An Efficient Web Cluster with Content-Aware Request Distribution on Linux Kernel 2.6

Shang-Yi Zhuang, Cheng-Wei Lin, Mei-Ling Chiang
Department of Information Management, National Chi-Nan University, Puli, NanTou 545, Taiwan, Republic of China
snowhigh1211@gmail.com, {joanna, s96213514} @ncnu.edu.tw

Abstract

The web cluster has been commonly used in popular web sites because of its high availability and scalability. Many researches on web clusters have focused on content-aware request distribution because dispatching requests from clients to servers according to the content of request (i.e. URI) can obtain better system performance and achieve high locality in back-end servers' main memory caches. In our previous work, we have proposed the TCP Rebuilding mechanism and the fast handshaking mechanism to build a content-aware web cluster named LVS-CAD on Linux kernel 2.4.

In this paper, to gain performance improvement from new features in kernel 2.6, we have implemented the LVS-CAD on kernel 2.6.18. Besides, in order to make LVS-CAD platform easier to maintain and debug, we have also implemented the related mechanisms as Linux loadable kernel modules that can be dynamically loaded/unloaded into/from Linux kernel. Moreover, we have proposed and implemented a mechanism to reduce excess multiple handoffs under persistent connection to further improve performance of a web cluster. This enhanced LVS-CAD is named LVS-eCAD. Experimental results show that LVS-CAD on Linux kernel 2.6 can perform 158.22% better than LVS-CAD on Linux kernel 2.4 and LVS-eCAD can outperform LVS-CAD and LVS by 25% on Linux kernel 2.6.

Keywords: Web Cluster, Content-aware Request Distribution, Multiple Handoffs

1. Introduction

Because of the popularization of internet, many popular web sites need to deal with large amount of requests in a short period of time. Web clusters consisting of multiple web servers connected by a high-speed LAN have been widely adopted due to the advantages of load sharing or balancing, scalability, and high availability.

The general web cluster employs a layer-4 web switch [4] in the point of view of network layers to distribute requests among back-end servers with request distribution policies. A web cluster with a layer-4 web switch is also called a content-blind platform, in which the web switch is not aware of HTTP contents of requests. So the layer-4 web switch is not sufficient while the cluster needs to provide different quality of services or dispatch requests based on request contents (i.e. URI). In contrast, a web cluster with a layer-7 web switch [4] can distribute requests according to the request contents and is called content-aware web cluster [4].

The architecture of the content-aware web cluster can be classified into one-way and two-way. In the two-way architecture, back-end servers receive requests from front-end server and all response packets have to pass through the front-end server. Whereas, in the one-way architecture, back-end servers receive requests from front-end server and respond to clients directly. Thus, web cluster can perform better in one-way architecture.

Some researches [4,8] have proposed techniques to construct a layer-7 web switch with one-way or two-way architecture. In our previous work [8], we have implemented a content-aware dispatching web cluster called LVS-CAD on Linux with kernel 2.4.18. This cluster is efficient since it belongs to one-way architecture and uses the light-weight TCP Rebuilding mechanism [8] in back-end servers to rebuild the connection with client and the fast handshaking mechanism in front-end server to establish connection with client.

In recent years, many new features in kernel 2.6 make a web server more stable and efficient [6], including Native POSIX Thread Library (NPTL) [5], O(1) scheduler [1], and some I/O improvements [6]. To gain performance improvement from new features in kernel 2.6 [6], in this research, we have transplanted the LVS-CAD on Linux kernel 2.6.18. Different from our previous work, we have implemented these two mechanisms as loadable kernel modules that can be dynamically loaded/unloaded into/from Linux kernel, to make the LVS-CAD platform more flexible and easier to maintain and debug. Experimental results show that LVS-CAD in Linux kernel 2.6.18 can outperform the original LVS-CAD in Linux kernel 2.4.18 by 158.22%. Moreover, we have designed and implemented a mechanism to reduce multiple handoffs under persistent connection to further improve performance of a web cluster. Experimental results show that this enhanced LVS-CAD named LVS-eCAD can outperform LVS-CAD and LVS by 24.62% and 24.55% respectively under Linux kernel 2.6.

