

Hierarchical Cluster for Scalable Web Servers^{*}

Jonghyuck Hong and Dongseung Kim

Department of Electrical Engineering
Korea University, Seoul 136-701, Korea
e-mail: dkim@classic.korea.ac.kr

Abstract

Cluster architecture is widely used for the availability and scalability of web servers in these days. To increase the performance we have developed a hierarchical architecture for web servers that can accommodate wide range of connection requests by gradually expanding the nodes in proper levels in the hierarchy. Especially, multiple dispatchers can distribute the incoming client requests to various backend servers in the cluster. Experimental results with two-level cluster with dynamic load balancing are included to verify the idea.

1. Introduction

Clusters of high-performance PCs/ workstations are broadly accepted for high-performance reliable computing with moderate cost compared to high-end commercial servers. In the WWW applications multiple servers (back-end nodes) in the cluster respond clients' requests by forming a worker farm of identical tasks or as dedicated servers in case of partitioned contents. To give a transparent access on a given web site, the cluster usually have a low-level router/dispatcher that either forwards the clients' requests to proper backend nodes using round-robin scheduling [3], or distribute them based on the contents the clients look for [1,5]. For faster response and better throughput, content-based server partitioning may be used. To have a better control the dispatcher needs to keep track of previous connection history, current load status, etc. If more factors are included and more clients are requesting connections, the load of the dispatcher increases rapidly, and it eventually becomes a bottleneck in providing desired web services with a reasonable latency.

Also some nodes may be allocated jobs beyond their service capability. In these cases the corresponding servers/dispatchers should be expanded easily not to saturate the overall performance. This is the problem that we are to solve in this research, based on hierarchical architecture and dynamic load balancing. Previous systems employ at most two-level hierarchy, front-end and back-end nodes, and the front-end node usually consists of a single packet-router. We can increase the number of nodes in any level as needed, and can increase the number of levels according to the content partitioning requirement, thus, we are able to retain the scalability of web servers.

2. Hierarchical cluster architecture

The hierarchical architecture is a generalization of web servers built with cluster of workstations as illustrated in Figure 1. The top-level of the cluster consists of multiple nodes serving as front-end nodes with which each client initially contacts to get into the web site. The number of nodes is determined by the requirement of the maximum request forwarding or distribution rate. The second level and the levels below include backend servers that actually perform web services. They may consist of a single level (laterally dispersed) or multiple levels according to the partitioning requirements of the contents or applications. For simplicity it is assumed that all nodes in the same level in a group deal with the same contents or applications.

Since the web site address (IP address) is uniquely known outside, all requests arrive at the same IP address. If clients could know addresses of all

^{*} This research was supported by Korea University Special Research Grant, March 2001.

front-end nodes and the requests arrive randomly/or evenly at all front-end nodes, the scalability of them could be achieved. We divide arriving requests internally to different nodes in a time-shared manner using HTTP redirection described as follows [4,7].

Top level nodes or front-end nodes work simultaneously except only one, called a *master node* which serves as a virtual IP server that bears the IP address known to outside world for a given time interval, and the virtual IP address circulates among the front-end nodes. The master node captures all incoming packets toward the known site and redirects them to other front-end nodes.

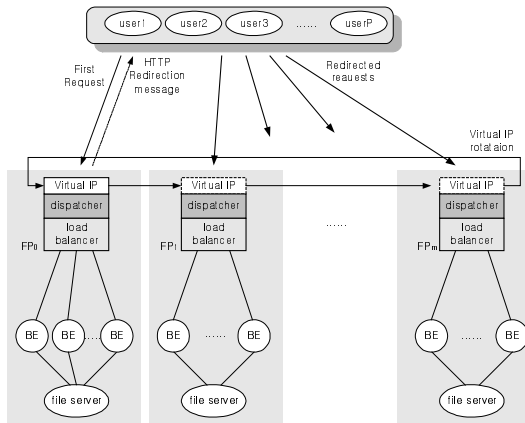


Figure 1. The architecture of the hierarchical web server

When a client request firstly arrives at the site, it contains the default port address, whereas if it is the second or later it would have the address different from the default. Each front-end node examines the port address in the request: if it is the default (say 8080) then it is sent back (redirected) with one IP address of the front-end nodes with other port number (say 8000) as shown in Figure 2. Or the front-end node forwards it to one of its backend nodes for getting the request serviced.

Our first attempt to remove the bottleneck was to add more nodes to the front-end, and let the single dispatcher forward requests to other front-end nodes for further processing. It failed since the cost of forwarding was about the same as the one to be needed for processing there, thus there were no performance gains from forwarding (even

worse since forwarded request eventually got serviced but the time lapsed more).

