LinuxDirector: A Connection Director for Scalable Internet Services

Wensong Zhang, Shiyao Jin, Quanyuan Wu
National Laboratory for Parallel & Distributed Processing
Changsha, Hunan 410073, China
wensong@LinuxVirtualServer.org
http://www.LinuxVirtualServer.org

Abstract

LinuxDirector is a connection director that supports load balancing among multiple Internet servers, which can be used to build scalable Internet services based on clusters of servers. LinuxDirector extends the TCP/IP stack of Linux kernel to support three IP load balancing techniques, VS/NAT, VS/TUN and VS/DR. Four scheduling algorithms have been implemented to assign connections to different servers. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately. This paper describes the design and implementation of LinuxDirector and presents several of its features including scalability, high availability and connection affinity.

KEYWORDS

Internet services, server clustering, load balancing, high availability

1. Introduction

With the explosive growth of the Internet and its increasingly important role in our lives, traffic on the Internet is increasing dramatically, which has been growing at over 100% annual rate. More and more sites are often unable to serve their workload, particulary during peak periods of activity. Some of them have already received tens of millions hits per day. The long delay of services will lower the quality of services. With the increasing number of users and the increasing workload, companies often worry about how systems grow over time. Companies are moving their mission-critical applications on the Internet, and any stop of services causes companies to loose customers and money. Therefore, the demand for hardware and software solution to support highly scalable and highly available services is growing urgently. The requirements can be summarized as follows:

- **Scalability**, when the load offered to the service increases, system can be scaled to meet the requirement.
- **24x7 availability**, the service as a whole must be available 24x7, despite of transient partial hardware and software failures.
- **Cost-effectiveness**, the whole system must be economical to afford and expand.
- **Manageability**, although the whole system may be physically large, it should be easy to manage.

A single server is usually not sufficient to handle this aggressively increasing load. The server upgrading process is complex, and the server is a single point of failure. The higher end the server is upgraded to, the much higher cost we have to pay.

Clusters of servers, connected by a fast network, are emerging as a viable architecture for building a high-performance and highly available server. This type of loose-coupled architecture is more scalable, more cost-effective and more reliable than a single processor system or a tightly coupled multiprocessor system. However, there are challenges to provide transparency, efficiency, scalability and high availability of parallel services in the cluster.

LinuxDirector [18] is our solution to the requirements. LinuxDirector is a connection director that supports load balancing among multiple Internet servers, which can be used to build scalable Internet services based on clusters of servers. Prototypes of LinuxDirector have already been used to build many sites of heavy load in the Internet.

LinuxDirector directs network connections to the different servers according to scheduling algorithms and makes parallel services of the cluster to appear as a virtual service on a single IP address. Client applications interact with the cluster as if it were a single high-performance and highly available server. The clients are not affected by interaction with the cluster and do not need modification. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately.

The remainder of the paper is organized as follows: In Section 2, we discuss the related works. In Section 3, we describe three require dispatching techniques and their working principles, and also discuss their advantages and disadvantages. In Section 4, we describe the four scheduling algorithms that have been developed for LinuxDirector. In Section 5, we describe the high availability issue of LinuxDirector. In Section 6, we describe how connection affinity is handled in LinuxDirector. In Section 7, we present some big LinuxDirector application that we have known. Finally, conclusion and future work appear in Section 8.

2. Related Works

In the client/server applications, one end is the client, the other end is the server, and there may be a proxy in the middle. Based on this scenario, we can see that there are many ways to dispatch requests to a cluster of servers in the different levels. Existing request dispatching techniques can be classified into the following categories:

- **The client-side approach**

  Berkeley's Smart Client [17] suggests that the service provide an applet running at the client side. The applet makes requests to the cluster of servers to collect load information of all the servers, then chooses a server based on that information and forwards requests to that server. The applet tries other servers when it finds the chosen server is down. However, these client-side approaches are not client-transparent, they requires modification of client applications, so they cannot be applied to all TCP/IP services. Moreover, they will potentially increase network traffic by extra querying or probing.