2. LVS and LVS-CAD Web Clusters

Linux Virtual Server (LVS) [7] is a set of independent Linux-based servers, which acts as a single server to serve requests from clients. In LVS architecture, a web cluster comprises a request-dispatching front-end server and several request-handling back-end servers. The front-end server is a layer-4 web switch which can perform only content-blind request distribution that does not consider request content (i.e. URI) in dispatching requests from clients to servers.

In the content-aware LVS-CAD web cluster [8], the TCP Rebuilding mechanism is applied on each back-end server and the fast handshaking mechanism is implemented on the front-end server. The front-end server can use not only content-aware dispatching policies, but also various content-blind dispatching policies of LVS. The IPVS-CAD module is modified from LVS's IPVS module [7] to apply fast handshaking mechanism. The fast handshaking mechanism can perform three-way handshaking with clients at IP layer instead of TCP layer so that it is more efficient than the ordinary three-way handshaking.

Figure 1 shows the packet flow of LVS-CAD. In the step 1, the IPVS-CAD module in the front-end server performs three way handshaking with the client. The front-end server then will receive the packet with HTTP payload so that the front-end server can perform its content-aware request dispatching policies. In the step 2, the packets will be forwarded to the selected back-end server without any modification. In the step 3, the back-end server performs TCP Rebuilding to rebuilding the connection when it receives the packet and then responds to the client directly.

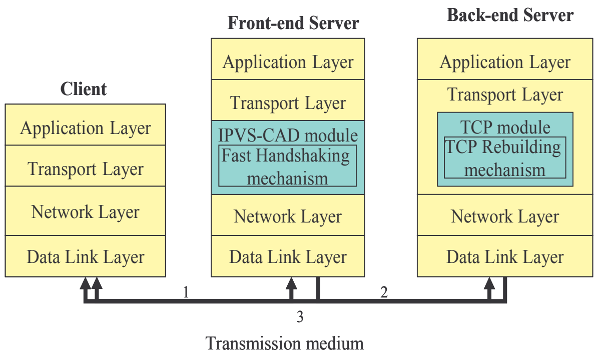


Figure 1: Packet flow of LVS-CAD

LVS-CAD is efficient not only because it belongs to one-way architecture, but also the front-end server does not have to modify those packets forwarded to the target back-end server. Especially, the front-end server does not have to send extra packets to the target back-end server for TCP state transfer so that the internal network will not be full of extra packets for TCP connection transfer. Besides, the back-end server's response packets do not have to be modified, so the back-end server can be more efficient to send response packets to clients.

3. Implementation of LVS-CAD on Linux Kernel 2.6

3.1 Front-end Server Implementation

This section introduces our major works of the front-end server implementation including the fast handshaking module implementation in IPVS module on Linux kernel 2.6.18, major changes of IPVS module from Linux kernel 2.4 to 2.6, and the connection state machine issue in IPVS-CAD module.

In LVS-CAD of our previous work, to enable the front-end server to adopt content-aware request dispatching policies, at first the front-end server has to perform three-way handshaking to establish connection with client when it receives a SYN request from client. After the connection has been established, the front-end server then can receive the following request packet with PSH flag and parse the content of the request to perform content-aware dispatching. Therefore, the fast TCP module handshaking [8] is proposed and implemented to perform three-way handshaking with clients at network layer instead of transport layer in IPVS-CAD module. The fast handshaking mechanism is implemented in the modified IPVS module named IPVS-CAD module [8].

In this paper, we have implemented this fast handshaking mechanism from Linux kernel 2.4.18 to Linux kernel 2.6.18 and implemented it as a loadable kernel module. For easier to debug and maintain, we implement the fast handshaking mechanism as a single module instead of in the IPVS module, but the IPVS module still has to be modified due to the export function problems described in Section 3.2.3 and the connection state machine problem described in Section 3.1.2. The modified IPVS module is still named IPVS-CAD module.

