Scalable Contents-based Web Cluster Server with Self-Processing Web Switch

Young hwan Kim¹, Hee Yong Youn², Hyoung Su Lee¹, Chang Won Park¹

¹ IT System Research Center, Korea Electronics Technology Institute
Pyoungtaek, Korea
² School of Information and Communications Engineering
Sungkyunkwan University
Suwon, Korea
{yhkim93, youn}@ece.skku.ac.kr, {hslee, cwpark}@keti.re.kr

Abstract

This paper describes the motivation, design and performance of web cluster server with self-processing web switch for enhanced packet throughput. The goal of the proposed web cluster server is to provide a framework for building highly scalable, fault-tolerant services using a large cluster of commodity servers. It employs web switches processing packets by themselves while operating at contents-based layer-7 of the OSI protocol stack. Extensive computer simulation reveals that the proposed approach improves packet throughput and reply rate about 20% compared to earlier designs.

Keywords: Contents-based, layer-7 switch, load balancer, scalability, web cluster server.

1 Introduction

As the application of Internet pervades every aspect of life, network traffic has grown exponentially and thus efficient web service has become a crucial factor. For some popular web sites, unbearably long response time or even no accessibility may occur. The need to optimize the performance of popular web sites stimulated a variety of novel architectures. In this paper we consider a special web system, namely web cluster, that uses a tightly coupled cluster architecture. From the user’s point of view, any HTTP request to a web cluster is presented to a logical server that acts as a representative for the web site.

In a web cluster server, web switch is responsible for dispatching client requests to the servers. It can be classified according to the OSI protocol stack layer at which they operate (e.g. layer-4 and layer-7 web switch). The web switch retains transparency of parallel architecture for the user, guarantees backward compatibility with Internet protocols and standards, and distributes all client requests to the back-end servers. Cluster architectures employing web switch have been adopted in various academic and commercial web clusters [1-3, 5].

The web cluster server can be classified into two types according to the web switch employed layer-4 or layer-7 web switch. Layer-4 web switches work at TCP/IP level. Since packets pertaining to the same TCP connection must be assigned to the same server node, the web switch has to maintain a binding table to associate each client TCP session with the back-end server. This approach may increase the CPU workload of all the hosts in the web cluster server for packet filtering and distribution. Since all the hosts individually have to process the task of load balancing, the service capability of each node may also be decreased. Layer-7 web switches can establish a complete TCP connection with the client and inspect the contents of HTTP request prior to a decision on dispatch. This can be based on the web service/content requested, URL content, SSL identifiers, and cookies. However, it introduces additional processing overhead at the web switch and may cause it to become the system bottleneck [4, 7].

In order to solve this problem we propose self-processing web switch acting at Layer-7 that directs the request packets to back-end servers. The self-processing web switch makes parallel services of the web cluster sever appear as a virtual service on a single IP address. This allows highly scalable and fault-tolerant services. Computer simulation shows that the proposed scheme improves packet throughput and reply rate about 20% compared to earlier approaches. Note that the proposed self-processing web switch acts as web switch and back-end server at the same time. The former is to distribute the requests, while the latter is to directly serve the request packets. The dispatch algorithm used in web switches is also an important factor affecting the performance of the web switch.

The rest of the paper is organized as follows: the next section gives some brief examples of web cluster server and the dispatch algorithms. Then Section 3 describes the proposed self-processing web switch that we have implemented and tested. In Section 4 we show the performance of the proposed scheme. Finally, in Section 5 we conclude the paper with a summary.

2 Related Work

2.1 Web Cluster Server Architecture

Here we consider web cluster server consisting of homogeneous servers that provide a same set of documents and services, and a dedicated machine acting as a Layer-4 or Layer-7 web switch. The primary DNS
switches \[4-6,10\]. While the non-active web switch is named as slave web switch, the initial active web switch is named as master web switch. Another web switch can be used for fault-tolerance.

Directing the requests from clients to back-end servers.