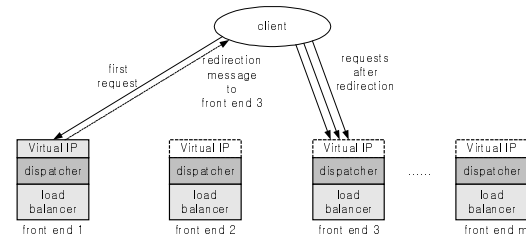


Figure 2. Redirection of client requests

After the redirection of the first connection requests, the redirected one arriving at the given node is guaranteed to get through. Thus, all redirected ones can proceed without further delay and also they arrive in different nodes, thus, the scalability of the front-end nodes is achieved. Of course this scheme lengthens the connection time of the clients. In addition, redirection may result in uneven job distribution due to different clients' browsing activities although it disperses the requests evenly. Thus, the load of each front-end nodes is collected at the master node, which is used in selecting the redirection node of the initial request to make truly even distribution of the clients' requests. Thus, front-end nodes use the weighted round-robin scheduling.

3. Dynamic load balancing

Backend nodes are grouped/partitioned based on the contents they use for web clients, thus, nodes in the same group can provide the same services. The front-end nodes in charge of a group forward client connection requests to backend nodes, which may cause load imbalance among them when they do only by the round-robin scheduling.

Backend load balancing is performed in two ways. Firstly, the front-end node associated with the group selects one backend node for a particular request in the group base on the round-robin request in the group. The front-end node periodically receives the load figures from backend nodes, with which it determines the rate of dispatching new requests to the corresponding node. Next, backend nodes

also reroute some jobs to other lightly loaded nodes if their load figures exceed the given threshold.

4. Experimental results and discussion

The hierarchical web server has been built and experimented on a 15-node linux cluster of identical PCs of a 300MHz Pentium-II CPU with 128 MB memory interconnected by a Fast-ethernet switch. Each runs Apache web server, and supports HTTP redirection and direct routing developed in the linux virtual server project [6]. They are divided into 3, 6, and 6 nodes for front-end, back-end, and clients, respectively. Client nodes generate requests to read from or write on a given set of synthetic static/dynamic web documents. We use four web documents with different sizes for read, and four dynamic contents with different work load needed. Throughout the experiment, client nodes continuously generate requests for web documents selected randomly such that all servers in the web site are busy.

Firstly, we like to check the scalability of the cluster in various levels separately. Figure 3 plots the throughput versus the number of front-end nodes. It is measured by the number of clients' requests forwarded to backend nodes per unit time. The throughput obtained in a single node is the maximum. The plot reveals that the system yields the scalable performance in the top level.

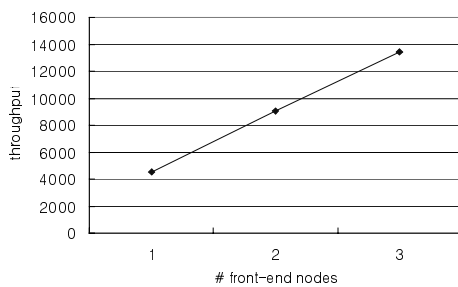


Figure 3. Throughput vs. the number of front-end nodes

In addition, Figure 4 shows the maximum number of requests serviced by backend servers, where each transmits data read from a few simple docu-

ments with different sizes. The client requests arrive at a maximum rate too. The curve on the top for the small sized web document is not so linear, where the overhead of the http demon in Apache server is relatively high, thus, it impairs the service throughput.

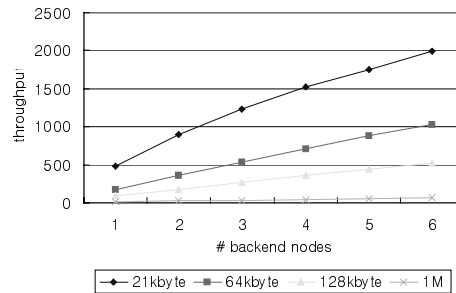


Figure 4. Throughput vs. the number of front-end servers with various web document sizes

Now, dynamic load balancing is implemented on the hierarchical web server to maximize the utilization of the system [1,2]. Each backend server has its own static and dynamic web documents to be processed upon the request of clients, and the CPU load varies depending on the documents. There are four different documents and their relative load indexes (data size) for processing each are set as 8:1:1:1. With the simple round-robin scheduling, requests are distributed evenly, and the throughput is measured approximately 8:1:1:1 as shown in Figure 5(a). Now with the dynamic scheduling, the relative values of the numbers of connections are close to 1:8:8:8, so the amount of work performed in each server is about the same, (balanced) as shown in Figure 5(b).