- **The server-side Round-Robin DNS approach**

  The NCSA scalable web server is the first prototype of a scalable web server using the Round-Robin DNS approach [12, 13, 5]. The RRDNS server maps a single name to the different IP addresses in a round-robin manner so that the different clients will access the different servers in the cluster for the ideal situation and load is distributed among the servers. However, due to the caching nature of clients and hierarchical DNS system, it easily leads to dynamic load imbalance among the servers, thus it is not easy for a server to handle its peak load. The TTL(Time To Live) value of a name mapping cannot be well chosen at RR-DNS, with small values the RR-DNS will be a bottleneck, and with high values the dynamic load imbalance will get even worse. Even the TTL value is set with zero, the scheduling granularity is per host, different client access pattern may lead to dynamic load imbalance, because some clients (such as a proxy server) may pull lots of pages from the site, and others may just surf a few pages and leave. Futhermore, it is not so reliable, when a server node fails, the clients who maps the name to the IP address will find the server is down, and the problem still exists even if they press ”reload” button in the browsers.

- **The server-side application-level scheduling approach**

  EDDIE [6] , Reverse-proxy [15] and SWEB [4] use the application-level scheduling approach to build a scalable web server. They all forward HTTP requests to different web servers in the cluster, then get the results, and finally return them to the clients. However, this approach requires to establish two TCP connections for each request, one is between the client and the load balancer, the other is between the load balancer and the server, the delay is high. The overhead of dealing HTTP requests and replies in the application-level is high. Thus the application-level load balancer will be a new bottleneck soon when the number of server nodes increases.

- **The server-side IP-level scheduling approaches**

  Berkeley's MagicRouter [3] and Cisco's LocalDirector [2] use the Network Address Translation approach similar to the NAT approach used in LinuxDirector. However, the MagicRouter didn't survive to be a useful system for others, the LocalDirector is too expensive and only supports part of TCP protocol.

  IBM's TCP router [8] uses the modified Network Address Translation approach to build scalable web server on IBM scalable Parallel SP-2 system. The TCP router changes the destination address of the request packets and forwards the chosen server, that server is modified to put the TCP router address instead of its own address as the source address in the reply packets. The advantage of the modified approach is that the TCP router avoids rewriting of the reply packets, the disadvantage is that it requires modification of the kernel code of every server in the cluster. NetDispatcher [9] , the successor of TCP router, directly forwards packets to servers that is configured with router address on non arp-exported interfaces. The approach, similar to the VS/DR in LinuxDirector, has good scalability, but NetDispatcher is a very expensive commercial product.

ONE-IP [7] requires that all servers have their own IP addresses in a network and they are all configured with the same router address on the IP alias interfaces. Two dispatching techniques are used, one is based on a central dispatcher routing IP packets to different servers, the other is based on packet broadcasting and local filtering. The advantage is that the rewriting of response packets can be avoided. The disadvantage is that it cannot be applied to all operating systems because some operating systems will shutdown the network interface when detecting IP address collision, and the local filtering also requires modification of the kernel code of server.

3. IP Load Balancing Techniques

Since the IP load balancing techniques have good scalability, LinuxDirector extends the TCP/IP stack of Linux kernel to support three IP load balancing techniques, VS/NAT, VS/TUN and VS/DR. The box running LinuxDirector act as a load balancer of network connections from clients who know a single IP address for a service, to a set of servers that actually perform the work. In general, real servers are idential, they run the same service and they have the same set of contents. The contents are either replicated on each server's local disk, shared on a network file system, or served by a distributed file system. We call data communication between a client's socket and a server's socket **connection**, no matter it talks TCP or UDP protocol. The following subsections describe the working principles of three techniques and their advantages and disadvantages.

3.1. Virtual Server via NAT

Due to the shortage of IP address in IPv4 and some security reasons, more and more networks use private IP addresses which cannot be used on the Internet. The need for network address translation arises when hosts in internal networks want to access or to be accessed on the Internet. Network address translation relies on the fact that the headers of packets can be adjusted appropriately so that clients believe they are contacting one IP address, but servers at different IP addresses believe they are contacted directly by the clients. This feature can be used to build a virtual server, i.e. parallel services at the different IP addresses can appear as a virtual service on a single IP address.

The architecture of virtual server via NAT is illustrated in Figure 1. The load balancer and real servers are interconnected by a switch or a hub. The workflow of VS/NAT is as follows: When a user accesses a virtual service provided by the server cluster, a request packet destined for virtual IP address (the IP address to accept requests for virtual service) arrives at the load balancer. The load balancer examines the packet's destination address and port number, if they are matched for a virtual service according to the virtual server rule table, a real server is selected from the cluster by a scheduling algorithm, and the connection is added into the hash table which records connections. Then, the destination address and the port of the packet are rewritten to those of the selected server, and the packet is forwarded to the server. When an incoming packet belongs to an established connection, the connection can be found in the hash table and the packet will be rewritten and forwarded to the right server. When response packets come back, the load balancer rewrites the source address and port of the packets to those of the virtual service. When a connection terminates or timeouts, the connection record will be removed in the hash table.

