Design and Implementation of a Scalable Web Cluster System Using a Light-Weight Cluster Control Protocol

IN HWAN DOH and SAM H. NOH
Department of Computer Engineering
Hongik University
Seoul 121-791
KOREA

Abstract: - The need for web services is constantly increasing, while the requirements for web services are getting more and more complicated. Thus, web server systems with ever increasing performance are constantly in need. To efficiently satiate this need, web cluster systems, which have been a topic of much research, have been suggested due to its scalability. In this paper, we design and implement a scalable web cluster system using a light-weight cluster control protocol on the IP layer in the Linux operating system. Experimental results illustrate that our implementation of the web cluster system linearly improves performance as the server nodes increase. We show that although the web cluster control protocol implemented on the IP layer does not support any reliable mechanism, the problem related to packet loss or fault has little effect on the system performance.

Key-Words: - Web Cluster System, Web Server, Web Switch, Linux, Network Protocol

1 Introduction

With the continuous increase of Web users, the web server not only executes lots of complicated programs, but also contains web pages that include more and more heavy embedded objects such as image, sound, and movie clips. In this regard, web server systems with ever increasing performance are constantly in need. However, in order to address this need, it is not a cost-effective solution to continually purchase a server system that has ever more higher performance. As a cost-effective solution to satiate this need, a web cluster system has been suggested due to its scalability.

Although a web cluster system consists of several system nodes, it gives an illusion of a single system. One of the most important advantages of a web cluster system is that it is easy to extend its performance. One research goal related to web cluster systems is to design and implement a scalable web cluster system that linearly improves its performance as the number of internal server nodes increase. Considerable research has been conducted in an attempt to achieve this goal [1].

Generally, a conventional web cluster control protocol is implemented on the TCP layer. This control protocol can provide reliability in a web cluster system when the control packet disappears or is damaged. However, control protocols based on TCP are generally complex and heavy because they incorporate many complicated processing mechanisms in order to deal with the several exceptions that could possibly happen [2].

Considering current advanced network infrastructures, the ratio of packet loss or fault in LAN is very low. Studies have shown that the growth of network technology is around 2 times faster than that of computer system technology [3]. Through this study, the inference that packet loss or error will not be a concern any more in the near future is persuasive. In this respect, our goal is to explore the effect on system performance when a web cluster system is implemented on the IP layer, which does not support any reliability for the control packet.

If the web cluster system is implemented on the IP layer, the only way to ensure control packet reliability is to rely upon the retransmission mechanism of the TCP connection that is established between a client and the web cluster system. Even so, our experimental results illustrate that the web cluster system based on the IP layer shows scalable performance that is almost identical to an ideal system.

This paper describes the design, implementation and performance of a web cluster system that uses a Light-weight Cluster Control Protocol (LCCP) implemented on the IP layer. The following section reviews general approaches in implementing a web cluster system. Section 3 describes our design of the web cluster system. Section 4 presents a brief

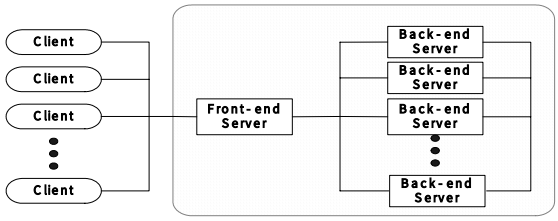

Fig.1: Architecture of a general web cluster system

description of the implementation of the web cluster system in Linux. Section 5 presents experiments that evaluate the performance, stability, and scalability of our web cluster system. The final section summarizes our work.

2 Related Work

Considerable studies have been conducted regarding web cluster systems and they can be classified into 3 different categories. In the first category, the studies consider the mechanisms used to construct and operate web cluster systems. In the second category, researchers consider developing algorithms that balances loads among several server nodes in the web cluster system [1]. In the last category, approaches to overcome the overloaded state in the web cluster system are considered [4]. In this section, we focus on the first category as this is most relevant to the topic of this paper.