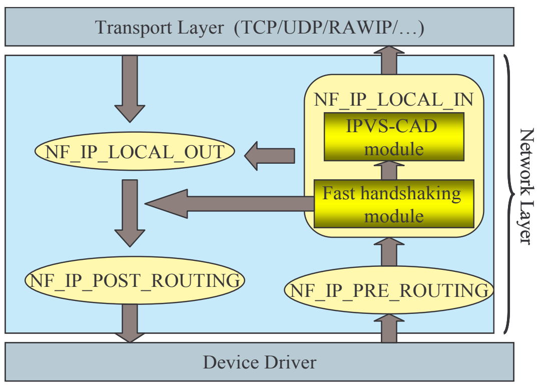


Figure 2. Simplified Netfilter packet flow with Netfilter hooks in the implementation of the fast handshaking module

Figure 2 depicts the packet flow with Netfilter hooks (except the NF_IP_FORWARD hook) in the implementation of the fast handshaking module of the front-end server. In LVS-CAD, since fast handshaking module has to deal with three-way handshaking with the client before the incoming SYN packet is forwarded by IPVS-CAD module, the fast handshaking module is hooked in front of the IPVS-CAD module.

3.1.1 Content-Aware Routing

In the original IPVS module of LVS, the dispatching policy is invoked when the SYN packet is arrived for connection establishment and the connection information is stored in a connection table including the information of the selected back-end server. Therefore, the following packets in a persistent connection will be forwarded to the target server without calling the dispatching policy. Under a persistent connection, it is efficient to call scheduler once and for all. However, if the front-end server wants to perform content-aware routing, it has to call dispatching policy when each request arrives. Thus, the fast handshaking module calls *conn_schedule* when each request arrives and changes the destination server information is stored in the connection table if necessary.

Figure 3 shows the flow of fast handshaking module. First, the incoming packet (step 1) has to be examined to check if it is served by this cluster. If not, the packet is past to the upper layer (step 2.a). Otherwise (step 2.b), if the packet is a SYN packet for connection establishment (step 3.b), fast handshaking handling will send SYN-ACK packet (step 4) to establish a connection with the client. If the packet belongs to an established connection, the designated dispatching policy (step 3.a) will be used to select a back-end server to serve the request and send the request to IPVS-CAD module (step 5.b). If front-end server has to perform multi-handoff, the fast handshaking module will send a RST packet (step 5.a) to the current connected back-end server to terminate the connection.

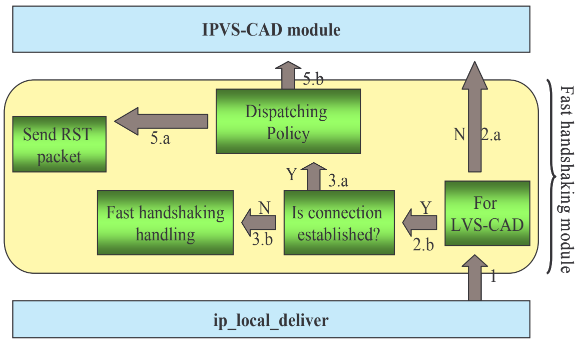


Figure 3. Packets flows in the fast handshaking module

The client may send different requests to the web cluster for receiving different web files in a persistent connection. When a client sends two requests for receiving two web files which are served by different back-end servers in a persistent connection, the front-end server has to send the first request to one back-end server to serve the request. Then, the front-end server has to handoff the client connection to another back-end server to serve the second request. Therefore, a persistent connection between a client and a back-end server may be hand off several times by the front-end server if needed. Therefore, to support persistent connection, the front-end server has to perform multi-handoff mechanism.

Figure 4 is an example of connection handoff between back-end servers that is transparent to clients. The front-end server simply sends reset packet (i.e. RST packet) to the current connected back-end server 2 to terminate the connection and then sends the request with PSH flag to the newly selected back-end server 1 to serve the request.

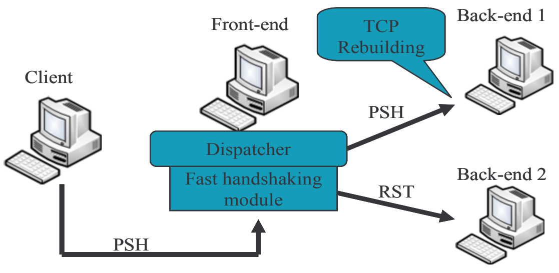


Figure 4. An example of multi-handoff

3.1.2 Connection State Machine in IPVS module

The front-end server uses its designated request dispatching policy to select a back-end server for serving the request. Some request dispatching policies (e.g. Weighted Least Connection policy) need information (e.g. the number of connections) on back-end servers to select a proper back-end server to serve a request. Thus, the connection state should be maintained in front-end server. As shown in Figure 5, IPVS module in front-end server will maintain the connection state for each connection and each connection's state will be changed depending on the incoming packet. In IPVS module, before each packet is forwarded to the selected back-end server, the connection state will be maintained by calling the *state_transition* function and the parameter is the received packet.

