따른 패턴 지정을 편리하게 하기 위함이다. 실제 서버가 4대일 때에는 4개의 디렉터리로 패턴을 지정하고 실제 서버 수가 2대일 때에는 BBB를 AAA안에 포함시키고 DDD를 CCC안에 포함시켰다. 부하 발생기에서 요청을 보낼 때에는 210.115.229.VIP/AAA/test.html 과 같은 방식으로 한다.

4. 성능 평가 및 결과

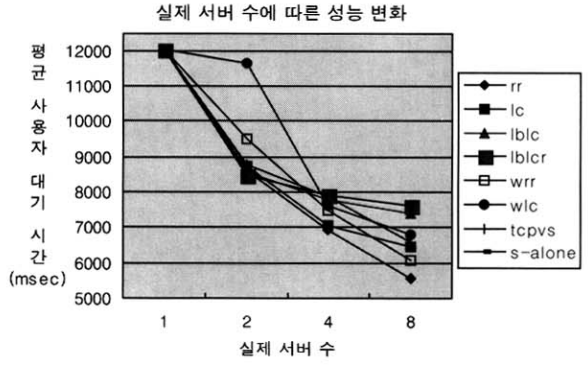
본 논문에서 실시한 성능 평가의 항목은 다음과 같다. 첫째, 실제 서버를 1대, 2대, 4대, 8대로 증가시키면서 요청 분배 기법간의 사용자 평균 대기시간을 알아보고, 각 기법간의 사용자 평균 대기시간을 비교, 분석한다. 둘째, 요청하는 URL의 수를 1, 4, 5, 7, 9, 13, 17개로 증가시켜가고 실제 서버를 1대, 2대, 4대로 증가시키면서 부하 균형을 위한 요청 분배 기법과 내용에 기반한 요청 분배 기법간의 성능을 비교, 분석한다.

모든 실험은 클라이언트 1의 Webserver Stress Tool의 결과만을 채택하며 클라이언트 1의 MS Tool과 클라이언트 2의 Webserver Stress Tool은 언제나 각각 150명, 100명의 동시 사용자가 가상 서버로의 요청을 생성한다. 요청 파일의 형식은 HTML, GIF, JPG, AVI, WAV의 형식이며 크기는 77Kbyte에서 960Kbyte까지 다양한 분포를 갖도록 하였다.

본 논문에서 사용한 성능 평가 방법은 기존 연구들에서 사용했던 방법과는 다소 차이가 있다. 기존 연구들[9-11]에서는 대부분 실제 웹 서버에 가해지는 요청 트래픽의 궤적을 생성한 후 이를 입력으로 하여 시뮬레이션을 수행함으로써 스케줄링 알고리즘의 성능을 평가하고 있다. 대표적인 웹 서버 요청 트래픽 궤적으로는 Rice 트레이스와 IBM 트레이스를 들 수 있다[9-11]. 하지만 시뮬레이션 평가 방법과는 달리 본 논문에서 사용한 실행-기반 성능 평가 방법에서는 미리 생성된 트레이스 데이터를 사용할 수 없다. 트레이스 데이터에 있는 요청들을 실제 HTTP 요청으로 바꾸어 웹 서버에게 전달할 수 없기 때문이다. 따라서 본 논문에서는 상용 웹 서버 부하 생성기를 사용하였으며, 부하 생성기가 인위적으로 발생시키는 HTTP 요청을 웹 서버에 실제로 전달함으로써 웹 서버의 성능을 측정하였다. 한편 기존 트레이스-구동 시뮬레이션 기법을 사용하면 트레이스 데이터에 몇 가지 종류의 컨텐츠 유형이 들어 있는가에 따라 내용-기반 스케줄링 기법의 성능 평가가 달라질 수 있다는 점이다. 트레이스 데이터가 실제 웹 서버에 가해지는 요청 트래픽을 일정 기간 동안 수집한 것이므로 현실성 있는 데이터이기는 하지만 내용-기반 스케줄러의 성능을 평가하기에는 부족한 개수의 컨텐츠 유형이 포함되어 있을 수 있기 때문이다. 따라서 본 논문에서는 부하 생성기로 하여금 인위적으로 설정된 충분한 개수의 컨텐츠 타입을 요청하게 함으로써 내용-기반 스케줄링 기법의 성능 평가를 좀 더 정확하게 할 수 있도록 하였다.