The architecture of a general web cluster system is represented in Figure 1. A web cluster system consists of a front-end server, which is commonly called the web switch, and back-end servers. Web cluster systems are divided into layer-4 or layer-7 web cluster systems according to the information that is considered by the front-end server when distributing client requests to the server nodes. Web cluster systems are also classified into one-way or two-way architectures according to whether the front-end server relays the packet flows between a client and a back-end server in only one direction or in both directions, respectively [1].

The layer-4 web cluster system consists of a layer-4 web switch and back-end servers. The layer-4 web switch forwards packets from clients to the back-end server according to the destination IP address and the port number of the packet. There are many packet forwarding mechanisms in the layer-4 web switch such as NAT (Network Address Translation) [5] and IP Tunneling [6]. Currently, this type of layer-4 web switch showing high performance is available in the market [7, 8]. Furthermore, NAT and IP Tunneling

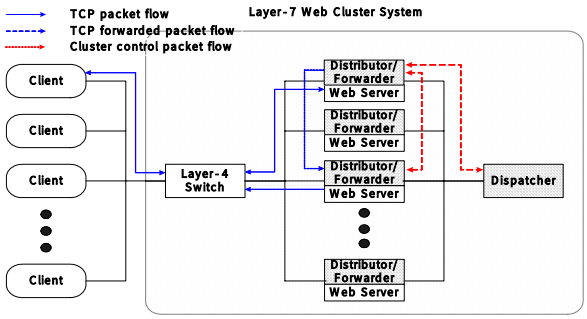

Fig.2: Architecture of Layer-7 web cluster system

techniques are being provided as patches or built right into the Linux kernel through the Linux Virtual Server Project [9, 10].

When layer-7 web cluster systems distribute the packets from clients to the back-end servers, the front-end server applies a content-aware distribution algorithm while considering the IP address and port number of the requested packet. TCP handoff [2] and TCP splicing [11, 12, 13] are two of numerous mechanisms that allow the front-end server to use content-aware distribution algorithms. TCP splicing adopts a two-way architecture, easily making the front-end server a bottleneck. To relieve this issue, the TCP handoff mechanism uses a one-way architecture, but this mechanism still suffers from limited scalability. Aron et al. resolve this problem by dividing the front-end server's functionality into the dispatcher that executes the layer-7 dispatching algorithm and the distributor that operates the TCP handoff mechanism [14]. The distributor is scattered to all of the server nodes.

3 Design of the LCCP

In this section, we describe the architecture of the web cluster system that our implementation is based on. Then, we describe the design of a scalable Light-weight Cluster Control Protocol (LCCP) that controls the web cluster system.

3.1 Components

Figure 2 shows the architecture and internal components of the web cluster system implemented in this paper. We use the layer-7 web cluster system design proposed by Aron et al. [14]. The web cluster system consists of a layer-4 switch, a dispatcher, and web server nodes that are internally composed of a distributor and forwarder. Each component works as follows.

A layer-4 switch is an interface between external

web clients and the web cluster system. Basically, a layer-4 switch passes SYN packets from web clients to server nodes in the web cluster system. A TCP connection is established between a client and the server node that receives the SYN packet. To preserve the TCP connection between the client and the server node, the layer-4 switch forwards subsequent packets to the server node which received the SYN packet. A dispatcher decides which server node will actually serve the web client request. This decision is based on the content of the web request. A distributor manages the TCP handoff mechanism [2], which hands off a TCP connection from one server node to another. Through the TCP handoff mechanism, it is possible to pass along a TCP connection among internal server nodes whenever the server originally connected to the client differs from the server that actually has to serve that client. A forwarder forwards subsequent packets such as ACKs and FIN to the server node that actually processes the web request, since the TCP handoff was already occurred.

3.2 Light-weight Cluster Control Protocol

Internal nodes of the web cluster system need to communicate with each other so that the web cluster system gives an illusion of a single system. This cluster communication mechanism is what controls the web cluster system.

