Creating Linux Virtual Servers

Wensong Zhang, Shiyao Jin, Quanyuan Wu

National Laboratory for Parallel & Distributed Processing
Changsha, Hunan 410073, China
wensong@iinchina.net
http://proxy.iinchina.net/˜wensong

ABSTRACT

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 loosely coupled architecture is more scalable, more cost-effective and more reliable than a single processor system or a tightly coupled multiprocessor system.

This paper presents how to create Linux virtual servers. Virtual server is a high-performance and highly available server built on a cluster of real servers. Two methods of IP-level load balancing were developed to make parallel services of the cluster to appear as a virtual service on a single IP address, one is virtual server via Network Address Translation, the other is virtual server via IP tunneling. Currently 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.

1. Introduction

With the explosive growth of the Internet and its increasingly important role in our lives, the traffic on the Internet is increasing dramatically, which has been growing at over 100% annual rate. The load on popular Internet sites is growing rapidly, some have already got tens of millions hits per day. More and more administrators have met the performance bottleneck problem of their servers, and with the increasing access requests the servers will be easily overloaded for a short time. Nowadays, more and more companies are moving their businesses on the Internet, any interrupt/stop of services on the servers means business lose, and high availability of these servers becomes increasingly important. Therefore, the demand for scalable and highly available servers is growing urgently. The requirements for this type of servers are summarized as follows:

* Incremental scalability
* 24x7 availability
* Cost-effectiveness

The single server solution, which is to upgrade the server to a higher performance server, has its shortcomings to meet the requirements. The upgrading process is complex, and the original machine may be wasted. When requests increase, it will be overloaded soon so that we have to upgrade again. The server is a single point of failure. The higher end the server is upgraded to, the 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, scalability and high availability of parallel services in the cluster.

This paper presents how to create Linux virtual servers. Virtual server is a scalable and highly available server built on a cluster of loosely coupled independent servers. The architecture of the cluster is transparent to clients outside the cluster. 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. The architecture of a generic virtual server is illustrated in Figure 1.

