An introduction to Linux Load Balancing Cluster

작 성 자 : 서 진우 (alang@clunix.com)
소 속 : 클루닉스 기술부 / 부장
작 성 일 : 2004년 8월 25 일
Clunix Homepage : http://www.clunix.com
Private Homepage : http://www.sysmng.com

1. 소개

인터넷의 발달과 더불어 생활의 정보 전달 매체로 웹이란 수단을 선택하기 시작한지 10 년에 다되어가면서 매년 인터넷의 사용 증가는 100%를 넘고 있다. 뿐만 아니라 근래에 들어 인터넷 초기 시절의 단순한 정보 공유 매개체로서의 역할에서 사업상의 중요한 업무 시스템 구성의 중요한 요소까지 차지하게 되었다. 인터넷의 발달과 더불어 하드웨어와 소프트웨어 그리고 네트워크의 발달 속도 역시 빠른 추세로 증가 하고 있지만 기하급수적으로 늘어난 서비스 사용량을 처리하기에 보편적인 시스템의 성능과 시스템 환경의 부족함이 많이 존재하고 있다.

이에 보다 높은 가용성 (Availability) 과 확장성 (scalable)을 가진 서버의 요구가 절실히 높아지고 있다. 이에 고성능 SMP 서버를 이용하는 경우도 있지만 비 경제적인 가격 문제와 실제 시스템 병목 원인이 꼭 서버의 성능에만 있는 것이 아니기 때문에 고성능 SMP 서버만이 진정한 해결 방법은 아니다.

빠른 네트워크의 발달과 함께 분산된 서버의 자원을 하나의 작업에 이용하는 클러스링 기술이 발달하게 되고 인터넷 서비스 역시 클러스터링 기술이 부각되기 시작하였다.

클러스터란 여러 대의 컴퓨터를 네트워크를 통해 연결하여 하나의 단일 컴퓨터처럼 작동하게 하는 기술을 말한다. 클러스터는 사용 목적에 따라 구현 방법 및 소프트웨어가 달라지며, 클러스터의 기능과 운영이 달라진다. 따라서 클러스터는 그 사용 목적에 따라 베어울프와 같은 과학 계산용 클러스터와 인터넷 서비스에서 사용되는 부하분산 클러스터로 나눌 수 있다.

이런 클러스터 시스템의 장점으로 추가 확장성과 높은 가용성 그리고 비용-효율성을 두루 갖추고 있어 인터넷 서비스에 사용되는 시스템의 요구 조건을 충족 시켜 줄 수 있다.

부하 분산 클러스터는 이전부터 L4 Switch 란 하드웨어를 이용하여 시스템 구축을 해 왔지만 워낙 고가의 시스템이여서 일반 사이트에서는 적용이 힘든 방식으로 알려져 있었다. 하지만 Linux 의 커널에서 L4 Switch 의 기능과 동일하고 더 월등한 성능의 LVS(Linux virtual server) 기능을 지원함에

따라 일반 사이트에서도 손쉽게 클러스터 기술을 이용하여 고성능의 서버를 이용할 수 있게 되었다. 리눅스의 LVS 커널은 커널 2.0.x 때부터 지원 되었고, 사용 된지도 7~8 년에 이르고 있다.

이에 지속적인 안정성과 성능이 개선되어져 리눅스의 수많은 기능 중 가장 선호하는 기능으로 손꼽고 있다.

이런 리눅스 부하 분산 클러스터도 여러가지 방식이 있고 리눅스 LVS 커널과 다른 리눅스 응용 프로그램을 조합하여 고성능 뿐만 아니라 고가용성의 기능까지 탑재하게 된다.

본문 에서는 네트워크 구성에 따른 부하 분산 방식과 작업 활당 방식 그리고 고 가용성을 구현하는 방법에 대해 자세히 알아보도록 하겠다.