Server architecture. Generally, there is one web switch in the cluster. The following Figure 1 shows a web cluster architecture. Generally, there is one web switch directing the requests from clients to back-end servers. Another web switch can be used for fault-tolerance. Initial active web switch is named as master web switch, while the non-active web switch is named as slave web switch [4-6,10].

[Image 76x415 to 266x519]

Figure 1. The cluster server architecture.

2.2 Web Switching Algorithms

The web switch can use various global scheduling policies to assign the load to the hosts of a web cluster server. The scheduling methods are classified into several ways depending on the criteria, and there are generally three aspects. Load balancing vs. load sharing problem, centralized vs. distributed algorithms, and static vs. dynamic policies. The web cluster architecture with a single web switch motivates the choice for centralized scheduling policies. If we consider that load balancing strives to equalize the server workload while load sharing attempts to smooth out transient peak overload periods on some nodes, a web switch should aim to share cluster workload more than to balance them. The layer-4 and layer-7 web switch need system information to make assignment decisions. The web switch cannot use highly sophisticated algorithms because it has to make fast decisions for dozens or hundreds of requests per second [8-9].

Static algorithms: The static algorithms are fast because they do not rely on the current state of the system at the time of decision making. Typical examples are Random (RAN) and Round Robin (RR) algorithm. RAN distributes the arrivals uniformly throughout the nodes. RR scheduling algorithm directs the network connections to different servers in the round-robin manner. It equally treats all back-end servers regardless of the number of connections or response time.

Dynamic algorithms: Dynamic algorithms require mechanisms that collect, transmit, and analyze the state information, thereby incurring potentially high overheads. We consider the following three classes of dynamic algorithms. First, server-aware algorithms route requests on the basis of some server state information such as load condition and latency. Second, clients-aware algorithms route requests on the basis of some basic client information. Layer-4 web switches can use only some basic client network information such as IP address and TCP port. Layer-7 web switches can examine the entire HTTP request and make decisions on the basis of detailed information on the content of the client request. Third, client and server-aware algorithms route requests on the basis of client and server state information.

3 The Proposed Scheme

3.1 The Basic Concept

In earlier designs master web switch and slave web switch are used to improve fault tolerance and availability. Here even though the master web switch fails, existing slave web switch can provide continuous service. Let us here study the function of web switch in more detail. It distributes incoming packets to each back-end server. If there is no back-end server managed by ipvsadm of kernel, the master web switch can become a fail-back server to continually serve the clients as a cluster web server. It, however, cannot be used as fail-back server if it assumes only the packet distribution job. We, therefore, propose to increase the throughput of the cluster web server by utilizing the master web switch also as a fail-back server as needed.

In normal condition slave web switch checks the status of master web switch through an agent. When the master web switch fails, it takes over the role of master web switch. We thus propose to utilize it as a back-end server if no failure occurs. With the proposed approach of self-processing master and slave web switch, throughput can be improved significantly since two more servers, the master and slave web switch, can serve the client requests in normal operation condition.

3.2 The operation

The operation of the cluster web server with self-processing web switch is as follows. Firstly, the client uses the TCP/IP protocol to connect to the chosen dispatcher. The dispatcher accepts the connection and parses the client request. After the web switch accepts the packet, it decides whether it processes the packet by itself or passes it to a back-end server. If it distributes the packet to a back-end server using the TCP handoff.
protocol [2, 6], it can start to serve other clients. Figure 2 shows how a web switch schedules the request packets. Here we focus on the function of web switch.

In the figure two web switches, two back-end servers and nine specific objects (A,B,C,D) in the incoming request stream exist. An object can be classified into text, static HTML page, and image etc. The master web server directs all requests for A to itself, and all requests for D to the slave web switch, and the other B and C to each back-end server. By doing so, the possibility that the requests find the requested objects in the cache at each server will increase. Also, packet throughput will be increased because web switch can operate as a back-end server.

- **Self-processing master web switch**

Self-processing master web switch in the proposed cluster web server acts as shown in Figure 3. First, the web switch has two Ethernet cards, one for web switch, the other for back-end server. The operations are as follows.