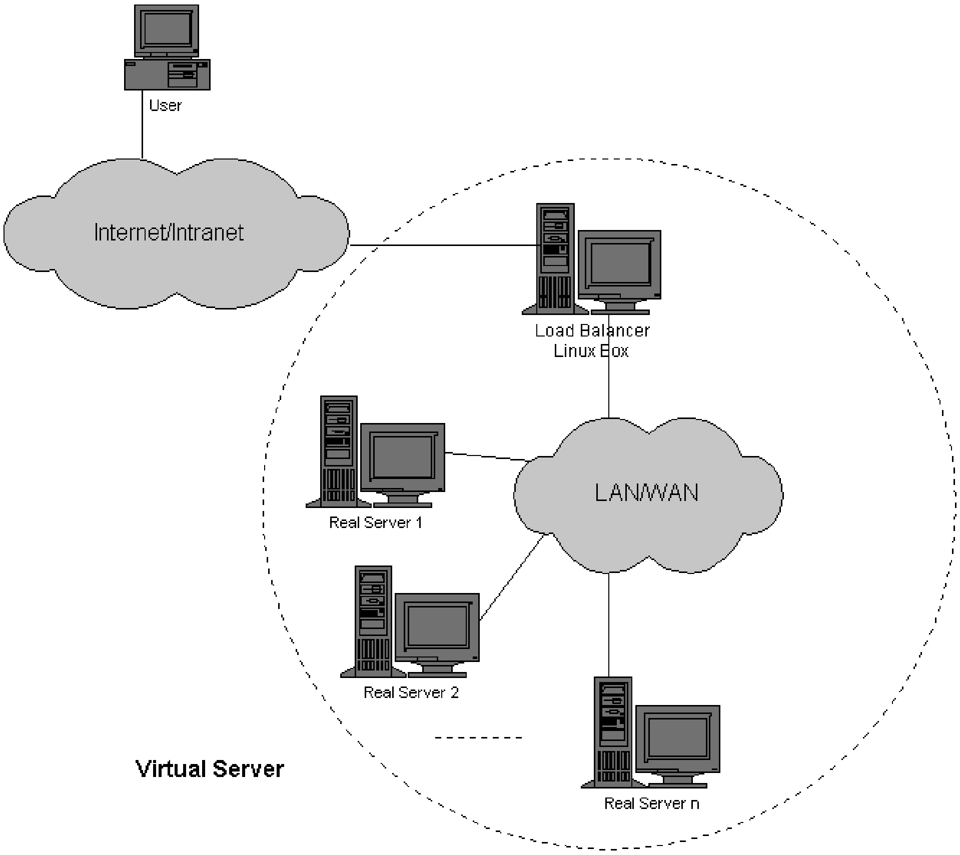

Figure 1. generic architecture of a virtual server

The real servers may be interconnected by high-speed LAN or by geographically dispersed WAN. The front-end of the real servers is a load balancer, which schedules requests to the different servers and make parallel services of the cluster to appear as a virtual service on a single IP address. 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 the architectures and working principles of virtual server via Network Address Translation and virtual server via IP tunneling, and also discuss their advantages and disadvantages. In Section 4, we describe the four scheduling algorithms that have been developed for Linux virtual server. In Section 5, we describe the high availability issues for Linux virtual server. Finally, conclusion and future work appear in Section 6.

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, such as a proxy server for web services. Based on this scenario, we can see that there are many ways to dispatch requests to a cluster of servers in the different levels. In general, these servers have the same service and the same set of contents. The contents are either replicated on each server's local disk, shared on a network filesystem, or served by a distributed file system. Existing request dispatching techniques can be classified into the following categories:

- **The client-side approach**

  Berkeley's Smart Client[1] 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. Boston University developed dynamic server selection techniques[2] using bandwidth probing on the client side, in which the maximum possible bandwidth along a given path and the current congestion along a path are estimated through probing. 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 by using of the Round-Robin DNS approach.[3, 4, 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. In this way, the 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 host-based, different users' access pattern may lead to dynamic load imbalance, because some people may pull lots of pages from the site, and others may just surf a few pages and go away. Moreover, 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" or "refresh" button in their browsers.

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

  EDDIE[6] , Reverse-proxy[7] , pWEB[8] and SWEB[9] use the application-level scheduling approach to build a scalable web server. They all forward the HTTP requests to the 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. And, the overhead of dealing the HTTP requests and replies in the application-level is high too, so 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[10] and Cisco's LocalDirector[11] use the Network Address Translation approach to make parallel service on different server to appear as a virtual service on a single IP address. The load balancer changes the destination address of the request packets and forwards the chosen server, then change the source address of the reply packets back to the original destination address. However, the MagicRouter doesn't survive to be a useful system for other users, the LocalDirector is too expensive, and they only support part of TCP protocol. This approach was well implemented in the Linux virtual server, and it will be discussed in detail in the following sections.

  IBM's TCP router[12] 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.

ONE-IP[13] is another IP-level scheduling approach. It requires that all servers have their own IP addresses in a network and they are all configured with the same IP 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 the reply packets can be avoided. The disadvantage is it can be applied to all operating systems because some operating systems will shutdown the network interface when detecting IP address collision, the local filtering also requires the modification of the kernel code of every server.

3. Virtual Server Architectures

Linux virtual server[14] is now implemented in two ways. One is virtual server via Network Address Translation; the other is virtual server via IP tunneling. The following two subsections explain their different architectures and working principles respectively. The third subsection discusses their advantages and disadvantages. The virtual server code is now developed based on Linux IP Masquerading code in the Linux kernel 2.0, and some of Steven Clarke's port forwarding code is reused. It supports both TCP and UDP services, such as HTTP, Proxy, DNS and so on. For protocols that transmit IP addresses and/or port numbers as application data, additional code is need to handle it. Currently FTP was supported, others require adding new handling code.