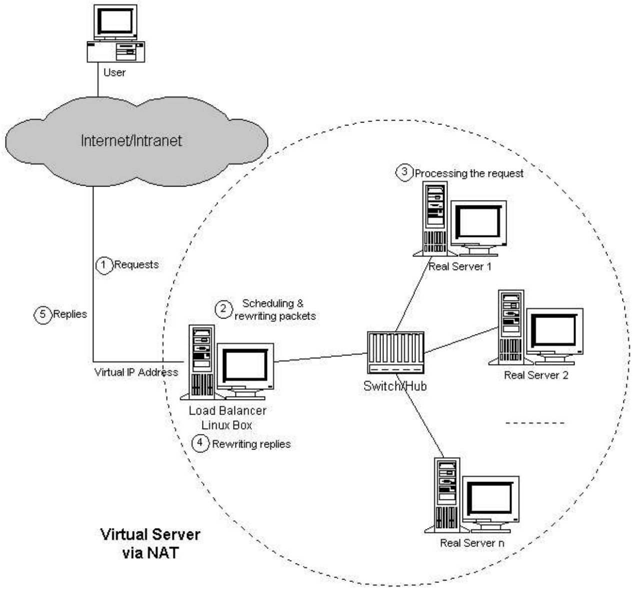

Figure 1. Architecture of a virtual server via NAT

3.2. Virtual Server via IP Tunneling

IP tunneling (IP encapsulation) is a technique to encapsulate IP datagram within IP datagram, which allows datagrams destined for one IP address to be wrapped and redirected to another IP address. This technique can be used to build a virtual server that the load balancer tunnels the request packets to the different servers, and the servers process the requests and return the results to the clients directly, thus the service can still appear as a virtual service on a single IP address.

The architecture of virtual server via IP tunneling is illustrated in Figure 2. The real servers can have any real IP address in any network, and they can be geographically

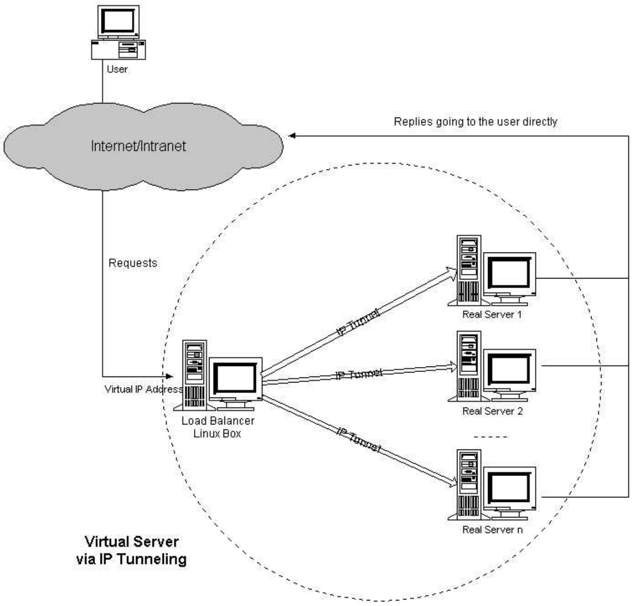

Figure 2. Architecture of a virtual server via IP tunneling

distributed, but they must support IP tunneling protocol and they all have one of their tunnel devices configured with VIP.

The workflow of VS/TUN is the same as that of VS/NAT. In VS/TUN, the load balancer encapsulates the packet within an IP datagram and forwards it to a dynamically selected server. When the server receives the encapsulated packet, it decapsulates the packet and finds the inside packet is destined for VIP that is on its tunnel device, so it processes the request, and returns the result to the user directly.

3.3. Virtual Server via Direct Routing

This IP load balancing approach is similar to the one implemented in IBM's NetDispatcher. The architecture of VS/DR is illustrated in Figure 3. The load balancer and the real servers must have one of their interfaces physically linked by an uninterrupted segment of LAN such as a HUB/Switch. The virtual IP address is shared by real servers and the load balancer. All real servers have their loopback alias interface configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets.