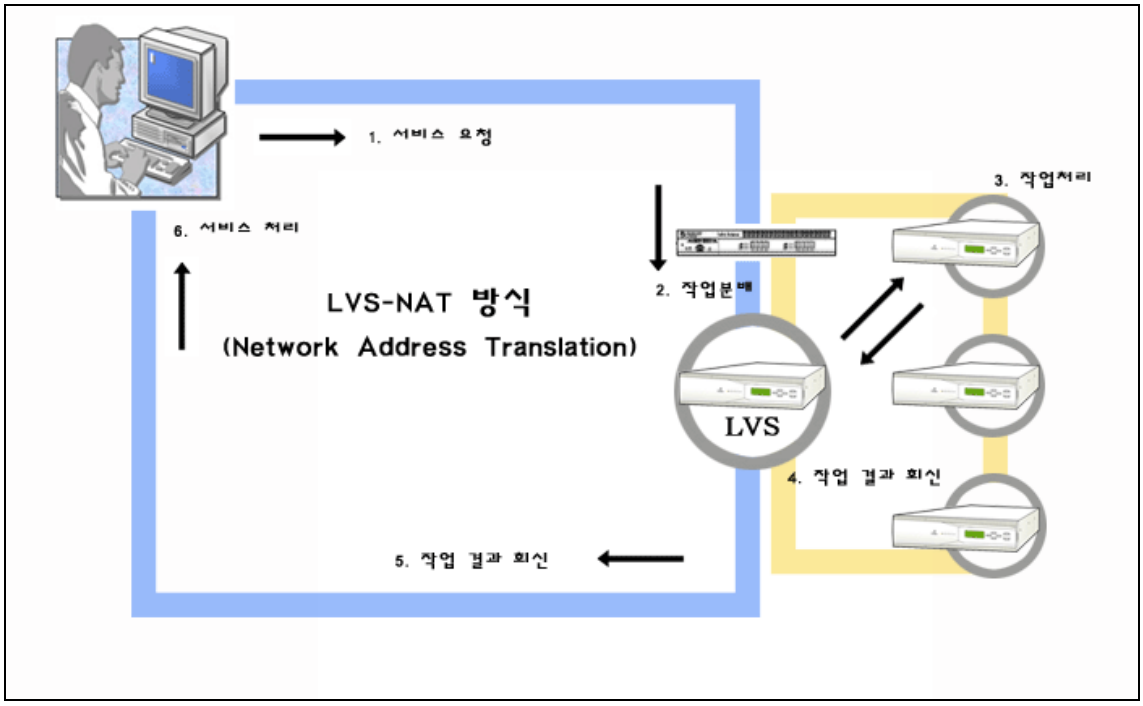
2. 네트워크 구성에 따른 부하 분산 클러스터 종류

부하 분산 클러스터를 생각하면 대부분 Load Balancer 을 앞 단에 두고 backend 의 Real Server 에 작업을 분산하는 서버 측면의 부하 분산 방식을 알고 있다. 하지만 클라이언트에 부하분산 모듈을 넣어 서버의 상황을 클라이언트가 서버로부터 얻어내어 클라이언트가 직접 서버로 분산해서 들어가는 클라이언트 측면의 부하분산 방식도 있다. (예: Clunix 의 LB Less , 버클리의 스마트 클라이언트, 보스턴 대학의 클라이언트) 서버 측면의 부하분산 방식으로는 DNS 의 호스트를 이용한 Round-Robin DNS 방법과 Reverse-Proxy 와 같은 응용프로그램 계층의 부하분산 그리고 L4 Switch 와 같은 IP 계층의 부하분산 방식이 있다. 리눅스 가상 서버 역시 IP 계층의 부하 분산 방식에 속한다.

리눅스 가상 서버에서도 네트워크 구성에 따라 주소 변환 방식 (NAT)과 IP tunneling 에 의한 방식 그리고 Direct Rounting dispatching 기술을 이용한 방식 등이 있다.

2.1 Network address translation 부하 분산 방식 (네트워크 주소 변환 방식)

IPv4 에서 IP 주소가 부족하고 보안상에 몇가지 문제가 있어서 인터넷에서 사용할 수 없는 사설 IP (10.0.0.0/255.0.0.0 , 172.16.0.0/255.240.0.0 , 192.168.0.0 /255.255.0.0)를 사용하는 경우가 많아 지고 있다. 이때 인터넷 망에서 사설망의 호스트에 접근하기 위해서 네트워크 주소 변한 (NAT) 기능이 필요하게 된다. 이런 리눅스에서의 NAT 방식의 부하분산 방식은 초기의 리눅스 IP 마스커레이딩 코드와 Steven Clarke 의 포트 포워딩 코드를 재사용 하여 구현하고 있다.