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 outside the network (or in the Internet). The need for network address translation arises when hosts in internal networks want to access the Internet or to be accessed in the Internet. Network address translation relies on the fact that the headers for Internet protocols 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 via NAT.

The architecture of virtual server via NAT is illustrated in Figure 2. The load balancer and real servers are interconnected by a switch or a hub. The real servers usually 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 filesystem, or served by a distributed file system (such as AFS or CODA). The load balancer dispatched requests to different real servers via NAT.

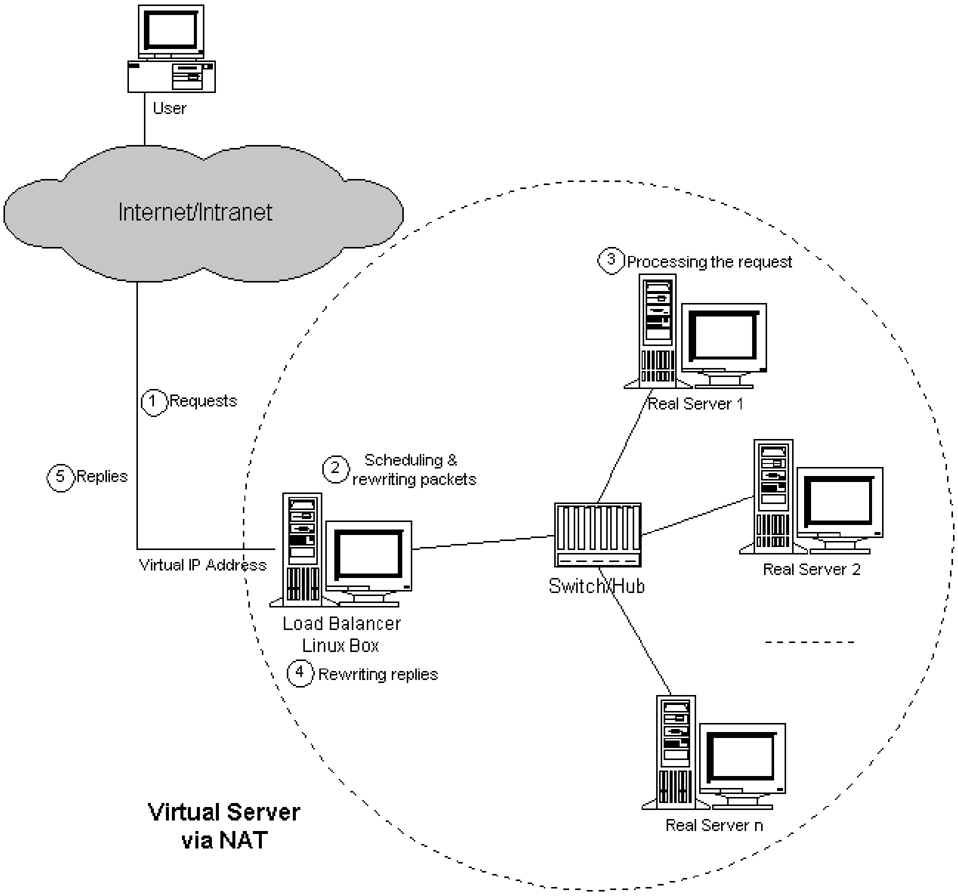

Figure 2: Architecture of a virtual server via NAT

The workflow of virtual server via NAT is as follows: When a user accesses the service provided by the server cluster, the request packet destined for virtual IP address (the external IP address for the load balancer) arrives at the load balancer. The load balancer examines the packet's destination address and port number. If they are matched for a virtual server service according to the virtual server rule table, a real server is chosen from the cluster by a scheduling algorithm, and the connection is added into the hash table which records all established connections. Then, the destination address and the port of the packet are rewritten to those of the chosen server, and the packet is forwarded to the server. When the incoming packet belongs to this connection and the established connection can be found in the hash table, the packet will be rewritten and forwarded to the chosen server. When the reply packets come back, the load balancer rewrites the source address and port of the packets to those of the virtual service. When the connection terminates or timeouts, the connection record will be removed in the hash table.