4.1 실제 서버 수에 따른 스케줄링 알고리즘간의 변화

본 실험은 결과에 채택되지 않고 부하만 발생하는 사용자 수가 총 250명이며 이들은 계속해서 서버에 요청을 하게 된다. 결과에 채택되는 사용자 수를 1에서 50명까지 증가시키면서 평균 대기시간을 측정했고, 그 값들의 평균값으로 비교를 실시했다.



(그림 9) 실제 서버 수에 따른 성능 변화

(그림 9)는 실제 서버 수에 따른 각 스케줄링 알고리즘의 성능 변화를 나타낸다. *tcpvs*라고 표기된 것은 내용 기반 요청 분배 기법을 나타내며, *s-alone*은 클러스터 웹 서버가 아닌 단일 노드 웹 서버 상에서의 성능 수치를 의미한다.

4.1.1 실제 서버 2대일 때의 알고리즘 성능 분석

실제 서버를 2대로 설정했을 때에는 가중치를 고려하지 않은 알고리즘들(*rr*, *lc*, *lblc*, *lblcr*, *tcpvs*)이 가중치를 고려한 알고리즘(*wlc*, *wrr*)보다 좋은 성능을 보이고 있다. 이는 실험에 사용된 실제 서버의 성능이 동일하기 때문에 가중치를 고려하는 연산이 오버헤드로 작용했기 때문이다. 각 알고리즘의 성능을 순서대로 나열하면 *lblcr*, *lblc*, *rr*, *lc*, *tcpvs*, *wrr*, *wlc*의 순이다. *tcpvs*는 내용에 기반하여 요청 패킷을 분배하는 알고리즘으로 이 성능 평가 항목에서는 단일 유형의 컨텐츠만을 요청하도록 했기 때문에 모든 실제 서버가 요청을 받아들일 수 있다. 따라서 요청 내용 검사 시간이 오버 헤드로 작용하고 캐쉬 친화성을 활용할 수 없으므로 성능이 좋지 못함을 알 수 있다. 하지만 가중치를 고려한 알고리즘보다는 나은 성능을 보였다. 이는 예상과는 달리 가중치 연산 오버헤드가 요청 내용 검사 오버헤드보다 높기 때문인 것으로 판단된다. *lblc*와 *lblcr*이 가장 좋은 성능을 보이는 이유는 과부하(overload)가 발생하기 전까지 한 서버에서 요청을 처리하므로 캐쉬 참조율의 이득을 보기 때문이다[2]. *rr*이 *lc* 보다 좋은 성능을 보이는 이유는 *rr*은 무조건적으로 요청을 분배하는 반면 *lc*는 각 실제 서버들의 현재 활성 접속 수와 비활성 접속 수에 따라 요청을 분배하는 상황에서 실제 서버가 2대 밖에 되지 않고 그 성능차가 거의 없으므로 최소 연결 개수 서버를 선택함으로써 얻을 수 있는 이점을 거의 확보하지 못했다고 할 수 있다. *wrr*이 *rr*보다 좋지 않은 성능을 보이는 이유는 가중치에 따른 부가적인 배분작업이 있기 때문이다. 실험에 사용된 실제 서버들의 성능이 동일하기 때문에 가중치를 고려한 *wrr*이 역효과를 나타내는 것이다. *lc*와 *wlc*의 관계도 *wrr*의 경우와 동일한 이유에서 *wlc*의 성능이 떨어지는 것이며 *wlc*가 다른 알고리즘에 비해 성능이 뒤떨어지는 이유는 가중치에 따른 부가적인 배분 오버헤드와 실제 서버들의 부하 상태를 유지해야 하는 오버헤드가 복합적으로 작용하기 때문이다.