Since in LVS-CAD for the port 80, the three-way handshaking has been handled by our fast handshaking module, the IPVS-CAD module will not receive the SYN and ACK packets. Therefore, the connection state of each connection will never be the ESTABLISHED state and the information of established connection (i.e. active connection) numbers in back-end servers will not be correct. To make the connection state of every connection work properly, when each fast three-way handshaking has been performed and the IPVS-CAD has received a packet with PSH flag, we first set the SYN flag on in the received packet and call the *state_transition* function twice. In the first call, since the received packet's SYN flag is set, the connection state will be the SYN_RECV state. In the second call, since the ACK flag is also set in the received packet, the connection state will be ESTABLISHED. After the connection state is ESTABLISHED, we set the packet's SYN flag off and continue the processing in IPVS-CAD.

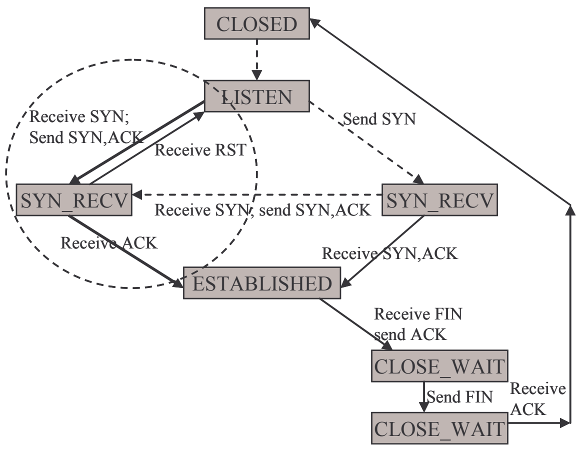


Figure 5. Simplified IPVS state machine in IPVS module

However, the connection state is recorded in a connection entry created in IPVS-CAD module and the *state_transition* function must been referenced in kernel instead of modules, so we can not change the connection state in fast handshaking module. To solve this problem, we simply use a patch file to patch IPVS-CAD module to make connection state of each connection work properly by using above method so that the IPVS-CAD module can maintain the correct information of established connections in back-end servers.

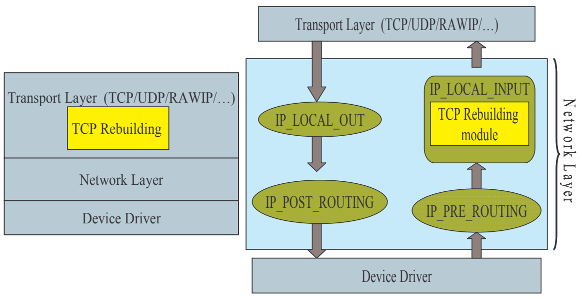
3.2 Back-end Server Implementation

We mainly implement TCP Rebuilding mechanism in the back-end servers of LVS-CAD. This section presents our implementation and some problems we solve.

3.2.1 Different Layers Problem

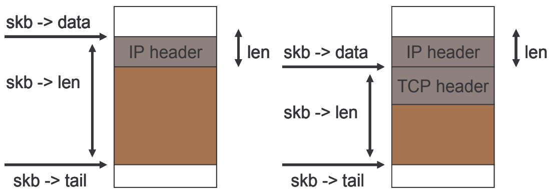
TCP Rebuilding mechanism is originally implemented at transport layer in kernel 2.4, as shown in Figure 6. In this paper, we have implemented it as the kernel module hooked at NF_IP_LOCAL_IN in kernel 2.6. Like fast handshaking module, the TCP Rebuilding module is also hooked by Netfilter at the network layer.