The workflow of VS/DR is the same as that of VS/NAT or VS/TUN. In VS/DR, the load balancer directly routes a packet to the selected server, i.e. the load balancer simply changes the MAC address of data frame to that of the server and retransmits it on the LAN. When the server receives the forwarded packet, the server finds that the packet is for the address on its loopback alias interface and processes the request, finally returns the result directly to the user. Note that real servers' interfaces that are configured with virtual IP address should not do ARP response, otherwise there would be a collision if the interface to accept incoming traffic for VIP and the interfaces of real servers are in the same network.

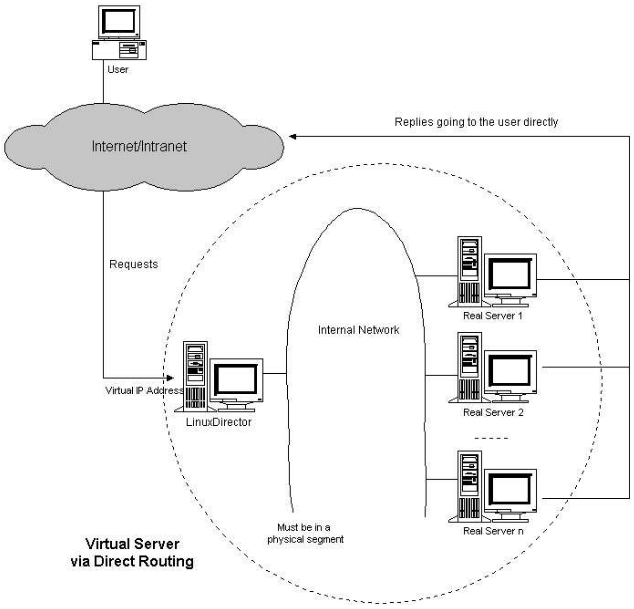

Figure 3. Architecture of a virtual server via direct routing

Table 1. the comparison of VS/NAT, VS/TUN and VS/DR

	VS/NAT	VS/TUN	VS/DR
Server	any	tunneling	non-arp device
server network	private	LAN/WAN	LAN
server number	low (10 20)	high (100)	high (100)
server gateway	load balancer	own router	own router

3.4. Advantages and Disadvantages

The characteristics of three IP load balancing techniques are summarized in Table 1.

- **Virtual server via NAT**

  In VS/NAT, real servers can run any operating system that supports TCP/IP protocol, and only one IP address is needed for the load balancer and private IP addresses can be used for real servers.

  The disadvantage is that the scalability of VS/NAT is limited. The load balancer may be a bottleneck of the whole system when the number of server nodes increases up to 20, because both request and response packets need to be rewritten by the load balancer. Supposing the average length of TCP packets is 536 Bytes and the average delay of rewriting a packet is around 60us on the Pentium processor (this can be reduced a little by using of faster processor), the maximum throughout of the load balancer is 8.93 Mbytes/s. The load balancer can schedule 15 servers if the average throughout of real servers is 600KBytes/s.

- **Virtual server via IP tunneling**

  For most Internet services (such as web service) that request packets are often short and response packets usually carry large amount of data, a VS/TUN load balancer may schedule over 100 general real servers and it won't be the bottleneck of the system, because the load balancer just directs requests to the servers and the servers reply the clients directly. Therefore, VS/TUN has good scalability. VS/TUN can be used to build a virtual server that takes huge load, extremely good to build a virtual proxy server because when the proxy servers receive requests, they can access the Internet directly to fetch objects and return them to the clients directly.

  However, VS/TUN requires servers support IP Tunneling protocol. This feature has been tested with servers running Linux. Since the IP tunneling protocol is becoming a standard of all operating systems, VS/TUN should be applicable to servers running other operating systems.

- **Virtual Server via Direct Routing**

  Like VS/TUN, a VS/DR load balancer processes only the client-to-server half of a connection, and the response packets can follow separate network routes to the clients. This can greatly increase the scalability of virtual server.

  Compared to VS/TUN, VS/DR doesn't have tunneling overhead , but it requires the server OS has loopback alias interface that doesn't do ARP response, the load balancer and each server must be directly connected to one another by a single uninterrupted segment of a local-area network.

3.5. Implemention Issues

We have modified the TCP/IP stack inside Linux kernel 2.0 and 2.2 respectively, in order to support the above three IP load balancing technologies. The system implementation of LinuxDirector is illustrated in Figure 4. The "VS Schedule & Control Module" is the main module of LinuxDirector, it hooks two places at IP packet traversing inside kernel in order to grab/rewrite IP packets to support IP load balancing. It looks up the "VS Rules" hash table for new connections, and checks the "Connection Hash Table" for established connections. The "IPVSADM" user-space program is to administrator virtual servers, it uses setsockopt function to modify the virtual server rules inside the kernel, and read the virtual server rules through /proc file system.