4.1.2 실제 서버 4대일 때의 알고리즘 성능 분석

실제 서버를 4대로 설정했을 때에는 *rr*, *lc*, *wrr*, *wlc*, *lblc*, *tcpvs*, *lblcr* 의 순서로 나타났다. *tcpvs*는 실제 서버 2대일 때와 유사한 이유로 *rr* 계열이나 *lc* 계열보다 성능이 떨어진다는 것을 알 수 있다. 실제 서버 2대 일 때 성능이 가장 뛰어났던 *lblc*와 *lblcr*이 다른 알고리즘들에 비해 실제 서버 수가 4대 일 때 성능이 급격히 떨어진다는 것을 알 수 있는데, 이는 실제 서버가 4대로 증가하여 다른 스케줄링 알고리즘의 성능은 많이 좋아진 반면에 이 두 알고리즘은 2대 일 때보다 실제 서버들 사이의 부하 불균등이 더 커짐

으로 인해 성능 향상이 떨어진 것으로 판단된다.

4.1.3 실제 서버 8대일 때의 알고리즘 성능 분석

실제 서버 수를 8대로 설정했을 때에는, *rr*, *wrr*, *tcpvs*, *wlc*, *lc*, *lblc*, *lblcr* 의 순으로 나타났다. *lblc*와 *lblcr*의 성능은 실제 서버 4대일 때와 유사한 이유로 *rr* 계열보다 떨어진다는 것을 알 수 있다. 본 실험에서 주목할 점은 대부분의 *lc* 계열 알고리즘들이 *tcpvs*보다 성능이 떨어졌다는 것이다. *rr*, *wrr*, *tcpvs*은 모두 실제 서버들의 부하에 관계없이 부하를 분배한다. 반면 *lc* 계열의 알고리즘들은 실제 서버들의 현재 접속 수에 따른 부하 상태를 점검하여 실제 서버를 선택하는 방법으로, 실제 서버들의 접속 수를 확인하는 오버헤드로 인해 성능이 떨어지는 것으로 판단된다. 실제 서버 수 증가에 따른 각 스케줄링 알고리즘의 성능 향상도를 알아보면 다음 표와 같다.

〈표 2〉 실제 서버 수에 따른 성능 향상도(%)

	1대 → 2대	1대 → 4대	2대 → 4대	4대 → 8대
rr	28.5	**42.5**	19.7	**19.7**
lc	27.4	41.2	19.1	8.8
lblc	28.8	35.3	**9.15**	4.74
lblcr	**29.8**	**34.4**	**6.6**	3.43
wrr	21	37.9	21	**18.7**
wlc	**3.2**	36.8	33.6	10.8
tcpvs	27.2	34.6	10	16.1

실제 서버 수가 1대에서 4대로 증가되었을 때, *rr*이 42.5%로 가장 많은 성능 향상을 보였고 *lblcr*과 *tcpvs*가 가장 적은 34.4%, 34.6%로 가장 적은 성능 향상을 보였다. 실제 서버 수가 2대~4대, 4대~8대로 증가되었을 때, *rr*은 약 20%의 성능 향상을 보인 반면, *lblc* 계열은 각각 10%, 5%미만의 증가율을 보였다. *lblc* 계열 알고리즘이 성능 향상도가 떨어지는 이유는, 앞서 언급했듯이 실제 서버들 간의 부하 불균등이 전체 클러스터의 성능에 악영향을 미치기 때문이다. 전체 클러스터의 성능 저하 요인을 순서대로 살펴보면 다음과 같다.