An example of Virtual Server via NAT is illustrated in Figure 3.

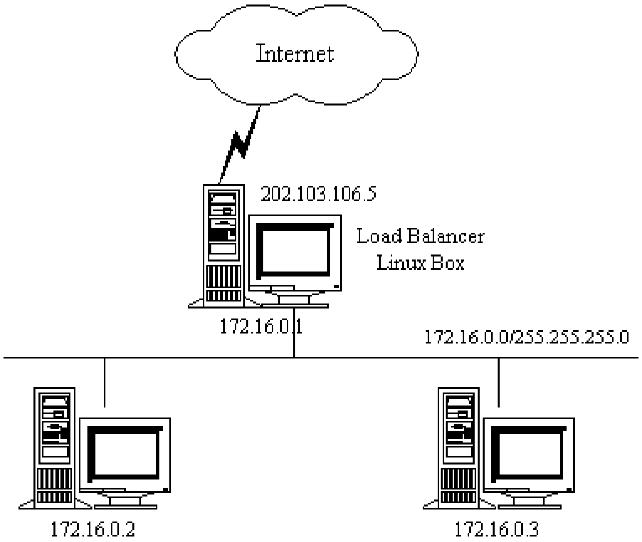

Figure 3: an example of Virtual Server via NAT

Table 1 illustrates the rules specified in the Linux box with virtual server support.

Table 1: an example of virtual server rules

Protocol	Virtual IP Address	Port	Real IP Address	Port	Weight
TCP	202.103.106.5	80	172.16.0.2	80	1
			172.16.0.3	8000	2
TCP	202.103.106.5	21	172.16.0.3	21	1

All traffic destined for IP address 202.103.106.5 Port 80 is load-balanced over real IP address 172.16.0.2 Port 80 and 172.16.0.3 Port 8000. Traffic destined for IP address 202.103.106.5 Port 21 is port-forwarded to real IP address 172.16.0.3 Port 21. Note real servers can run any OS that supports TCP/IP, the default route of real servers must be the virtual server(172.16.0.1 in this example).

Packet rewriting works as follows.

The incoming packet for web service would has source and destination addresses as:

SOURCE	202.100.1.2:3456	DEST	202.103.106.5:80

The load balancer will choose a real server, e.g. 172.16.0.3:8000. The packet would be rewritten and forwarded to the server as:

SOURCE	202.100.1.2:3456	DEST	172.16.0.3:8000

Replies get back to the load balancer as:

SOURCE	172.16.0.3:8000	DEST	202.100.1.2:3456

The packets would be written back to the virtual server address and returned to the client as:

SOURCE	202.103.106.5:80	DEST	202.100.1.2:3456

3.2. Virtual Server via IP Tunneling

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

The architecture of virtual server via IP tunneling is illustrated in Figure 4. The most different thing of virtual server via IP tunneling to that of virtual server via NAT is that the load balancer sends requests to real servers through IP tunnel in the former, and the load balancer sends request to real servers via network address translation in the latter.