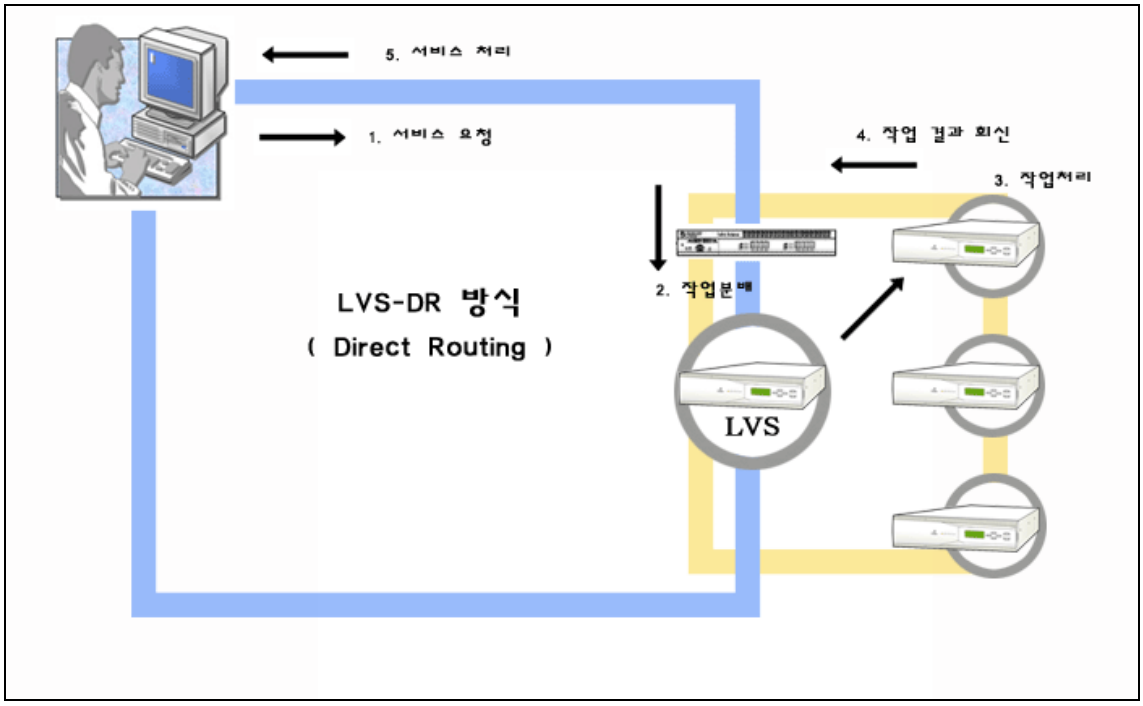


사용자가 LVS 에서 가지고 있는 VIP 를 향해 서비스 요청을 하면 LVS 에서는 그 서비스 요청된 IP 와 Port 가 가상 서버 정책에 있는지를 확인하고 있다면 이 정책에 정해진 스케줄에 따라 실제 작업 서버로 패킷을 포워딩 시키게 된다. 이때 LVS 서버는 사용자가 요청한 패킷의 네트워크 주소 부분을 공인 IP 에서 사설 IP 로 주소를 변환 시켜 같은 사설망에 존재하는 작업 서버로 보내게 된다. 작업서버는 LVS 서버로부터 패킷을 전달 받고 이에 대한 요청 처리된 내용을 다시 LVS 에 보내게 된다. 그럼 LVS 가 사설 IP 로 된 패킷을 다시 받아 다시 공인 IP 로 주소를 변환시켜서 사용자에게 돌려 보내는 방식이다.

2.2 Direct Routing 부하 분산 방식

실제 서버와 부하 분산 서버 사이에서 가상 IP 를 공유하는 방식으로 부하 분산 서버와 실제 작업 서버에 모두 동일한 가상 IP 를 가지도록 네트워크 구성해야 한다. 가상 IP 가 설정된 인터페이스를 통해 사용자의 요청을 받아 들여 이 요청이 가상 서버 정책에 적용되는 요청인지를 확인 후 적용되는 패킷이면 정책에 따라 실제 작업 서버로 패킷을 포워딩하게 된다. 이때 실제 서버 역시 별도의 Arp 캐싱을 남기지 않는 Alias 인터페이스를 통해 부하분산서버로부터 포워딩되는 패킷을 받아 들이게 된다.