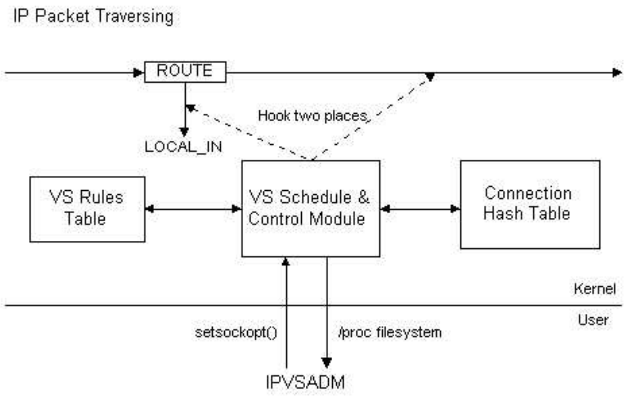

Figure 4. Implementation of LinuxDirector

The connection hash table is designed to hold millions of concurrent connections, and each connection entry only occupies 128 bytes effective memory in the load balancer. For example, a load balancer of 256 Mbytes free memory can have two million concurrent connections. The hash table size can be adapted by users according to their applications, and the client $< protocol, address, port >$ is used as hash key so that hash collision is very low. Slow timer is ticked every second to collect stale connections.

LinuxDirector implements ICMP handling for virtual services. The incoming ICMP packets for virtual services will be forwarded to the right real servers, and outgoing ICMP packets from virtual services will be altered and sent out correctly. This is important for error and control notification between clients and servers, such as the MTU discovery.

LinuxDirector implements three IP load balancing techniques. They can be used for different kinds of server clusters, and they can also be used together in a single cluster, for example, packets are forwarded to some servers through VS/NAT method, some servers through VS/DR, and some geographically distributed servers through VS/TUN.

4. Connection Scheduling

We have implemented four scheduling algorithms for selecting servers from the cluster for new connections:

Round-Robin, Weighted Round-Robin, Least-Connection and Weighted Least-Connection. The first two algorithms are self-explanatory, because they don't have any load information about the servers. The last two algorithms count active connection number for each server and estimate their load based on those connection numbers.

4.1. Round-Robin Scheduling

Round-robin scheduling algorithm directs the network connections to the different servers in the round-robin manner. It treats all real servers as equals regardless of number of connections or response time. Although the round-robin DNS works in this way, there are quite different. The round-robin DNS resolves the single domain to the different IP addresses, the scheduling granularity is per host, and the caching of DNS hinder the algorithm take effect, which will lead to significant dynamic load imbalance among the real servers. The scheduling granularity of virtual server is per connection, and it is more superior to the round-robin DNS due to fine scheduling granularity.

4.2 Weighted Round-Robin Scheduling

The weighted round-robin scheduling can treat the real servers of different processing capacities. Each server can be assigned a weight, an integer that indicates its processing capacity, the default weight is 1. The WRR scheduling works as follows:

Assuming that there is a list of real servers $S = \{S_0, S_1, ..., S_{n-1}\}$, an index i is the last selected server in S, the variable cw is current weight. The variable i is initialized to -1 and cw is initialized to zero. If all $W(S_i) = 0$, there are no available servers, all the connection for virtual server are dropped.

```
while (1) {
  i = (i + 1) mod n;
  if (i == 0) {
     cw = cw - 1;
     if (cw <= 0) {
       set cw the maximum weight of S;
       if (cw == 0) return NULL;
     }
  }
  if (W(Si) >= cw) return Si;
}
```

In the WRR scheduling, all servers with higher weights receives new connections first and get more connections than servers with lower weights, servers with equal weights get an eaqual distribution of new connections. For example, the real servers A,B,C have the weights 4,3,2 respectively, then the scheduling sequence can be AABABCABC in a scheduling period (mod sum(Wi)). The WRR is efficient to schedule request, but it may still lead to dynamic load imbalance among the real servers if the load of requests vary highly.

4.3 Least-Connection Scheduling

The least-connection scheduling algorithm directs network connections to the server with the least number of active connections. This is one of dynamic scheduling algorithms, because it needs to count active connections for each server dynamically. At a virtual server where there is a collection of servers with similar performance, the least-connection scheduling is good to smooth distribution when the load of requests vary a lot, because all long requests will not be directed to a single server.