The web cluster system proposed by Aron et al. establishes a permanent TCP control connection among all of the nodes during the initialization state [14]. Through this TCP control connection, the nodes of the web cluster system can pass along control packets to each other. The web server system based on HTTP/1.0 protocol usually needs to exchange 9 TCP packets between the server and the client in order to process a request. However, when the server that initially establishes a TCP connection with the client is different from the server that will serve the client, the web cluster system has to exchange at least 12 additional packets in order to serve the request [14]. This additional overhead can not be ignored. Furthermore, due to technological advances in network infrastructure, the rate of packet loss is extremely low in current Local Area Networks (LAN). Thus, using a TCP protocol that has extensive code to support reliable connections may be wasting resources.

In this respect, we analyze the performance degradation suffered from packet loss or faults for a web cluster system implemented on the IP layer when deployed on current LAN infrastructure. To do so, we design and implement a Light-weight Cluster Control Protocol (LCCP) on the IP layer. Our experimental analysis shows that an IP layer implementation to support web cluster systems is indeed feasible and efficient.

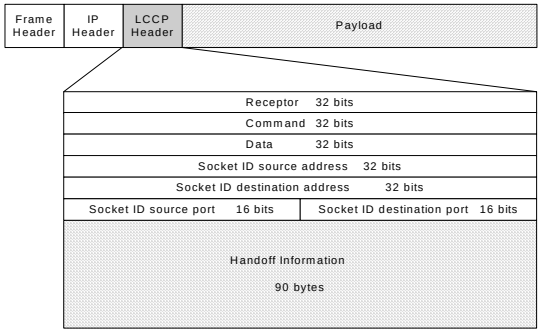

Fig.3: LCCP packet header

Figure 3 depicts the web cluster control packet using LCCP. This packet does not include the transport layer protocol header such as the TCP header. The header of the LCCP packet consists of only essential information required for control of a web cluster system. The details of the LCCP header are as follows.

Receptor (32 bits): This field indicates which component of the web cluster system has to process the current LCCP packet received from the IP layer.
Command (32 bits): Through this field, the component is able to identify what kind of mission has to be fulfilled when the component receives the LCCP packet. This field includes many kinds of tasks, for instance, the initialization of each component, the request or reply between the distributor and the dispatcher, the request or reply related to the TCP handoff mechanism, and so on.
Data (32 bits): This field includes the data indicated by the 'Command' field. The value of this field will vary according to the value of the 'Command' field.
Socket ID (96 bits): This field consists of four different subfields; the client's IP address, the client's port number, the server's IP address, and the server's port number. When the current node is not the server node that initially establishes the TCP connection with the client, the current node can obtain the initial TCP connection by exploiting these fields.
Handoff Information (90 bytes): When the server that establishes the initial TCP connection with the client is different from the server that actually provides service to the client, the distributor has to

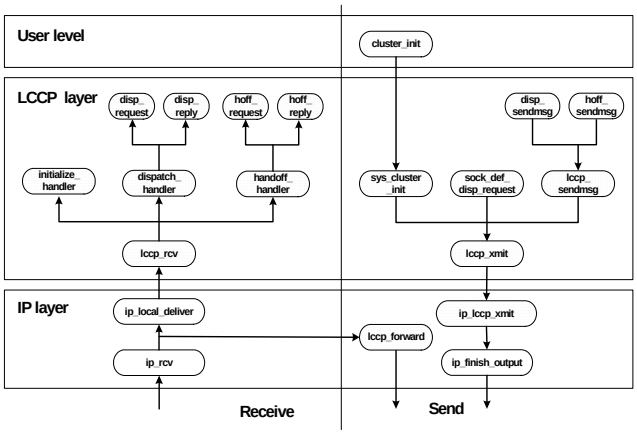

Fig.4: Function call graph for the LCCP

pass along the TCP connection information from one server to another. In this case, the distributor creates the LCCP packet whose 'Handoff Information' field contains the TCP connection information needed to support the TCP handoff mechanism.