만일 가상 IP 가 설정된 인터페이스에 arp caching 이 남는다고 하면 사용자가 부하 분산 서버를 통해 초기에 연결된 실제 작업와 연결되면 그 이후부터는 초기 접속한 작업 서버와만 통신이 이루어 지게 된다. 그렇기 때문에 반드시 커널 차원에서의 arp hidden patch 를 시켜 줘야 한다.



DR 방식은 패킷의 데이터 프레임 부분의 MAC 주소만을 변경하여 패킷을 포워딩 하기 때문에 물리적인 세크먼트 (같은 subnet)안에서만 동작이 가능하다. DR 방식의 arp 문제 해결 방법에 대해서는 실제 DR 구축 문서를 참조하길 바란다.

2.3 IP Tunneling 에 의한 부하 분산 방식

IP 터너링은 IP 화된 정보 (IP datagram)안에 IP 화된 정보를 감싸 넣는 (encapsulate) 기술이다. 리눅스 커널에서 지원하고 이를 이용하여 보다 WAN 을 대상으로 작업 서버 노드를 구성할수 있게 된다.

실제 가상 IP 주소로 향하는 요구 패키이 부하분산 서버로 전달 되면 부하분산 서버에서 패킷의 목적지 주소와 포트를 검사하여 가상 서버 정책과 일치 하면 스케줄링에 따라 실제 작업 서버로 전달하게 된다. 이때 패킷을 일반적인 스트링 방식으로 전달하는 것이 아닌 캣슐 형식으로 싸서 전달하게 된다. 이리 전달 되면 작업 서버에서는 감싸진 패킷을 다시 풀고 요청을 처리한 다음 실제 서버의 라우팅 테이블에 따라 사용자에게 직접 결과를 돌려주는 방식이다.

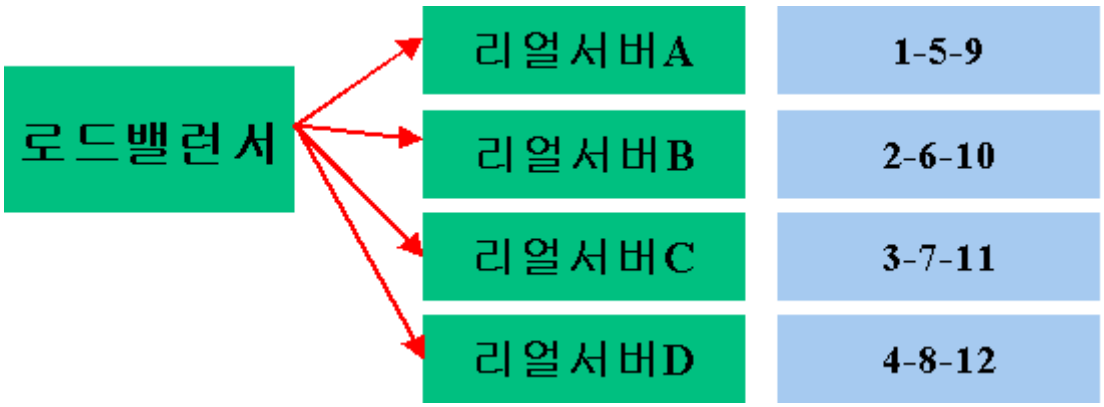
IP 터너링 방식은 DR 방식의 물리적인 세그먼트의 영향을 받지 않고 어디든지 작업을 분산 할 수 있는 장점이 있다. 그래서 거의 무한한 확장성을 가지게 된다. 하지만 모든 서버가 IP 터널링 전송 규약을 지원해야 하기에 지원하지 못하는 OS 에서는 문제가 발생하게 된다. .

2.4 작업 활당 방식 (Scheduling Algorithms)

클러스터로부터 서버를 선택하기위한 네 개의 작업 할당 방식들(scheduling algorithms)이 준비되어있다: Round-Robin, 가중치가 있는(weighted) Round-Robin, 최소-접속(Least-Connection) 그리고 가중치가 있는 최소-접속의 방식들이다. 처음의 두 방식들은 서버에대한 어떠한 부하 정보(load information)도 가지고 있지 않기 때문에 자명(self-explanatory)하다. 뒤의 두 방식들은 각 서버에대한 활동 접속 수(active connection number)를 세어 이러한 접속 수에 의해 그들의 부하를 추정한다.