At a first look, the least-connection scheduling can also perform well even if servers are of various processing capacities, because the faster server will get more network connections. In fact, it cannot perform very well because of the TCP's TIME_WAIT state. The TCP's TIME_WAIT is usually 2 minutes, in which a busy web site often get thousands of connections. For example, the server A is twice as powerful as the server B, the server A has processed thousands of requests and kept them in the TCP's TIME_WAIT state, but but the server B is slow to get its thousands of connections finished and still receives new connections. Thus, the least-connection scheduling cannot get load well balanced among servers with various processing capacities.

4.4 Weighted Least-Connection Scheduling

The weighted least-connection scheduling is a superset of the least-connection scheduling, in which a performance weight can be assigned to each server. The servers with a higher weight value will receive a larger percentage of active connections at any time. The virtual server administrator can assign a weight to each real server, and network connections are scheduled to each server in which the percentage of the current number of active connections for each server is a ratio to its weight.

The weighted least-connections scheduling works as follows: supposing there is n real servers, each server i has weight W_i (i=1,..,n) and active connections C_i (i=1,..,n), all connection number S is the sum of C_i (i=1,..,n), the network connection will be directed to the server j, in which

$$(C_j/S)/W_j = min\{(C_i/S)/W_i\} \text{ (i=1,..,n)}$$

Since the S is a constant in this lookup, there is no need to divide C_i by S, it can be optimized as

$$C_j/W_j = min\{C_i/W_i\} \text{ (i=1,..,n)}$$

Since there is no floats in Linux kernel mode, the comparison of $C_j/W_j > C_i/W_i$ is changed to $C_j * W_i > C_i * W_j$ because all weights are larger than zero.

5. High Availability

As more and more mission-critical applications move on the Internet, providing highly available services becomes increasingly important. One of the advantages of a clustered system is that it has hardware and software redundancy. High availability can be provided by detecting node or daemon failures and reconfiguring the system appropriately so that the workload can be taken over by the remaining nodes in the cluster.

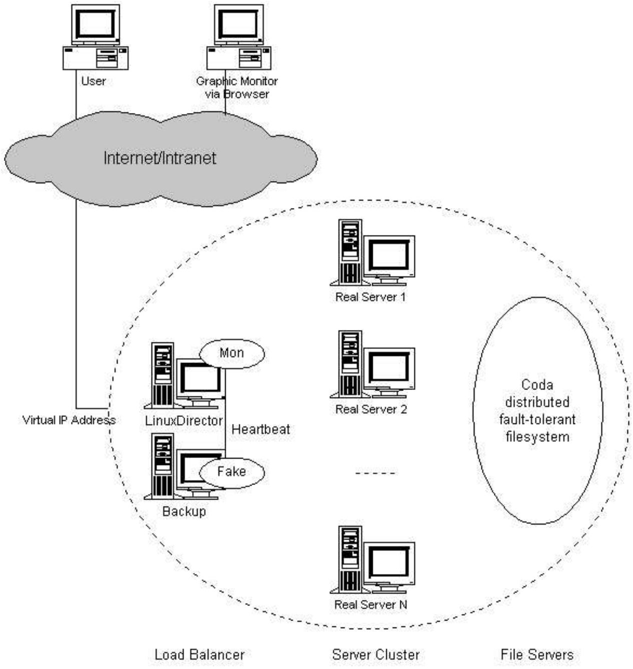

Figure 5. High availability in LinuxDirector

The high availability of LinuxDirector is now provided by using of ”mon” [16], ”heartbeat” [14] and ”fake” [11]. The ”mon” is a general-purpose resource monitoring system, which can be used to monitor network service availability and server nodes. The ”heartbeat” provides heartbeats (periodical communication) among server nodes. The ”fake” is IP take-over software by using of ARP spoofing (gratutious ARP). Figure 5 illustrates the high availability in LinuxDirector.

The server failover is handle as follows: The ”mon” daemon is running on the load balancer to monitor service daemons and server nodes in the cluster. The fping.monitor is configured to detect whether the server nodes is alive every t seconds, and the relative service monitor is also configured to detect the service daemons on all the nodes every m minutes. For example, http.monitor can be used to check the http services; ftp.monitor is for the ftp services, and so on. An alert was written to remove/add a rule in the virtual server table while detecting the server node or daemon is down/up. Therefore, the load balancer can automatically mask service daemons or servers failure and put them into service when they are back.