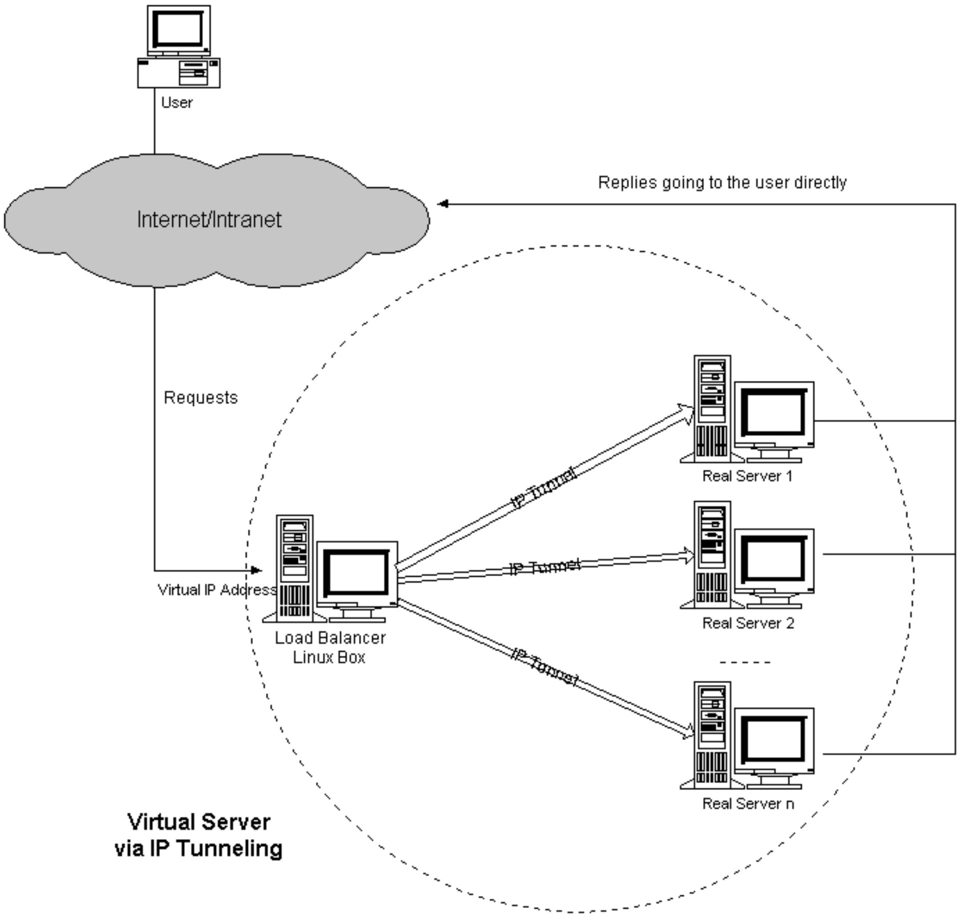

Figure 4: Architecture of a virtual server via IP tunneling

The workflow of virtual server via IP tunneling is as follows: When a user accesses the service provided by the server cluster, the packet destined for virtual IP address (the IP address for the load balancer) arrives. The load balancer examines the packet's destination address and port. If they are matched for a virtual server service, a real server is chosen from the cluster by a scheduling algorithm, and the connection is added into the hash table which record all established connections. Then, the load balancer encapsulates the packet within an IP datagram and forwards it to the chosen server. When the incoming packet belongs to this connection and the chosen server can be found in the hash table, the packet will be again encapsulated and forwarded to the server. When the server receives the encapsulated packet, it decapsulates the packet and processes the request, finally returns the result to the user directly. When the connection terminates or timeouts, the connection record will be removed from the hash table.

Note that real servers can have any real IP address in any network, they can be geographically distributed, but they must support IP encapsulation protocol and they all have one of their tunnel devices configured with <Virtual IP Address>, like "ifconfig tunl0 <Virtual IP Address>" in Linux. When the encapsulated packet arrives, the real server decapsulates it and finds that the packet is destined for <Virtual IP Address>, so it processes the request and returns the result directly to the user in the end.

3.3. Advantages and Disadvantages

3.3.1. Virtual server via NAT

The advantage of the virtual server via NAT is that real servers can run any operating system that supports TCP/IP protocol, real servers can use private Internet addresses, and only an IP address is needed for the load balancer.

The disadvantage is that the scalability of the virtual server via NAT is not very good. The load balancer may be a bottleneck of the whole system when the number of server nodes increase to around 25 or more which depends on the throughout of real servers, because both the request packets and the reply packets are need to be rewritten by the load balancer. Supposing the average length of TCP packets is 536 Bytes, 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 22 real servers if the average throughout of real servers is 400KBytes/s.

