The Design and Implementation of Content Switch On IXP12EB

Thesis Proposal

by

Longhua Li

Computer Science Department

University of Colorado at Colorado Springs

5/15/2001

Approved by:

Dr. Edward Chow (Advisor) _____

Dr. Jugal Kalita _____

Dr. Charlie M. Shub _____

1 Introduction to Content Switch

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 [1, 2]. The workload on the servers is increasing rapidly so that servers will be easily overloaded for a short time, especially for a popular web server. To overcome the overloading problem of the servers, there are two solutions. One is the single server solution, i.e., to upgrade the server to a higher performance server, but it will soon be overloaded when requests increases, and require upgrade again. The upgrade process is complex and the cost is high. The other is the multi-server solution, i.e., to build a scalable server on a cluster of servers[1,2]. When load increases, we can simply add a new server or more into the cluster to meet the increasing requests. A very efficient way to accomplish this is to use a load balancer to distribute load among servers in the cluster. Load balancing can be done in two levels, transport level using the layer 4 switch or application level using the content switch [3].

1.1 Layer 4 Switching/Transport Level Load Balancing

Linux Virtual Server is an example of transport level load balancing approach. The basic goal of the Linux Virtual Server Project is to:

> *Produce a high-performance and highly available server for Linux based on clustering, which provides a good scalability, reliability and serviceability. [1]*

The LVS (Linux Virtual Server) is a highly scalable and highly available server built on a cluster of real servers. The architecture of the cluster is transparent to end users, and the users see only a single 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 advantage of layer 4 load balancing in switching is the overhead of load balancing is small and the maxim number of real server nodes can reach 25 or up to 100 [1]. The reason is when a user request arrives at the load balancer, the load balancer only examine the source IP and port number of the incoming packet, the packet matching process can be easily speeded up with the hash table based on these two fields. The common problem of layer 4 switching is that it is content blind, and does not take the advantages of the content information in the request messages [4] [5].

1.2 Content Switching/Application Level Load Balancing

Application level load balancing (also known as content switching) provides the highest level of control over the incoming web traffic. When making a load balancing decisions, the content switch can check the header/content including HTTP meta header, URL, the

pay load of the incoming packet, rather than simply checking TCP/UDP port number or IP address. By examining the content of the request, these switches can make decisions on how to route the request to the real servers. The content switching system can achieve better performance through load balancing the requests over a set of specialized web servers, or achieve consistent user-perceived response time through persistent connections, also called sticky connections [4].

1.2.1 LCS (Linux-based Content Switch)

The Linux-based Content Switch (LCS) is based on the Linux 2.2-16.3 kernel and related LVS package [1] implemented by Weihong Wang. LCS examines the content of the request, e.g., URL in HTTP header and XML payload, besides its IP address and port number, and forwards the request to real servers based on the predefined content switching rules. Content switch rules are expressed in terms of a set of simple 'if' statements. These if statements include conditions expressed in terms of the fields in the protocol header or pattern in the payload and branch statements describing the routing decisions [5].

2 Introduction to IXP1200

The Intel® IXP1200 Network Processor [6] is the cornerstone of the Intel® Internet Exchange Architecture (Intel® IXA) [7]. It combines the best attributes of a network ASIC with the flexibility, performance, and scalability of a programmable embedded processor to accelerate development of next-generation Internet products. The IXP1200 Network Processor is specifically designed for network control tasks, such as wire-speed switching and routing of packets or cells in real time.

2.1 IXP12EB

The IXP12EB Ethernet Evaluation Kit is a powerful tool for developing and verifying hardware and software for the IXP1200 Network Processor. The kit supports the IXP12DE software development environment for programming the Network Processor's microengines and integrated Intel StrongARM* processor core.
The IXP12EB is already set up in our lab. The configuration is shown in Figure 1.

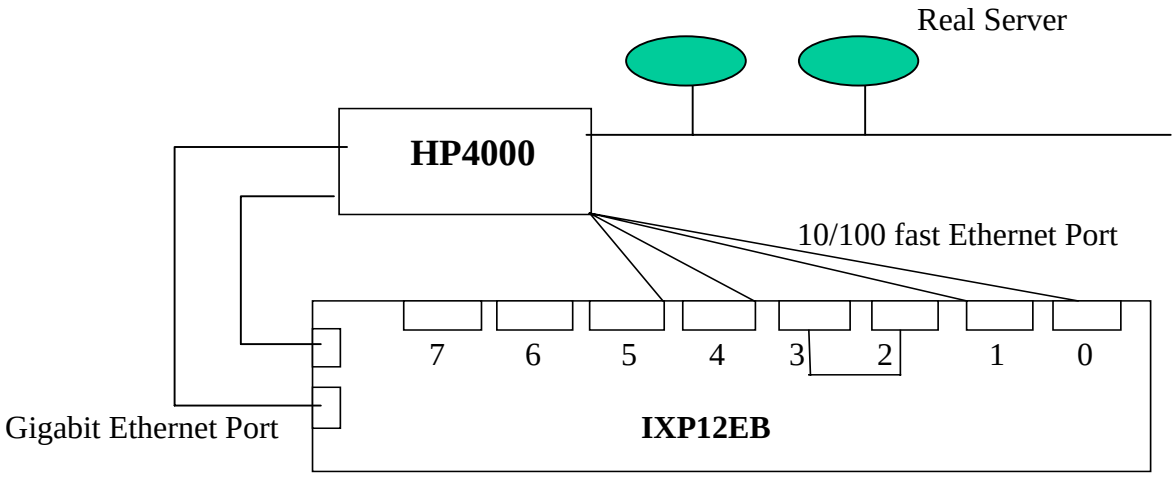

Figure 1. IXP12EB set up

3 Introduction to WindRiver VxWorks and Tornado IDE

Tornado[8] is an integrated environment for software cross-development. It provides an efficient way to develop real-time and embedded applications with minimal intrusion on the target system. Tornado comprises the following elements [8,9]:

- VxWorks [10], a high-performance real-time operating systems.
- Application-building tools (compilers and associated programs)
- An integrated development environment (IDE) that facilitates managing and building projects, establishing and managing host-target communication, and running, debugging, and monitoring VxWorks applications.

The Tornado environment is designed to provide this full range of features regardless of whether the target is resource-rich or resource-constrained. Tornado facilities execute primarily on a host system, with shared access to a host-based dynamic linker and symbol table for a remote target system. Communication between the host tools and VxWorks is mediated by the target server and target agent.

The development environment is already set up in our lab as shown in Figure 2.

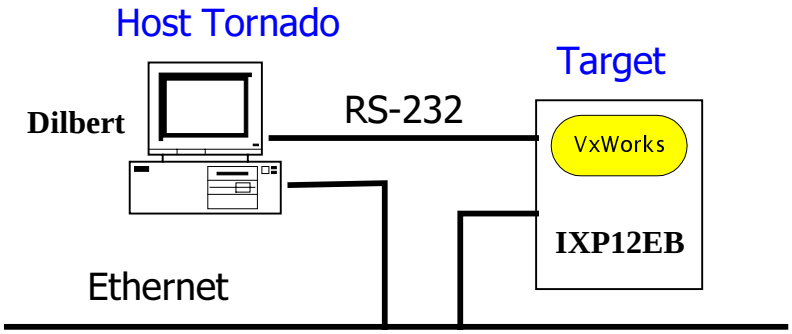

Figure 2. Development Environment set up

4 Goal

The goal of this thesis work is to design and implement an efficient content switch on IXP12EB. Since LCS (Linux Content Swtich) has already been developed, which can load balance the web requests based on the transport and the application layer header and content, I will concentrate port it onto IXP12EB. The tasks include:

- Modify LCS and port it onto IXP12EB by considering the parallel processing capability and the heterogeneous processor architecture of IXP1200. We will call it NPCS (Network Processor Content Switch).
- Analyze the performances between LCS and NPCS.
- Investigate possible improvements on NPCS.

5 Thesis Plan

The thesis will include the following activities:

- March-April 2001, Study WindRiver VxWorks and Tornado IDE.
- May 2001, Study and analyze the existing content switch technologies, especially LCS.
- June-July 2001, Design and implement the content switch on IXP12EB.
- August-September 2001, Test functionality and reliability of the design.
- October 2001, Compare the performances of LCS and NPCS.
- November 2001, Document the research results.

5.1 Design and Modification

This work will base on LCS version 0.2 which is first implemented by Weihong Wang and then improved by Chandra Prakash. The content switch should distribute the incoming requests to different web servers based on the content switch rule set. The five different classes of routing policies are as follow:

1. Based on the source IP address and TCP/UDP port number.

2. Based on the URL regular expression.

3. Based on the HTTP meta header.

4. Based on values of XML tags.

5.2 Implementation

Because IXP1200 is a fairly new product, there are few products available on the market, not much research has been done, and very few software support tools and source code for it. Implementing efficient software is going to be very challenging. I will reuse the code of LCS and port it onto the IXP12EB.

5.3 Testing

The implementation will be tested thoroughly on the advanced content switch testbed in our computer science lab. The network configuration will include one content switch (IXP12EB), four real servers. The IXP12EB performance will be compared against to LCS running on Linux PC.

5.4 Deliverables

The deliverables will include:

- Design documentation for the content switch.

- Source code for implementing the NPCS on IXP1200.

- Testing documentation.

References

[1] "Linux Virtual Server", http://www.linuxvirtualserver.org

[2] High Performance Cluster Computing: Architectures and Systems, Vol. 1&2, by Rajkumar Buyya (Editor), May 21, 1999, Prentice Hall.

[3] Gregory Yerxa and James Hutchinson, "Web Content Switching", http://www.networkcomputing.com.

[4] C. Edward Chow, "Introduction to Content Switch," http://cs.uccs.edu/~cs622/doc/contentsw.ppt.

[5] C. Edward Chow and Weihong Wang, "Design and Implementation of a Linux-based Content Switch," to be published in Proceedings of Second International Conference on Parallel and Distributed Computing, Applications, and Techniques. http://cs.uccs.edu/~chow/pub/contentsw/status/chow1.doc

[6] Intel® IXP1200 Network Processor http://developer.intel.com/design/network/products/npfamily/ixp1200.htm

[7] Intel® IXA (Internet Exchange Architecture), http://developer.intel.com/design/ixa/index.htm

[8] WindRiver Tornado Development Tools, http://www.windriver.com/products/html/tornado2.html

[9] Tornado User's Guide (Windows Version) 2.0

[10] WindRiver VxWorks, http://www.windriver.com/products/html/vxwks54.html