To implement TCP Rebuilding mechanism in network layer as a kernel module, we have to solve some problems. In Linux kernel, each packet is represented by a socket buffer. The variables in a socket buffer will be changed when the socket buffer is past to different layers. As shown in Figure 7, the *data* pointer of the socket buffer (i.e. skb->data) points to IP header in the network layer and points to TCP header in the transport layer. In our implementation of the TCP Rebuilding module, we reuse some functions which are originally used by TCP module to rebuild the connection. However, transport layer functions used in network layer may cause unexpected errors because the *data* pointer of the socket buffer refers to different headers in different layers. Hence, we have to modify the socket buffer structure so that we can reuse TCP functions to perform TCP Rebuilding mechanism at the network layer. To do so, we use *skb_pull* function which is used to move the *data* pointer of a socket buffer from IP header to TCP header before performing TCP Rebuilding mechanism and use *skb_push* function which is used to move the *data* pointer of a socket buffer from TCP header to IP header to modify the packet after finishing the operations of TCP Rebuilding mechanism. Therefore, we can reuse some functions which are originally used by TCP module to rebuild the connection in the network layer.



(a) Kernel 2.4 **(b) Kernel 2.6**

Figure 6. Implementation of TCP Rebuilding in different kernels and different layers



(a) Network layer **(b) Transport layer**

Figure 7. The socket buffer at different layers

3.2.2 Implementation of TCP Rebuilding Module

Figure 8 shows the packet flow in TCP Rebuilding module. In our previous work of implementation in kernel 2.4, the fake SYN packet and fake ACK packet are made at transport layer for spoofing the TCP module to rebuild the connection by conjecturing the sequence number from the received PSH packet. In the implementation of the TCP Rebuilding module in kernel 2.6, when a packet arrives for port 80, the module first checks the TCP state to see if the TCP Rebuilding needs to be performed (step 1). If the state is LISTEN and the PSH flag is set in the packet, TCP Rebuilding is performed (step 2). Then, a fake SYN packet will be produced by copying the received packet (step 3). Some TCP functions will then be called for establishing the connection. The fake ACK packet will then be produced to accomplish the connection (step 4). Finally, the received PSH packet will be sent to the upper transport layer (step 5).

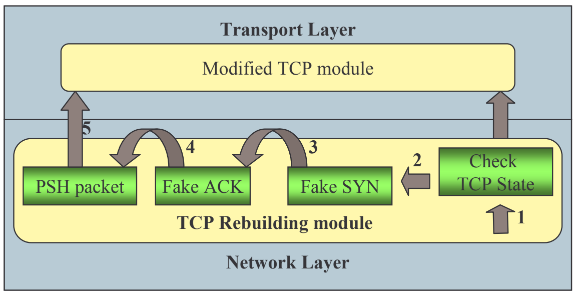


Figure 8. The packet flow with TCP Rebuilding module

3.2.3 Other Kernel Modifications

In kernel 2.6, a kernel module can not call all functions in the kernel and only the exported functions can be used by a kernel module. Therefore, we have to export some essential kernel functions for them to be used by TCP Rebuilding module by using the EXPORT_SYMBOL macro. To export these functions, we implement a patch file for users to patch the kernel to include the EXPORT_SYMBOL macros at first so that TCP Rebuilding module can use these functions to rebuild connections.

Besides, compared to kernel 2.4, some function names in kernel 2.6 have been changed. Therefore, when we transplant the TCP Rebuilding codes from kernel 2.4 to kernel 2.6, we must modify the TCP Rebuilding codes to invoke correct functions for the changes of function names. For example, the *inet_protocol* structure which is used to register network protocol in kernel 2.4 has been changed to *net_protocol* in kernel 2.6 and some related variables have also been changed. In addition, Linux 2.6 kernel supports more network protocols like DCCP and SCTP. These new supported protocols may overlap some source codes with the existent TCP or UDP protocols. To avoid the duplicated source codes, TCP module also has some code changes in kernel 2.6. For example, the *__tcp_v4_lookup* function used to access *sock* structure is changed to *__inet_lookup* function. So, the DCCP protocol can also uses the *__inet_lookup* function to access *sock* structure. For the same reason, the *tcp_v4_synq_add* function which is used to record received SYN packet is changed to *inet_csk_reqsk_queue_hash_add*.