- 실제 서버 2대 : 가중치 연산 오버헤드, 내용 검사 오버헤드, 부하 불균등
- 실제 서버 4대 : 부하 불균등, 내용 검사 오버헤드, 가중치 연산 오버헤드
- 실제 서버 8대 : 부하 불균등, 부하 점검 오버헤드, 내용 검사 오버헤드

서버 수가 적을 때에는 가중치 연산 오버헤드가 전체 클러스터 성능에 큰 영향을 미쳤지만, 서버 수가 증가할수록 가중치에 대한 오버헤드보다는 실제 서버들간의 부하 불균등이 전체 클러스터에 미치는 영향이 크다는 것을 알 수 있다. 이는 일정한 크기의 단일 요청을 계속적으로 받는 환경에서는 부하 불균등을 가져오는 캐쉬 친화적인 기법보다는 실제 서버간의 부하 균형을 고려하고 실제 서버 수를 증가시키는 것이 전체 클러스터의 성능 향상에 도움이 된다는 것을 의미한다.

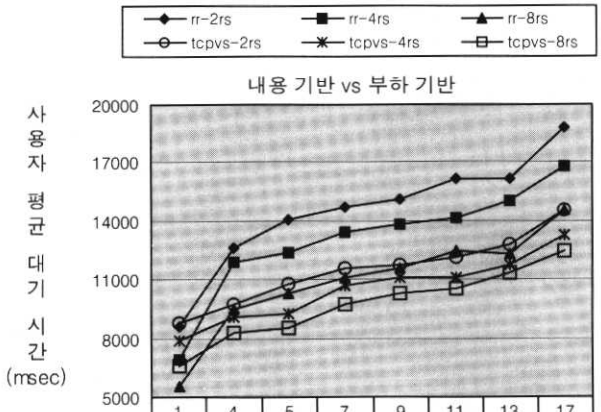
본 실험의 결과를 분석해 본 결과 다음과 같은 사실을 확인할 수 있었다. 첫째, 클러스터 내의 노드 수가 적을 때에는 캐쉬 친화적인 요청 분배 알고리즘인 *lblc* 계열의 알고리즘이 *rr* 계열의 알고리즘보다 좋은 성능을 보인다는 것이다. 이는 서버 수가 제한된 환경에서는 스케줄링 알고리즘이 클러스터 웹 서버의 성능에 큰 영향을 미친다는 것을 의미한다. 둘째, 클러스터 내에 소속된 실제 서버들의 성능이 동일한 경우에는 가중치를 고려한 알고리즘이 성능에 오히려 악영향을 미친다는 점이다. 이는 가중치를 고려한 요청 분배 알고리즘이 실제 서버들의 성능 차로 인하 오버헤드를 줄이기 위한 것이라는 측면을 고려할 때 당연한 결과라 하겠다. 셋째, 크기가 일정한 단일 요청을 받는 환경에서는 캐쉬 친화적인 내용 기반 스케줄링 알고리즘보다는 실제 서버 수를 증가시켜 전체 부하를 줄이고 실제 서버들 간의 부하 균형을 맞춰주는 것이 전체 클러스터의 성능 향상에 도움이 된다는 사실이다.

4.2 실제 서버 부하에 따른 요청 분배와 내용 기반 요청 분배에 관한 비교

이번 실험은, 내용에 기반하여 요청을 분배하는 알고리즘(*tcpvs*)과 실제 서버 부하 균등을 목적으로 하는 알고리즘인 *rr*간의 성능을 비교했다. 요청되는 내용의 종류를 1개에서 17개까지 증가시키면서 변화하는 대기 시간을 측정하였다. URL를 증가시키는 의미는 요청되는 파일들의 유형(type)을 여러 개로 두어 캐쉬 참조율을 낮게 만드는데 있다. IPVS의 여러 알고리즘 중 *rr*을 선택한 이유는 IPVS의 스케줄링 알고리즘 중 가장 뛰어난 성능을 보였기 때문이다(실제 서버들의 성능은 동일하다고 가정).