4 Implementation in Linux

We implemented the web cluster system mentioned in the previous section in Linux 2.4.20. In this section, we briefly describe the implementation of LCCP.

Figure 4 shows the position of the LCCP protocol and their functions within the Linux network stack. The left side of this figure involves functions processed when the control packet arrives at the LCCP layer; the right side functions are executed when sending the LCCP packet. Due to the paper length limitation, we omit the details of each of these functions and only briefly mention the roles of these functions.

In Figure 4, `lccp_rcv()` and `lccp_xmit()` are both interface functions for receiving and sending the LCCP packet, respectively. Both `sys_cluster_init()` in the sending part and `initialize_handler()` in the receiving part initialize components of the web cluster system. The function `lccp_forward()` in the IP layer forwards the LCCP packet. In the distributor, `sock_def_disp_request()` and `disp_sendmsg()` are in charge of creating and sending LCCP query packets to the dispatcher, respectively, and `dispatch_handler()` processes the reply packet for the query. The `handoff_sendmsg()` and `handoff_handler()` functions execute the TCP handoff mechanism in the distributor. In the dispatcher, the function `dispatch_handler()` executes the layer-7 content-aware load distribution algorithm.

5 Performance Evaluation

In this section, we first explain the experimental setup used to measure the performance of the web cluster system. We then describe the performance of the LCCP web cluster system from the viewpoint of stability and scalability.

5.1 Experimental Setup

Table 1 describes the specification of the node systems constituting the LCCP web cluster system and the client systems generating web requests in our experiments. Each of the 14 server nodes and dispatcher has an Intel Pentium II 350MHz CPU and 64MB RAM. In order to easily saturate the capacity of the web cluster system, 7 client systems with higher performance than the others are used in the experiments. All clients have 256MB RAM. Of these, 4 systems, 2 systems, and 1 system, respectively, use an Intel Pentium IV 2.4GHz CPU, an Intel Pentium III 750MHz CPU, and an Intel Pentium IV 1.7GHz CPU. All of these systems have a 3COM 100Mbps Ethernet Card (3c59x-TX-M) and are connected to each other through the NETGEAR Gigabit switching HUB with CAT 5 STP LAN cables.

The Apache web server is adopted as the web server application operating on each server nodes [15]. In order to generate synthetic HTTP/1.0 web requests, each client executes the SURGE program [16]. The workload created by SURGE consists of 10,000 data files of which the total size is 187MB and the average size is 19KB. The minimum generated file size is 74 bytes and the maximum size is about 1.4MB. In this workload, the patterns that clients request to a server follow the Zipf-like distribution and the most popular file among the 10,000 files is requested 100,000 times when the client request made to the server is 490,000 times [17]. For the purpose of flooding the server with requests, the SURGE program is modified to send a HTTP request as soon as the reply for the previous request is received.

Table 1: System's specification

NODE	NUM.	CPU	RAM	LAN CARD
Server	14	Pentium II 350MHz	64M	3COM 100Mbps Ethernet Card (3c59x-TX-M)
Dispatcher	1	Pentium II 350MHz	64M	
Client	4	Pentium IV 2.4GHz	256M	
Client	2	Pentium III 750MHz	256M	
Client	1	Pentium IV 1.7GHz	256M	