Virtual server via NAT can meet the performance request of many servers. When the load balancer is becoming a bottleneck of the whole system, there are two methods to solve it, one is the virtual server via IP tunneling, and the other is the hybrid approach. In the hybrid approach, there are many load balancers which all have their own server clusters, and the load balancers are grouped at a single domain name by Round-Round DNS.

3.3.2. Virtual server via IP tunneling

We can see from many Internet services (such as web service) that the incoming packets are always short and reply packets always carry large amount of data. In the virtual server via IP tunneling, the load balancer just schedules requests to the different real servers, and the real servers return replies directly to the users. Thus the load balancer can handle huge amount of requests; it may schedule over 100 real servers and won't be the bottleneck of the system. Therefore, using IP tunneling will greatly increase the maximum number of server nodes for a load balancer. The maximum throughout of the virtual server may reach over 1Gbps, even if the load balancer just has 100Mbps full-duplex network adapter.

The IP tunneling feature can be used to build a very high-performance virtual server, extremely good to build a virtual proxy cache server, because when the proxy servers receive request, it can access the Internet directly to fetch objects and return them directly to the users.

However, virtual server via IP tunneling requires real 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, virtual server via IP tunneling should be applicable to servers running other operating systems.

4. Scheduling Algorithms

We have implemented four scheduling algorithms for selecting a server from the cluster: 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, in its word meaning, 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 host-based, 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 network connection-based, and it is more superior to the round-robin DNS due to the 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 the processing capacity. The default weight is 1. For example, the real servers, A, B and C, have the weights, 4, 3, 2 respectively, a good scheduling sequence will be ABCABCABA in a scheduling period (mod sum(Wi)). In the implementation of the weighted round-robin scheduling, a scheduling sequence will be generated according to the server weights after the rules of virtual server are modified. The network connections are directed to the different real servers based on the scheduling sequence in a round-robin manner.

The weighted round-robin scheduling doesn't need to count the network connections for each real server, and the overhead of scheduling is smaller than dynamic scheduling algorithms, it can have more real servers. However, it may lead to dynamic load imbalance among the real servers if the load of requests vary highly. In short, it is still possible that most of long requests may be directed to a real server.

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 won't have chance to be directed to a server.

At a first look, the least-connection scheduling can also perform well even when there are servers 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, between this 2 minutes 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 processing thousands of requests and kept them in the TCP's TIME_WAIT state, but the server B is lagging to get its thousands of connections finished. So, 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 you can assign a performance weight to each real server. The servers with a higher weight value will receive a larger percentage of active connections at any one 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 default weight is one.

The weighted least-connections scheduling works as follows:

> Supposing there is n real servers, each server i has weight Wi (i=1,..,n) and active connections Ci (i=1,..,n), ALL_CONNECTIONS is the sum of Ci (i=1,..,n), the network connection will be directed to the server j, in which
>
> $$(C_j/ALL\_CONNECTIONS)/W_j = \min\{(C_i/ALL\_CONNECTIONS)/W_i\}(i = 1, .., n)$$

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

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

The weighted least-connection scheduling algorithm requires additional division than the least-connection. In a hope to minimize the overhead of scheduling when servers have the same processing capacity, both the least-connection scheduling and the weighted least-connection scheduling algorithms are implemented.

5. High Availability

As more and more critical commercial applications move on the Internet, providing highly available servers 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.

The high availability of virtual server is now provided by using of "mon"[15] and "fake"[16] software. The "mon" is a general-purpose resource monitoring system, which can be used to monitor network service availability and server nodes. Fake is IP take-over software by using of ARP spoofing.

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, setup a backup server of the load balancer, the "fake" software is used for the backup to takeover the IP addresses of the load balancer when the load balancer fails. Again, the "mon" is used to detect the status of the load balancer to activate/deactivate the "fake" on the backup server. The "mon" daemon also runs on the backup so that the backup has current status of the cluster. When the primary fails, the backup will take over its IP address(es) to continue to provide services, however the established connection in the hash table will be lost in the current implementation, which will require the clients to send their requests again.