(그림 10)은 URL의 수에 따른 사용자 평균 대기시간의 변화를 나타낸다. type 수가 하나인 경우를 제외한 나머지 경우에서는 *tcpvs*가 *rr*보다 좋은 성능을 보였다. type 수가 4개로 증가하였을 때, *rr*은 평균 대기시간이 급격히 증가한데 반해 *tcpvs*는 완만한 기울기를 보인다. 이는 *tcpvs*는 캐쉬 친화적인 성향으로 요청 type의 수에 크게 영향을 받지 않는 반면, *rr*에서는 서로 다른 요청을 계속해서 받게 되면 각 실제 서버에서의 캐쉬 참조율이 급격히 떨어질 것이다. 실험 결과에서 특히 주목할만한 점은 실제 서버 수가 적음에도 불구하고 *tcpvs*가 더 좋은 성능을 보인 것인데, 이는 *tcpvs*는 패턴에 따라 실제 서버를 선택하기 때문에 캐쉬 참조율이 IPVS보다 더 좋기 때문이다. 예를 들어, 실제 서버 수가 4대이고 type 수가 17개일 경우, 최악의 경우 *rr*은

17번째에서 같은 파일의 요청이 들어오지만 *tcpvs*는 4, 5번째에서 같은 파일 요청이 들어온다. 이런 이유로 type이 5개이면서 실제 서버 2대일 때의 *rr*과 type 17개이며 실제 서버 2대일 때의 *tcpvs*가 비슷한 대기 시간을 나타내고 있다. type 개수는 3배가 넘게 차이가 나는데도 불구하고 사용자의 평균 대기시간은 비슷한 수치를 기록하고 있다.



	1	4	5	7	9	11	13	17
rr-2rs	8613	12583	14069	14744	15079	16120	16115	18817
rr-4rs	6918	11933	12401	13402	13792	14142	15032	16783
rr-8rs	5557	9494	10282	11062	11597	12421	12306	14531
tcpvs-2rs	8758	9735	10793	11539	11756	12134	12761	14546
tcpvs-4rs	7871	9112	9273	10662	11070	11112	11768	13294
tcpvs-8rs	6605	8281	8535	9756	10294	10508	11362	12424

(그림 10) 요청 type 수에 따른 성능 변화

<표 3>은 *rr* 성능 대비 *tcpvs*의 성능 향상도를 나타낸 것이다. type 개수가 4개이고 실제 서버 수가 2대일 때, *tcpvs*는 *rr*에 비해 22.6%의 성능 향상을 보였다. 전체적으로 *rr*에 비해 *tcpvs*가 약 20% 정도의 성능 향상을 보이고 있다. *tcpvs* 알고리즘이 TCP 단계에서 패킷을 검사함에도 불구하고 IP 단계의 스케줄링 알고리즘보다 좋은 성능을 보인 이유는 TCP 단계에서의 패킷 검사 오버헤드가 캐쉬 미스에 의한 오버헤드보다 작기 때문이고 *tcpvs* 알고리즘의 캐쉬 참조율이 *rr*의 캐쉬 참조율보다 좋기 때문이다.

〈표 3〉 *rr*에 대한 *tcpvs* 알고리즘의 성능 향상도

	type-4	type-7	type-9	type-13	type-17
2RS	22.6%	21.7%	22%	20.8%	22.7%
4RS	28.9%	20.4%	19.7%	21.7%	16.9%

본 실험의 결과를 분석해 본 결과 다음과 같은 사실을 확인할 수 있었다. 첫째, type 수가 많은 웹 페이지에서는 내용 기반 요청 분배 알고리즘의 성능이 뛰어나다는 사실이다. type 수가 적을 때에는 내용 기반 기법의 검사 시간 오버헤드로 인해 성능에 악영향을 미쳤지만, type 수가 증가되면 부하 균형을 위한 기법들에서는 캐쉬 미스율이 증가될 수밖에 없기 때문이다. 둘째, type 수가 증가되면 실제 서버의 수보다는 요청 분배 기법이 전체 클러스터 성능에 미치는 영향이 크다는 것을 알 수 있었다. 실제 서버를 2대에서 4대로 증가시킨 *rr*의 성능보다 실제 서버 수 2대인 *tcpvs*의 요청 분배 기법이 성능이 더 좋은 점을 보면 알 수 있다. 따라서 요청되는 웹 페이지의 형태가 다양하다면 부하 균형을 위한 요청 분배 기법보다는 내용 기반 요청 분배 기법이 클러스터의 전체 성능 향상에 도움이 될 것이다.