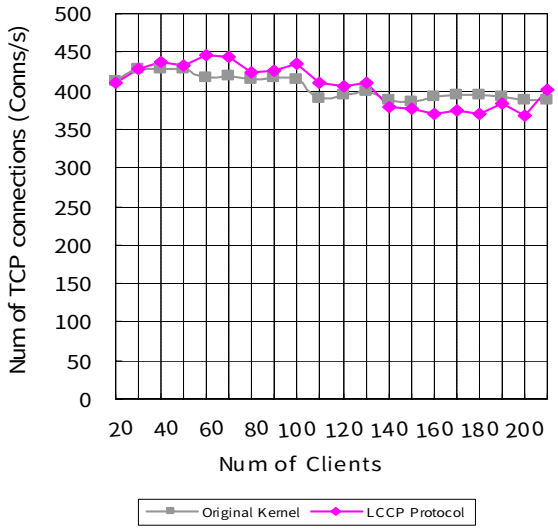

Fig.5: Stability of LCCP web cluster system

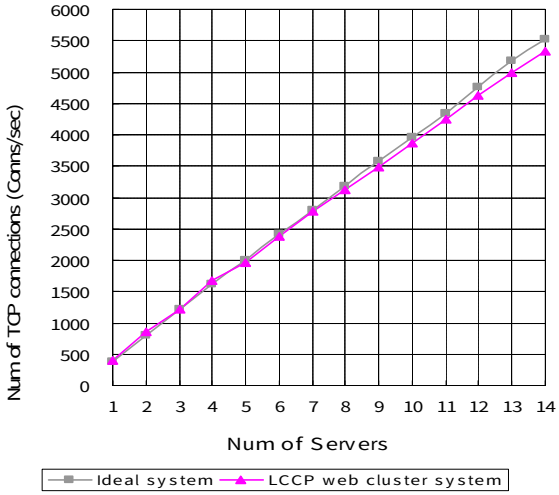

Fig.6: Scalability and performance of LCCP web cluster system as server node increase

5.2 Results

Since our intension is to estimate the stability and scalability of the LCCP protocol, we simply adopt the round-robin dispatching algorithm as this is one of the simplest algorithms used for load distribution. In the following experiments, we do not use a layer-4 switch as we assume that a layer-4 switch is in front of the web cluster system. However, it is not important whether a layer-4 switch is included or not because the layer-4 switch plays a simple role of relaying packets from clients to servers or vice versa.

5.2.1 Stability

Prior to describing the scalability of the LCCP web cluster system, we present the stability for our LCCP protocol implementation. To do this, we compare the LCCP web cluster system with an unmodified server system. In this experiment, the LCCP web cluster system is composed of only one server node that internally executes the TCP handoff mechanism all the time. The client floods HTTP requests to each of these two different web server systems, increasing the number of its SURGE threads from 10 to 200 in increments of 10. The number is limited to 200 threads as we found that one SURGE process executing 200 threads fully saturates a single web server node. In order to obtain a stable result, the experiments are conducted for 30 seconds of which the result data are gathered from the middle 20 seconds. This evaluation is repeated 10 times. Of these, we throw away the minimum and maximum values and obtain the average. Figure 5 shows these numbers.

Figure 5 represents the number of TCP connections serviced per second as the number of SURGE threads increases by 10. From this figure, we observe that the LCCP web cluster system performs comparably to the unmodified Linux kernel. In the LCCP web cluster system, the server node has to communicate with the dispatcher and internally execute the TCP handoff mechanism. Although the LCCP web cluster system needs to process additional operations in order to serve a web request, its performance is similar to the unmodified web server system performance. From this result, we can verify that the LCCP web cluster system which consists of only one server node stably operates in a LAN environment even if the LCCP protocol does not support any reliability mechanism for packet loss or error.

5.2.2 Scalability

Important performance metrics for the web cluster systems are scalability and capability, both measured by the number of TCP connections processed per second. In another experiment, we measured these two metrics, by adding a server node to the LCCP web cluster system one by one to upto 14 nodes. Each server node receives HTTP/1.0 requests generated by a SURGE process having 200 threads, so that the LCCP web cluster system is fully overloaded by these HTTP requests. The procedure for obtaining the result data in this experiment is the same as that described in Subsection 5.2.1. One client system can stably execute 2 SURGE processes with 200 threads, and each SURGE process can fully saturate one web server node. Considering this limitation and the 24 port switching hub that we have, the maximum number of server nodes, dispatcher, and client systems that can be experimented with is 15, 1, and 8, respectively. Due