Now, the load balancer becomes a single failure point of the whole system. In order to prevent the failure of the load balancer, we need setup a backup of the load balancer. Two heartbeat daemons run on the primary and the backup, they heartbeat the message like ”I’m alive” each other through the serial line periodically. When the heartcode daemon of the backup cannot hear the ”I’m alive” message from the primary in the defined time, it will activate the fake to take over the virtual IP address to provide the load-balancing service; when it receives the ”I’m alive” message from the primary later, it will deactivate the fake to release the virtual IP address, and the primary will take over the virtual IP address. However, the failover or the takeover of the primary will cause the established connection in the hash table lost in the current implementation, which will require the clients to send their requests again.

Coda [1] is a fault-tolerant distributed file systems, a descendant of Andrew file system. The contents of servers can be stored in Coda, so that files can be highly available and easy to manage.

6. Connection Affinity

Up to now, we have assumed that each network connection is independent of every other connection, so that each connection can be assigned to a server independently of any past, present or future assignments. However, there are times that two connections from the same client must be assigned to the same server either for functional or for performance reasons.

FTP is an example for a functional requirement for connection affinity. The client establishs two connections to the server, one is a control connection (port 21) to exchange command information, the other is a data connection (usually port 20) to transfer bulk data. For active FTP, the client informs the server the port that it listens to, the data connection is initiated by the server from the server’s port 20 to the client’s port. LinuxDirector could examine the packet coming from clients for the port that client listens to, and create any entry in the hash table for the coming data connection. But for passive FTP, the server tells the clients the port that it listens to, the client initiates the data connection to that port of the server. For the VS/TUN and the VS/DR, LinuxDirector is only on the client-to-server half connection, so it is imposssible for LinuxDirector to get the port from the packet that goes to the client directly.

SSL (Secure Socket Layer) is an example of a protocol that has connection affinity between a client and a server for performance reasons. When a SSL connection is made, port 443 for secure Web servers and port 465 for secure mail server, a key for the connection must be chosen and exchanged. Since it is time-consuming to negociate and generate the SSL key, the successive connections from the same client can also be granted by the server in the life span of the SSL key.

Our current solution to client affinity is to add persistent port handling. When a client first accesses the service marked persistent, the load balancer will create a connection template between the given client and the selected server, then create an entry for the connection in the hash table. The template expires in a configurable time, and the template won't expire if it has its controlled connections. Before the template expires, the connections for any port from the client will send to the right server according to the template. Although the persistent port may cause slight load imbalance among servers because its scheduling granularity is per host, it is a good solution to connection affinity.

7. LinuxDirector Applications

We started the Linux Virtual Server project in May 1998, the first version of LinuxDirector code was released at that time. The project has received a lot of public attention, and LinuxDirector has already been used to build a lot of real-life Internet sites. Since we don't have tens of servers and high-speed network to benchmark the ultra performance of LinuxDirector, we present a sampling of big sites and companies which currently use the LinuxDirector, in order to show the high performance and stability of LinuxDirector. Some big sites and products which we know are based on LinuxDirector includes:

- UK National JANET Web Cache Server, wwwcache.ja.net, provides web caching service for over 150 universities in the UK. They has used 28-node LinuxDirector cluster to replace their original over 50 independent cache servers, the speed now is like that of summer time (most people are on vacation).

- Linux portal, linux.com, has been using many VA Linux SMP machines to provide this highly loaded web service using LinuxDirector for a year.

- SourceForge, sourceforge.net, provides web, ftp, mailing list, cvs hosting services for open source projects all over the world. They use LinuxDirector to balance traffic over ten their servers.

- One of the largest computer manufacturing companies in the world deploys two LinuxDirector clusters for American and European operations of direct sales.

- NetWalk, www.netwalk.com, is using LinuxDirector for 1024 virtual services in a redundant fail over setting with many real servers, which includes the US mirror of our project www.us.linuxvirtualserver.org.

- Red Hat has included the LinuxDirector into Red Hat Linux Distribtion since version 6.1, is actively developing a GUI cluster management tool called piranha to control the LinuxDirector cluster, and provides commercial international support,

- TurboLinux's "world first software Linux clustering product" TurboCluster is actually based on LinuxDirector code and ideas, although TurboLinux never acknowledge LinuxDirector in their press release and demostrations.