5. 결론 및 향후 연구 과제

본 논문에서는 클러스터 웹 서버를 구축하고 실제 서버 수와 요청되는 type 수에 따라 부하 균형을 위한 요청 분배 기법과 내용 기반 요청 분배 기법간의 성능 평가를 실시하였다. 실제 서버가 2대일 때보다는 4대일 때가 성능이 더 좋았으며, 요청 type이 하나이고 실제 서버가 2대일 때에는 캐쉬 친화적인 *lblc* 계열 알고리즘이 좋은 성능을 나타냈다. 하지만 실제 서버가 4대로 증가되었을 때에는 *rr* 계열의 알고리즘이 더 나은 성능을 보였다. 이는 *lblc* 계열 알고리즘이 캐쉬 친화적인 요인으로 인한 성능 향상보다는 전체 클러스터의 부하 균형에 악영향을 미쳤기 때문이다. 요청되는 type 수를 증가시켰을 때에는 실제 서버의 수를 증가시키는 것보다는 내용 기반 스케줄링 알고리즘을 사용하는 것이 전체 클러스터의 성능에 좋은 영향을 미친다는 것을 알 수 있었다. 이는 요청 type이 많아 캐쉬 참조율이 떨어지는 부하 균형 요청 분배 기법보다는 내용 기반 요청 분배로 인한 높은 캐쉬 참조율을 보일 수 있는 기법이 클러스터 웹 서버의 성능에 좋은 영향을 미치기 때문이다. 따라서 소규모 클러스터에서는 캐쉬 친화적인 *lblc* 계열의 알고리즘이 좋은 성능을 나타냈고, 요청 type이 많은 클러스터에서는 요청 내용을 검사하여 실제 서버를 선택하는 내용 기반 요청 분배 기법이 좋은 성능을 보인다고 할 수 있겠다.

향후 연구 과제로 더 많은 실제 서버와 요청 type에 따른 성능 평가와 내용에 기반하여 실제 서버를 선택하는 알고리즘의 다양화가 이루어져야 할 것이다. 특히, 응답 패킷의 크기가 큰 현대의 웹 페이지의 특징을 고려할 때, 내용에 기반하고 실제 서버의 부하 상태에 따라 실제 서버를 선택하는 복합적인 알고리즘의 개발이 필요할 것이다.

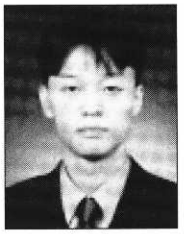
참 고 문 헌

[1] 인터넷 통계자료, "조사시기별 국내 인터넷이용자수 및 이용률", 한국 인터넷 정보 센터.

[2] Wensong Zhang and et al. Linux Virtual Server Project, http://www.linuxvirtualserver.org.

[3] EnCluster-WhitePaper, http://www.clunix.com/support.