2.4.1 Round-Robin Scheduling (라운드 로빈 스케쥴링)

말그대로 라운드-로빈 방식을 이용해 네트웍 연결을 서로 다른 서버에 연결하는 것을 말한 다. 이경우 실제서버의 연결갯수나 반응시간등은 고려를 하지 않는다. 그렇지만 약간의 차이 가 있다. 라운드 로빈 DNS 는 단일한 도메인을 서로 다른 IP 로 해석을 하지만, 스케쥴링의 기초는 호스트 기반이며 캐싱때문에 알고리즘을 효율적으로 사용하기 힘들다. 그래서 실제 서버사이에 동적인 부하 불균형이 심각해 질수 있다. 가상 서버의 스케쥴링 기초는 네트웍 기반이며 라운드 로빈 DNS 에 비해 훨씬 더 훌륭하다.



2.4.2 Weighted Round-Robin Scheduling (가중치기반 라운드 로빈 스케쥴링)

가중치기반 라운드 로빈 스케쥴링은 실제 서버에 서로 다른 처리 용량을 지정할 수 있다. 각 서버에 가중치를 부여할 수 있으며, 여기서 지정한 정수값을 통해 처리 용량을 정한다. 기본 가중치는 1 이다. 예를 들어 실제 서버가 A,B,C 이고 각각의 가중치가 4,3,2 일 경우 스케쥴링 순서는 ABCABCABA 가 된다.

가중치가 있는 라운드 로빈 스케쥴링을 사용하면 실제 서버에서 네트웍 접속을 셀 필요가 없고 동적 스케쥴링 알고리즘보다 스케쥴링의 과부하가 적으므로 더 많은 실제 서버를 운영 할 수 있다. 그러나 요청에 대한 부하가 매우 많을 경우 실제 서버사이에 동적인 부하 불균 형 상태가 생길 수 있다.

라운드 로빈 스케쥴링은 가중치기반 라운드 로빈 스케쥴링의 특별한 한 종류이며 모든 가중 치가 동일한 경우이다. 가상 서버의 규칙을 변경하고나서 스케쥴링 순서를 생성하는데는 거의 과부하가 걸리지 않으며 실제 스케쥴링에 어떠한 과부하도 추가하지 않는다. 그러므로 라운드 로빈 스케쥴링만 단독으로 실행하는 것은 불필요한 일이다.

2.4.3 Least-Connection Scheduling (최소 접속 스케쥴링)

최소 접속 스케쥴링은 가장 접속이 적은 서버로 요청을 직접 연결하는 방식을 말한다. 각 서버에서 동적으로 실제 접속한 숫자를 세어야하므로 동적인 스케쥴링 알고리즘중의 하나이다. 비슷한 성능의 서버로 구성된 가상 서버는 아주 큰 요구가 한 서버로만 집중되지 않기 때문에, 접속부하가 매우 큰 경우에도 아주 효과적으로 분산을 한다.
가장 빠른 서버에서 더 많은 네트웍 접속을 처리할 수 있다. 그러므로 다양한 처리 용량을 지닌 서버로 구성했을 경우에도 훌륭하게 작동 한다는 것을 한눈에 알 수 있을 것이다. 그렇지만 실제로는 TCP 의 TIME_WAIT 상태때문에 아주 좋은 성능을 낼수는 없다. TCP 의 TIME_WAIT 는 보통 2 분이다. 그런데 접속자가 아주 많은 웹 사이트는 2 분동안에 몇천개의 접속을 처리해야 할 경우가 있다. 서버 A 는 서버 B 보다 처리용량이 두배일 경우 서버 A 는 수천개의 요청을 처리하고 TCP 의 TIME_WAIT 상황에 직면하게 된다. 그렇지만 서버 B 는 몇천개의 요청이 처리되기만을 기다리게 된다. 그래서 최소 접속 스케쥴링을 이용할 경우 다양한 처리용량을 지난 서버로 구성되었을 경우 부하분산이 효율적으로 되지 못할 수 있는 것이다.