to the number of systems that is available to us, we conduct this scalability experiment using 14 server nodes, 1 dispatcher, and 7 client systems. We compare the LCCP web cluster performance with the simplest version of the web cluster system, which does not support any load balancing mechanism. This system consists of server systems that have an unmodified Linux kernel, and each of these server systems are directly connected to each other through the switching hub. This system can directly service any client HTTP request without requiring any internal communication with other system components. Hence, this system will result in the best layer-7 web cluster system performance. Therefore, we will call this an ideal system in this experiment.

Figure 6 depicts the results that compare the LCCP web cluster system performance with the ideal system performance. The performance of the ideal system as well as the LCCP web cluster system increases linearly as the server node increases. Not only is the system scalable, but we observe that the number of TCP connections per second processed on the LCCP web cluster system is as high as that of the ideal system.

The implication of these experimental results is that the LCCP web cluster system can stably serve overloaded web requests even though the LCCP protocol does not support any reliability mechanism for packet loss and/or error.

6 Conclusions

To identify the effect on the system performance when a web cluster system is implemented on the IP layer, we designed and implemented the LCCP web cluster system on the IP layer in Linux.

Experimental results show that the LCCP web cluster system is scalable and that performance is as high as that of an ideal system. Considering the current advanced network infrastructure and our experimental results, we conclude that although the web cluster control protocol implemented on the IP layer does not support any reliable mechanism, the problem related to packet loss and/or error has little effect on the system performance.

7 Acknowledgments

This work was supported by the Brain Korea 21 Project in 2006.

References:

[1] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, The state of the art in locally distributed Web-server systems, *ACM Computing Surveys (CSUR)*, Vol.34, No.2, 2001, pp.263-311.

[2] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, Locality-aware Request Distribution in Cluster-based Network Servers, *In Proc. of the 8th International Conference on ASPLOS*, 1998.

[3] G. Gilder, Fiber keeps its promise: Get ready. Bandwidth will triple each year for the next 25, *Forbes*, Vol.7, 1997.

[4] M. Andreolini and E. Casalicchio, A Cluster-Based Web System Providing Differentiated and Guaranteed Services, *Cluster Computing*, Vol.7, No.1, 2004, pp.7-19.

[5] P. Srisuresh and K. Egevang, Traditional IP Network Address Translator, RFC 3022, 2001.

[6] C. Perkins, IP encapsulation within IP, RFC 2003.

[7] Cisco Systems Inc., http://www.cisco.com.

[8] Netgear Inc., http://www.netgear.com.

[9] Linux Kernel Archives, http://www.kernel.org.

[10] The Linux Virtual Server Project, http://www.linuxvirtualserver.org.

[11] D. Maltz and P. Bhagwat, Application layer proxy performance using TCP splice, Tech. Rep. RC 21139, IBM T. J. W. Research Center, 1998.

[12] A. Cohen, S. Rangarajan, and H. Slye, On the performance of TCP splicing for URL-aware redirection, *In Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems*, 1999.

[13] O. Spatscheck, J. S. Hansen, J. H. Hartman, and L. L. Peterson, Optimizing TCP forwarder per-formance, *IEEE/ACM Trans. Networking*, Vol.8, No.2, 2000, pp.146-157.

[14] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, Scalable Content-aware Request Distribution in Cluster-based Network Servers, *In Proc. of the USENIX 2000 Annual Technical Conference*, 2000.

[15] The Apache Software Foundation, http://www.apache.org.

[16] P. Barford and M. Crovella, Generating Repre-sentative Web Workloads for Network and Server Performance Evaluation, *In Proc. of the ACM SIGMETRICS'98*, 1998.

[17] L. Breslau, P. Cao, Li fan, G. Phillips, and S. Shenker, Web Caching and Zipf-like Distribu-tions: Evidence and Implications, *In Proc. of Infocom'99*, 1999.