[4] Patrick O'Rourke, Mike Keefe, "Performace Evaluation of Linux Virtual Server," http://www.linuxvirtualserver.org/Documents.html.
[5] Wensong Zhang, Shiyao Jin and Quanyuan Wu, "National Laboratory for Parallel & Distributed Processing," Linux Expo, 1999.
[6] Valeria Cardellini, Michele Colajanni, Philip, S. Yu, "Dynamic Load Balancing on Web-Server Systems," IEEE Internet Computing, May, June, 1999.
[7] Joseph Mack, The LVS-HOWTO, http://www.linux-virtualserver.org.
[8] http://www.linuxvirtualserver.org/software/ktcpvs/ktcpvs.html.
[9] Vivek, S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, Erich Nahum, Locality-Aware Request Distribution in Cluster-based Network Servers, ACM, 1998.
[10] Mohit Aron, Peter Druschel and Willy Zwaenepoel, Efficient Support for P-HTTP in Cluster-Based Web Servers, Proceedings of the USENIX Annual Technical Conference Monterey, California, USA, June, 1999.
[11] Mohit Aron, Darren Sanders, Pter Druschel, Willy Zwaenepoel, Scalable Content-aware Request Distribution in Cluster-based Network Servers, Proceedings of the 2000 Annual Usenix Technical Conference, San Diego, CA, June, 2000.
[12] Guerney Hunt, Erich Nahum and John Tracey, Enabling Content-Based Load Distribution for Scalable Services, Tech. Rep., May, 1997.
[13] http://www.linuxvirtualserver.org/software/index.html.
[14] http://www.softwareqatest.com/qatweb1.html.
[15] Webserver Stress Tool-5 Professional, http://www.paessler.com/.
[16] MS Web Application Stress-1.1, http://webtool.rte.microsoft.com/.

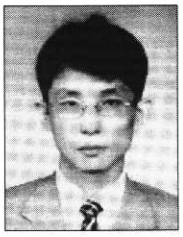


이 규 한

e-mail : smee96@mezzokorea.com
2001년 한림대학교 컴퓨터공학과(학사)
2003년 한림대학교 컴퓨터공학과(석사)
2003년~현재 메조마케팅코리아 AD-Tech 팀
관심분야 : 리눅스 클러스터 시스템, 온라인 광고 서버, 온라인 마케팅

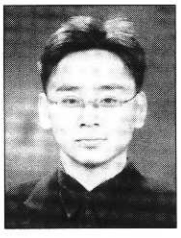
이 종 우

e-mail : jwlee44@shinbiro.com
1990년 서울대학교 컴퓨터공학과(학사)
1992년 서울대학교 컴퓨터공학과(석사)
1996년 서울대학교 컴퓨터공학과(박사)
1996년~1999년 현대전자산업 선임연구원
1999년~2002년 한림대학교 정보통신공학부 조교수
2002년~2003년 광운대학교 컴퓨터공학과 조교수
2003년~현재 (주)아이닉스 소프트 이사
관심분야 : 전산금융, 군집 컴퓨팅, 분산 병렬 시스템, 시스템 소프트웨어



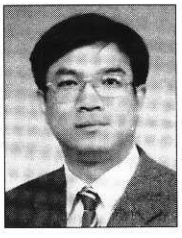
이 재 원

e-mail : jwlee@cs.sungshin.ac.kr
1990년 서울대학교 컴퓨터공학과(학사)
1992년 서울대학교 컴퓨터공학과(석사)
1998년 서울대학교 컴퓨터공학과(박사)
1999년~현재 성신여자대학교 컴퓨터정보공학부 전임강사
관심분야 : 전산금융, 인공지능, 기계학습, 자연어처리, 컴퓨터뮤직



김 성 동

e-mail : sdkim@hansung.ac.kr
1991년 서울대학교 컴퓨터공학과(학사)
1993년 서울대학교 컴퓨터공학과(석사)
1999년 서울대학교 컴퓨터공학과(박사)
2001년~현재 한성대학교 컴퓨터시스템공학과 조교수
관심분야 : 전산금융, 자연언어처리, 기계번역, 데이터마이닝, 기계학습



채 진 석

e-mail : jschae@incheon.ac.kr
1990년 서울대학교 컴퓨터공학과(학사)
1992년 서울대학교 컴퓨터공학과(석사)
1998년 서울대학교 컴퓨터공학과(박사)
1992년~1997년 서울대학교 공학연구소 조교수
1997년~1998년 한국연구정보센터 선임연구원
1998년~현재 인천대학교 컴퓨터공학과 조교수
관심분야 : 인터넷 소프트웨어, 마크업 언어, 전자 도서관