4. Performance Evaluation

Table 1 shows the hardware and software environment. Our web cluster includes eight back-end servers and one front-end server. Besides, ten computers are used as clients and each client runs httperf [9] benchmark to generate requests and send requests to the web cluster. All computers are connected to a ZyXEL Dimension GS-1124 switch. We use the publicly obtainable trace named WorldCup98 trace [2] from Internet Traffic Archive [10]. The average file size is 161 Kbytes. The maximal file size is 2,824 Kbytes and the total file size is 1,703,428 Kbytes.

Table 1. Hardware/Software Environment

	Front-end	Back-end	Client
CPU (Hz)	P4 3.4G		P4 2.4G P3 800M
RAM	1G	256/128MB	256MB
NIC(Mbps)	Intel Pro 100/1000		Reltek RTL8139 Intel Pro100/1000 D-LinkDGE-530T
IPVS	1.0.4/1.21	X	X
Red Hat Linux 8.0 / 2.4.18, Gentoo Linux / 2.6.18			

4.1 Platform Performance

We present the scalability comparison between the original LVS-CAD in Linux kernel 2.4.18 and the transplanted LVS-CAD in Linux kernel 2.6.18. Both LVS-CADs use Weighted Round-Robin (WRR) [7] policy to dispatch requests. Figure 9 shows that new features and improvement in Linux kernel 2.6 really make web server more efficient. The translated LVS-CAD in Linux kernel 2.6 greatly outperforms LVS-CAD in kernel 2.4. Even there is only one back-end server in web cluster, the throughput in Linux kernel 2.6 is still twice higher than the throughput in kernel 2.4. Especially, in web cluster with eight back-end servers, LVS-CAD in kernel 2.6 outperforms LVS-CAD in kernel 2.4 by 158.22%.

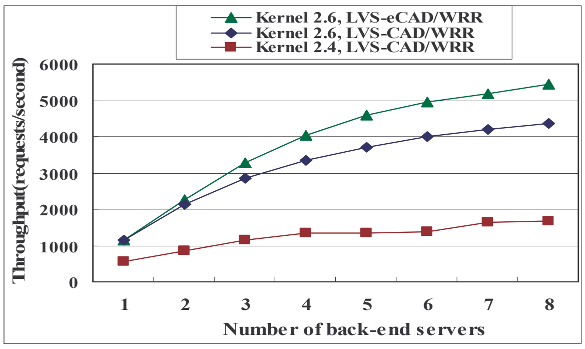


Figure 9. Scalability comparison

4.2 Reducing Multiple Handoffs

Because excessive multiple handoffs can degrade the performance of the whole web cluster under persistent connection, so we integrate the idea of reducing multiple handoffs into LVS-CAD. This new platform is named LVS with efficient Content-Aware Dispatching platform (LVS-eCAD). That is, no matter what policies have been adopted by the front-end server, LVS-eCAD always attempts to reduce excess multiple handoffs.

Figure 10 shows the flow of reducing multiple handoffs in LVS-eCAD. If the current request-handling server's number of active connections subtracts the new selected server's number of active connection is less than α, the request will be dispatched to the current request-handling server. This means that the difference of the loads between the current request-handling server and the newly selected back-end server is not obvious and not large enough, so it is not beneficial to hand off the connection to the selected back-end server. Otherwise, the request will be dispatched to the newly selected server. Therefore, α is used to determine if the request should be handed off to the selected back-end server.