(1) A client sends a request to the Apache web server, which is set by a Virtual IP address.
(2) The Ethernet address of the web switch acting as a master web switch is selected by heartbeat daemon.
(3) After the web switch accepts the packet, it decides to process it or passes it to a back-end server.
(4) If the web switch decides to process the packet by itself, it directly serves the client. If it distributes the packet to a back-end server, it starts to serve the client as a web switch.

- **Cluster server with self-processing web switch**

We propose cluster web server with self-processing web switches to enhance fault-tolerance and scalability. The incoming packets from router are multicast to the web switches. The self-processing web switch achieves load balancing using a load balancing algorithm. The self-processing web switch passes the packets to back-end servers including itself and slave web switch for maximizing the performance. For example, if there are two web switches and two back-end servers, four servers can service the clients.

![Figure 2. Contents-based scheduling method.](image)

![Figure 3. Self-processing master web switch.](image)

(1) As seen in Figure 4, we can build up a cluster web server consisting of a cluster of web switches to enhance fault tolerance and a cluster of back-end servers to increase scalability. Firstly, as introduced in the previous subsection, the cluster of web switches directly forwards incoming packets to the clients to increase packet throughput or the cluster of back-end servers forwards incoming packets.

In this mechanism, the web switches control the sub-clusters and distribute the requests. In addition, it acts as a back-end server if no packet comes in. The main advantages of the proposed scheme are higher availability, fault tolerance, scalability, and easy implementation. The sub-clusters controlled by each web switch are independent of each other. Therefore, even though one of the web switches fails, the others can continuously provide the service. As a result, availability of the cluster system can be significantly increased. In addition, since sub-cluster can be easily added as needed, the scalability is high.

4 **Performance Evaluation**

We use httperf [2,3], a tool measuring web server performance. It provides a flexible facility for generating various HTTP workloads and measuring server performance. Our primary goal is to stress the cluster web server (i.e. the self-processing web switch), and thus we keep our workload as simple as possible. The clients are assumed to generate requests based on the HTTP1.1 protocol. The Apache configuration file on each web switch and back-end server is altered to disable logging and keep a sufficient number of httpd daemons available to minimize the overhead in responding to client requests. Also, in order to maximize the number of concurrent connections a given client could create, it is necessary to
increase the number of per-process file descriptors as well as system wide limit on files and local port number.

All web servers are essentially of the same type of hardware, and thus we configure them with equal weight. The hardware used by the self-processing web switch was chosen such that it is a representative of a typical system on the market today. The system has 256MB SDRAM, 500MHz Intel Pentium III. The web switches are equipped with two 3Com PCI Ethernet cards, each connected to 3Com 100MB switching HUB. There are two back-end servers each containing a 500MHz Intel Pentium III. Each system was booted with one 256MB of SDRAM and 3Com PCI Ethernet card. The results here are based on a series of test runs as shown in Table I.

<table>
<thead>
<tr>
<th>Web switch</th>
<th>Back-end Server</th>
<th>Layer</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num.</td>
<td>Self-</td>
<td>Num.</td>
<td>Used</td>
</tr>
<tr>
<td></td>
<td>processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As shown in Table I, we tested three cases. The simulation result shows that the proposed self-processing web switches in layer-7 is more efficient than the system with ordinary web switches or self-processing web switches operating in layer-4 in terms of response time and packet throughput. Refer to Figure 5 and Figure 6. Observe that the improvement gets more significant as the connection rate increases.

5 Conclusion

The cluster web server has a potential for scalability and cost-effectiveness. In this paper we have presented a self-processing web switch scheme that directs the requests to back-end servers and slave web switch, while the master web switch operates as a back-end server when no packet arrives. This allows highly scalable and fault-tolerant services. Computer simulation shows that the proposed scheme significantly increases packet throughput and decreases response time, especially when the connection rate is high. Also, layer-7 content-based routing increases the performance compared to layer-4 based switching.

In the future, we need to further maximize the performance of the proposed scheme by developing an adaptive job distribution mechanism according to the type of application as well as job distribution.

Reference