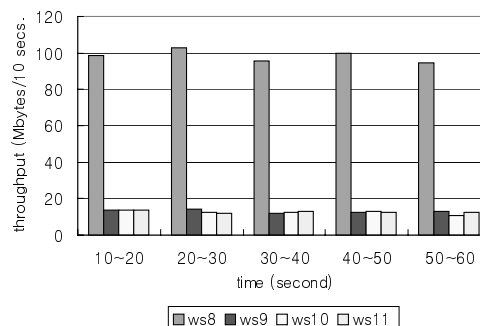
Finally, we simulate abrupt load changes at runtime in a certain backend node and observe the load balancing activities. All backend nodes handle the same number of requests except one, which deals with higher load request. The high loaded server is forced to rotate among backend nodes periodically with a 10-second interval. Figure 6 shows the snapshot of the number of clients' requests serviced by all backend nodes when the service of one node requires the work load of *twice* the others. The figure tells the fact the overall work load of each node is about the same during the experiment.

5. Conclusion

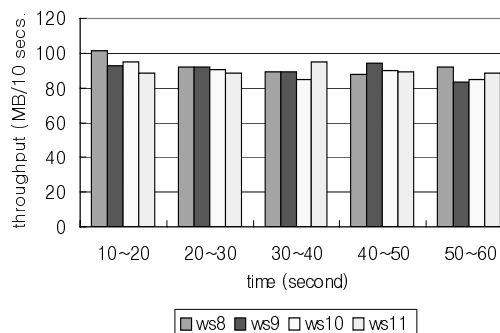
Web servers need to provide a prompt response to multiple clients, but the number of clients and their overall job requirements at a certain time are not easily predictable. Thus, they should be designed to provide services with a reasonable response time even in the case of peak connections or requests being concentrated into some contents. The hierarchical web server architecture can match these requirements in a wide range of applications. Bottlenecks in a certain part of the architecture can be avoided by increasing the number of nodes and content groups can be configured to have as many levels as they need. Even load distribution is performed in the front-end dispatchers as well as in the backend by dynamic load balancing.

References

- [1] V. Cardellini, M. Colajanni, Redirection algorithms for load sharing in distributed Web-server systems, *19th IEEE International Conference on Distributed Computing Systems*, 1999, pp 528 -535
- [2] V. Cardellini, M. Colajanni, P. S.Yu, Dynamic load balancing on web-server systems, *IEEE Internet Computing*, May/June 1999, pp. 28-39.
- [3] O.P. Damani et al., One-IP: techniques for hosting a service on a cluster machines, *J. Computer Networks and ISDN Systems*, Vol. 29, Elsevier Science, Amsterdam, Netherlands, Sept. 1997, pp. 1,019-1,027.
- [4] Fielding, et al, Hypertext transfer protocol-HTTP/1.1, <http://www.w3.org/hypertext/WWW/Protocols/>, June 1999.
- [5] T.T. Kwan, R.E. McGrath, and D.A. Reed, NCSA's World Wide Web server: design and performance, *Computer*, Vol. 28, No. 11, Nov. 1995, pp. 68-74.
- [6] Linux virtual server project, <http://www.linuxvirtualserver.org/>
- [7] V. Cardellini, M. Colajanni, Redirecion algorithms for load sharing in distributed Web-server systems, *19th IEEE International Conference on Distributed Computing Systems*, 1999, pp 528 -535



(a) Performance with the round-robin scheduling



(b) Performance with dynamic load balancing

Figure 5 Throughput of backend nodes with dynamic load balancing

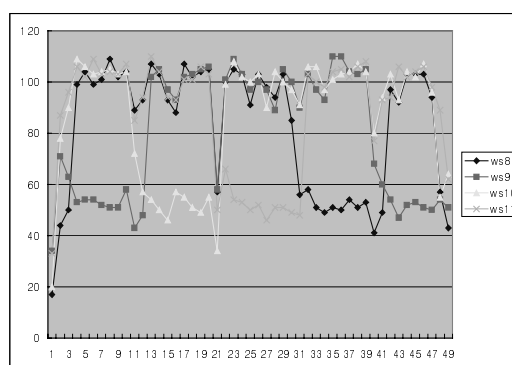


Figure 6. Number of clients' requested serviced by backend servers with a forced periodic load shift