8. Conclusion and Future Work

LinuxDirector extends the TCP/IP stack of Linux kernel to support three IP load balancing techniques, VS/NAT, VS/TUN and VS/DR. Four scheduling algorithms have been developed to meet different application situations. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately. The solutions require no modification to either the clients or the servers, and they support most of TCP and UDP services. LinuxDirector is designed for handling millions of concurrent connections.

Compared to other commercial products, LinuxDirector provides many unique features:

- forwarding packets to real servers can either be done using network address translation, fully transparent to the real servers, or using tunneling or direct routing, which provides very high performance. Its IP load balancing technologies is superset of all network load balancing products in the world.

- supporting multiple scheduling algorithms for dispatching connections to the real servers, and further schedulers can be flexibly added as loadable modules.

- a robust and stable code base, a large user and developer base and thus the maturity provided by worldwide peer review.

- proven reliability in the field and real world applications.

- free to everyone.

In the future, we would study and add more load-balancing algorithms to meet more different requirements, such as the load-informed scheduling, content-based scheduling, and geographic-based scheduling for VS/TUN. We would like to explore higher degrees of fault-tolerance; transaction and logging process [10] would be tried to add in the load balancer so that the load balancer can restart the request on another server and the client don't need to send the request again, and the primary and backup load balancers exchange their states so that the existing connection won't be lost when the backup takes over. We would also like to explore how to implement virtual server in IPv6.

Acknowledgements

We would like to thank Julian Anastasov for his bug fixes and smart comments to the LVS code, and Dr. Joseph Mack for writing the LVS-HOWTO document for the Linux Virtual Server project. Thanks must go to many other contributors to the Linux Virtual Server project too.

References

[1] The coda project. CMU Coda Team, 1987-now. http://www.coda.cs.cmu.edu/.

[2] Cisco local director. Cisco Systems, Inc., 1998. http://www.cisco.com/warp/public/751/lodir/index.html.

[3] E. Anderson, D. Patterson, and E. Brewer. The magicrouter: an application of fast packet interposing. http://www.cs.berkeley.edu/ eanders/magicrouter/, May 1996.

[4] D. Andresen, T. Yang, and O. H. Ibarra. Towards a scalable distributed www server on workstation clusters. In *Proc. of 10th IEEE Intl. Symp. Of Parallel Processing (IPPS'96)*, pages 850–856, Arpil 1996. http://www.cs.ucsb.edu/Research/rapid_sweb/SWEB.html.

[5] T. Brisco. Dns support for load balancing. http://www.ietf.org/rfc/rfc1794.txt, April 1995. RFC 1794.

[6] A. Dahlin, M. Froberg, J. Walerud, and P. Winroth. Eddie: A robust and scalable internet server. http://www.eddieware.org/, 1998 - now.

[7] O. P. Damani, P. E. Chung, and Y. Huang. One-ip: Techniques for hosting a service on a cluster of machines. http://www.cs.utexas.edu/users/damani/, August 1997.

[8] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available server. In *COMPCON 1996*, pages 85–92, 1996.

[9] G. Goldszmidt and G. Hunt. Netdispatcher: A tcp connection router. http://www.ics.raleigh.ibm.com/netdispatch/, May 1997.

[10] J. Gray and T. Reuter. *Transaction Processing Concepts and Techniques*. Morgan Kaufmann, 1994.

[11] S. Horman. Creating redundant linux servers. In *The 4th Annual LinuxExpo Conference*, May 1998. http://vergenet.net/linux/fake/.

[12] E. D. Katz, E. D. Katz, and R. McGrath. A scalable http server: The ncsa prototype. *Computer Networks and ISDN Systems*, pages 155–163, May 1994.

[13] T. T. Kwan, R. E. McGrath, and D. A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, pages 68–74, November 1995.

[14] A. Robertson and et al. High-availability linux project. http://www.linux-ha.org/, 1998-now.

[15] R. S.Engelschall. Load balancing your web site: Practical approaches for distributing http traffic. *Web Techniques Magazine*, 3(5), May 1998. http://www.webtechniques.com/.

[16] J. Trocki. mon: Service monitoring daemon. http://www.kernel.org/software/mon/, 1998-now.

[17] C. Yoshikawa, B. Chun, P. Eastharn, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *USENIX'97 Proceedings*, 1997. http://now.cs.berkeley.edu/.

[18] W. Zhang and et al. Linux virtual server project. http://www.LinuxVirtualServer.org/, 1998-now.