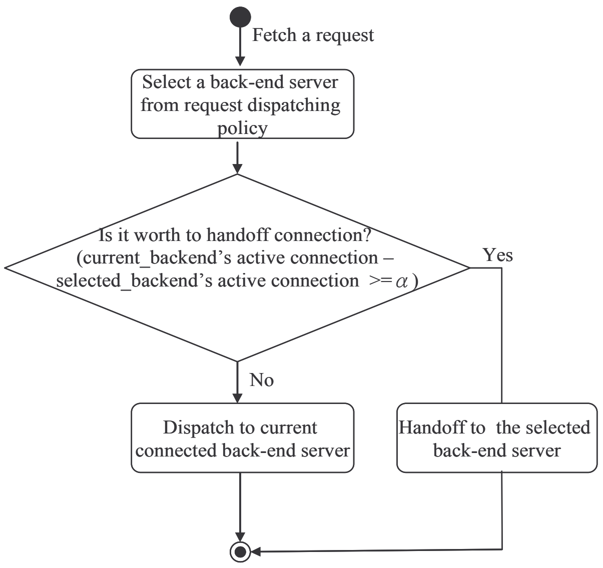

Fetch a request
Select a back-end server from request dispatching policy
Is it worth to handoff connection? (current_backend's active connection – selected_backend's active connection >= α)
Yes
No
Dispatch to current connected back-end server
Handoff to the selected back-end server

Figure 10. LVS-eCAD reduces multiple handoffs

Figure 11 shows the performance comparison of LVS-eCAD with different α value under the content-blind WRR policy. Besides, Figure 11 compares the LVS-eCAD with LVS and LVS-CAD under persistent connection environment and each connection contains ten requests. The performance of LVS is only slightly better than that of LVS-CAD policy under content-blind WRR policy. LVS-CAD is worse than that of the LVS platform. This is due to the fact that excessive multiple handoffs in LVS-CAD increase the server overhead to rebuild connections with clients. However, the throughput of LVS-eCAD platform is better than that of LVS-CAD and LVS platform by 24.62% and 23.17% respectively when the α is set to 1. Thus, reducing multiple handoffs can really improve the performance of a web cluster even if the request distribution policy is a content-blind policy.

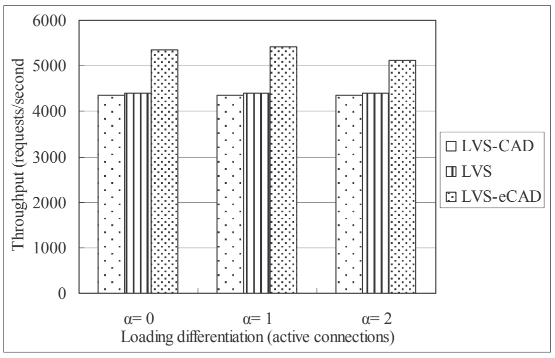


Figure 11. Performance comparison

5. Conclusions and Future Work

We have successfully transplanted the LVS-CAD platform from Linux kernel 2.4.18 to Linux kernel 2.6.18 to gain new features and performance improvement on Linux kernel 2.6. We have also implemented an efficient content-aware platform name LVS-eCAD to reduce excess multiple handoffs under persistent connection. Based on the LVS-eCAD platform, several issues could be further explored or enhanced, such as supporting the front-end server to transfer some TCP options negotiated during three-way handshaking to the selected back-end server, session persistence, etc.

References

[1] HP Labs, "A Closer Look at the Linux O(1) Scheduler," http://www.hpl.hp.com/research/linux/kernel/o1.php.
[2] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web site," Hewlett-Packard Technical Report, HPL-1999-35R1, Feb. 1999.
[3] C. Benvenuti, Understanding Linux Network Internals, First Edition, O'Reilly, Dec. 2005.
[4] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," ACM Computer Surveys, Volume 34, No. 2, Pages 263-311, June 2002.
[5] Ulrich Drepper and Ingo Molnar, "The Native POSIX Thread Library for Linux," https://news.wideopen.com/whitepapers/developer/POSIX_Linux_Threading.pdf.
[6] Li Ge, "Kernel Comparison: Web serving on 2.4 and 2.6,"http://www.ibm.com/developerworks/linux/library/l-web26/, Feb. 2004.
[7] Linux Virtual Server Website, http://www.linuxvirtual.server.org, Dec. 2007.
[8] H. H. Liu, M. L. Chiang, and M. C. Wu, "Efficient Support for Content-Aware Request Distribution and Persistent Connection in Web Clusters," Software Practice & Experience, Vol. 37, Issue 11, pp. 1215-1241, Sep. 2007.
[9] The httperf, http://www.hpl.hp.com/research/linux/httperf, 2007.
[10] The Internet Traffic Archive, http://ita.ee.lbl.gov, May 2007.