6. Conclusion and Future Work

We have demonstrated how to create Linux virtual servers based on a cluster of real servers. Two methods of IP-level load balancing were developed to make parallel services of the cluster to appear as a virtual service on a single IP address, one is virtual server via Network Address Translation, the other is virtual server via IP tunneling. Currently 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.

In the future, we would study and add more load-balancing algorithms to meet more different requirements, such as the load-informed scheduling, and geographic-based scheduling for virtual server via IP tunneling. We would like to integrate the "heartbeat" code and the CODA distributed fault-tolerant filesystem into virtual server, develop a cluster manager, and make it easy to setup and administrator Linux virtual servers. We would explore higher degrees of fault-tolerance; transaction and logging process[17] 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.

References

1. Chad Yoshikawa, Brent Chun, Paul Eastharn, Armin Vahdat, Thomas Anderson, and David Culler, "Using Smart Clients to Build Scalable Services," *USENIX'97* (1997). http://now.cs.berkeley.edu/.
2. Robert L. Carter and Mark E. Crovella, "Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks," Boston University Technical Report (1996). http://www.ncstrl.org/.
3. Eric Dean Katz, Michelle Butler, and Robert McGrath, "A Scalable HTTP Server: The NCSA Prototype," *Computer Networks and ISDN Systems,* pp. 155-163 (May 1994).
4. Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed, "NCSA's World Wide Web Server: Design and Performance," *IEEE Computer,* pp. 68-74 (November 1995).
5. T. Brisco, *DNS Support for Load Balancing,* RFC 1794. http://www.internic.net/ds/.
6. A. Dahlin, M. Froberg, J. Walerud, and P. Winroth, *EDDIE: A Robust and Scalable Internet Server* (May 1998). http://www.eddieware.org/.
7. Ralf S.Engelschall, "Load Balancing Your Web Site: Practical Approaches for Distributing HTTP Traffic," *Web Techniques Magazine,* 3, 5 (May 1998). http://www.webtechniques.com/.
8. Edward Walker, *pWEB - A Parallel Web Server Harness* (April, 1997). http://www.ihpc.nus.edu.sg/STAFF/edward/pweb.html.
9. Daniel Andresen, Tao Yang, and Oscar H. Ibarra, "Towards a Scalable Distributed WWW Server on Workstation Clusters," *Proc. of 10th IEEE Intl. Symp. Of Parallel Processing (IPPS'96),* pp. 850-856 (Arpil 1996). http://www.cs.ucsb.edu/Research/rapid_sweb/SWEB.html.
10. Eric Anderson, Dave Patterson, and Eric Brewer, *The Magicrouter: an Application of Fast Packet Interposing* (May 1996). http://www.cs.berkeley.edu/˜eanders/magicrouter/.
11. Cisco System, *Cisco Local Director* (1998). http://www.cisco.com/warp/public/751/lodir/index.html.
12. D. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A Scalable and Highly Available Server," *COMPCON 1996,* pp. 85-92 (1996).
13. Om P. Damani, P. Emerald Chung, and Yennun Huang, *ONE-IP: Techniques for Hosting a Service on a Cluster of Machines* (August 1997). http://www.cs.utexas.edu/users/damani/.
14. Wensong Zhang, *Linux Virtual Server Project* (May 1998). http://proxy.iinchina.net/˜wensong/ippfvs/.
15. Jim Trocki, *mon: Service Monitoring Daemon* (July 1998). http://consult.ml.org/˜trockij/mon/.
16. Simon Horman, "Creating Redundant Linux Servers," *The 4th Annual LinuxExpo Conference* (May 1998). http://linux.zipworld.com.au/fake/.
17. Jim Gray and T. Reuter, *Transaction Processing Concepts and Techniques,* Morgan Kaufmann (1994).