2.4.4 Weighted Least-Connection Scheduling (가중치 기반 최소 접속 스케쥴링)

가중치 기반 최소 접속 스케쥴링은 최소 접속 스케쥴링의 한 부분으로서 각각의 실제 서버에 성능 가중치를 부여할 수 있다. 언제라도 가중치가 높은 서버에서 더 많은 요청을 받을 수 있다. 가상 서버의 관리자는 각각의 실제 서버에 가중치를 부여할 수 있다. 가중치의 비율인 실제 접속자수에 따라 네트웍 접속이 할당된다. 기본 가중치는 1 이다. 가중치가 있는 최소 접속 스케쥴링은 다음과 같이 작동한다: n 개의 실제 서버가 있는 경우 각 서버 i 는 가중치 Wi (i = 1, ... , n)를 가진다고 가정하자. 서버 i 의 활동 접속(active connection)은 Ci (i = 1, ... , n)이고 모든_접속 은 Ci (i = 1, ... , n)의 합이다. 서버 j 로 가는 넷트웍 접속은 아래와 같다.

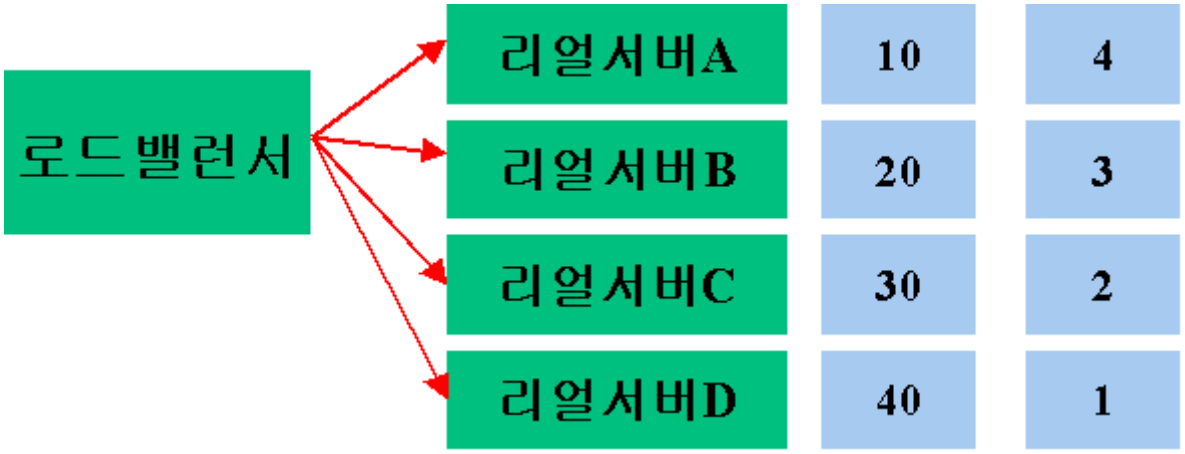
(Cj/ALL_CONNECTIONS)/Wj = min { (Ci/ALL_CONNECTIONS)/Wi } (i=1,..,n)

이 비교에서 ALL_CONNECTIONS 는 상수이므로 Ci 를 모든_접속 으로 나눠줄 필요가 없다.그러면 다음과

같이 최적화될 것이다.

Cj/Wj = min { Ci/Wi } (i=1,..,n)

가중치가 있는 최소 접속 스케쥴링 알고리즘은 최소 접속 스케쥴링 알고리즘에 비해 부가적인 배분작업이 필요하다. 서버들이 같은 처리 용량을 가졌을때는 작업 할당의 간접 비용을 최소화하기위해 최소 접속 스케쥴링과 가중치가 있는 최소 접속 스케쥴링 알고리즘 둘 다 사용할 수 있다.



3. 고 가용성

중요한 상업적 응용 분야가 인터넷으로 점점 더 많이 옮아옴에 따라 가용성이 높은 서버들을 제공하는 것이 점점 더 중요해지고 있다. 클러스터화된 시스템의 장점중 하나는 하드웨어와 소프트웨어의 여유분이 있다는 것이다. 높은 가용성은 노드(node)나 데몬(deamon)의 장애들(failures)이나 시스템의 재구성(reconfiguring)을 적절히 감지해 클러스터에 남아있는 노드로 작업 부하(workload)를 이전하는 것으로 이룰 수 있다. 가상 서버의 높은 가용성을 현재 "mon"과 "fake" 소프트웨어를 사용해 제공한다. "mon"은 넷트웍 서비스 가용성과 서버 노드를 모니터할 수 있는 범용 자원 모니터링 시스템(general-purpose resource monitoring system)이다. "fake"는 ARP 속이기(spoofing)를 사용해 IP 를 이전하는 소프트웨어다.

서버의 장애 극복(failover)은 다음과 같이 제어한다: 부하 분산기(load balancer)에 클러스터의 서비스 데몬들과 서버 노드들을 모니터하기 위한 "mon" 데몬이 운영된다. 매 t 초마다 서버 노드들이 살아있는지 감지하도록 fping.monitor 를 구성하고 매 m 분마다 모든 노드의 서비스 데몬들을 감지하기 위해 연관된 모니터 프로그램 역시 구성한다. 예를 들어 http 서비스를 점검하기위해 http.monitor 를, ftp 서비스를 위해 ftp.monitor 를 또, 기타 다른 모니터 프로그램을 사용할 수 있다. 서버 노드나 데몬이 새로 사용 불능 상태(down)가 되거나 사용 가능 상태(up)가 되면 가상 서버 테이블에 규칙을 제거하거나 추가하기위한 경보가 작성된다. 따라서 부하 분산기가 서비스 데몬이나 서버를 제거하거나 복구되었을 때 서비스에 추가하는 것을 자동으로 할 수 있다.

이제 부하 분산기는 장애가 있으면 전체 시스템에 영향을 미치는 곳(single failure point)이 되었다. 부하 분산기의 장애를 방지하기 위해 부하 분산기의 백업(backup) 서버를 설치한다. 부하 분산기에 장애가 있을 때 부하 분산기의 IP 주소를 백업 서버로 이전하기위해 "fake" 소프트웨어가 사용된다. 또, 백업 서버에서는 부하 분산기의 상태를 검사하여 "fake"를 실행하거나 실행 중단하기위해 "mon"을 사용한다. "mon" 데몬은 백업 서버에서도 또한 운용되어 백업 서버가 클러스터의 현재 상태를 알고 있어야 한다. 주 부하 분산기에 장애가 있을 때 백업 서버가 그 IP 주소(들)를 인수하여 서비스들을 계속 제공한다. 그러나 해쉬 테이블의 성립된 접속(established connection)은 소멸되며 클라이언트들은 그들의 요구들(requests)을 다시 보내야할 필요가 있다.

4 문제점

현재의 로드 밸런서 시스템은 일반적으로 5 분의 주기로 현재의 Load 를 감시 하여 스케줄링에 반영하나 이는 5 분이내의 급격한 클라이언트로부터의 요청을 처리하는데 있어서의 문제점이 있다. 이는 또한 급격한 부하의 증가로 인하여 서버의 다운시 다운된 서버의 Load 를 다른 서버로 전환하기 때문에 다른 서버의 다운을 야기 할 수 있으므로 최악의 경우 웹 서버 전체의 다운을 가져올수 있다.

현재의 로드 밸런서 시스템에 있어서의 가중치에 대한 고려사항은 단지 전체 접속자 수에 대한 각각의 리얼 서버의 접속자 수로서 서버의 성능 및 현재 서버의 부하에 대한 반영이 불가능하다.

서버의 성능은 항상 같을 수 없으므로 각각의 가중치를 부여 하여야 하지만 이에 대한 방안이 없다.

5. LVS 의 장단점

LVS 시스템은 별도의 하드웨어 추가 없이 기존의 네트워크 스위치와 서버들 만으로 대규모 인넷 서비스 처리 요청에 대해 훌륭한 해답을 제시한다.
그러나 LVS 와 관련 프로젝트는 몇가지 문제점이 있다. 우선 LVS 는 사용이 쉽지 않다. 처리 용량이나 가중치의 설정이라거나, 서버 추가나 제거시 관리자가 이를 위해 일일이 시스템을 재설정해야 하는 문제가 있다. 또한 LVS 의 자매 프로젝트인 HA 는 고가용성 요구에 대한 해결책으로 보기에는 한계가 있다. 또, 단일 클러스터 내에 여러 개의 로드 밸런서의 설정이나 서브 클러스터의 구성 등, 기능성과 접근성에서 기존 L4 스위치와 비교하기엔 무리가 있다.