Using Proximity Information for Load Balancing in Geographically Distributed Web Server Systems[*]

Dheeraj Sanghi, Pankaj Jalote, and Puneet Agarwal

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, UP, INDIA
{dheeraj,jalote}@iitk.ac.in, puneet.agarwal@alumni.cse.iitk.ac.in

Abstract. Many popular web sites get millions of hits everyday. To service a large number of requests, clusters of fully replicated web servers are used. In such a setup, the client's request has to be directed to a cluster and then to a server within the cluster in a manner that the client receives the response in minimum time. In this paper, we propose an adaptive policy of selecting the *nearest* cluster for a request. Proximity is assessed by the round trip delay between the cluster and the client. An innovative idea is to measure this delay only for those clients who are sending a large number of requests. We have implemented this scheme, and using a test-bed which simulates the world wide web environment, compared the performance of the scheme with that of some existing schemes. The results indicate that the proposed scheme performs better, both in terms of average response time, as well as throughput.

1 Introduction

Multiple clusters of servers placed in different geographical regions is a very common architecture for web server systems. Deploying such a system requires a proper strategy for selecting a cluster, and then selecting a server within the cluster for handling a client request. The strategy should result in low response time for the client. In many existing strategies, a server or cluster is selected without taking system state into account, e.g., random, round robin, etc. Some policies use weighted capacity algorithms to direct more percentage of requests to more capable servers or clusters but most policies do not take into account the proximity between clients and clusters.

When clients are geographically far apart, network conditions play an important role in the latency perceived by clients. In such a situation, we can provide better service to clients by taking proximity information into account. A few schemes that do try to use proximity information, either use only the IP address allocation table, which is not very reliable, or use probes between client and DNS/server, which has high overhead.

[*] This research was partially supported by a grant from Avaya Labs, NJ, USA.

In this paper, we propose a scheme for load balancing that employs proximity information, as well as information about server loads. But proximity is estimated only when some client has several requests for the website (e.g., proxy server of a large organization.) Proximity is estimated by round trip delay, which is measured by different clusters sending probes to the client. We have used a test-bed for evaluating the performance of this approach and comparing it with other approaches. The experimental results show that our proximity based approach performs better than other approaches.

2 Related Work

Cardelini et al [6] classify web server architectures based on the entity which distributes the incoming requests among the servers.

In *client-based approach*, the client side entity is responsible for selecting the server. The selection can be done by the client software (browser) or Java Applet run by the client [16] or client-side DNS [5] or proxy servers [4]. Evaluation of several approaches for server selection has been done by [15].

In *DNS-based approach*, server side authorized DNS maps the domain name to IP address of one of the servers. Several DNS based approaches are discussed in References [6] and [7]. An example of this type is the round robin approach (DNS-RR) [13]. The selection of the server can also be based on the server load or based on proximity with clients or a combination of both.

In *dispatcher-based approach*, DNS returns the address of a *dispatcher* that routes all the client request to other servers in the cluster. Thus it acts as a centralized scheduler at the server side. Dispatching of requests can be done using various techniques, like *Packet single-rewriting*, or *Packet double-rewriting* [2], or *Packet forwarding using MAC address* [12], or using *ONE-IP address* [9], or using *HTTP redirection*.

Many server side approaches for load balancing have been proposed. Guyton et al [11] focus on hop count and cost of collection of information for server selection. In Cisco's Distributed Director, server selection at DNS is done based on geographical proximity estimated using client IP address or hop count information obtained from routers. Given that clients are distributed geographically, static metrics like hop count for proximity information were not found good in study by [8]. Ammar et al [10, 17] propose local anycast resolver that is near a large number of clients, to which servers push their performance information.

3 Load Balancing using Proximity Information

We assume a two level architecture. The web server system consists of multiple clusters, each cluster having a dispatcher and multiple servers, In this system, the response time depends on the load at selected server and the path characteristics between the client and the server. the DNS selects the cluster (returns IP address of its dispatcher), and the dispatcher selects the server within the cluster.

3.1 Using Proximity Metrics for Cluster Selection

We aim to use proximity information to provide a better response time. There are various metrics to measure proximity. Some of these metrics are: round-trip time, number of hops, geographical distance, bandwidth availability, etc.

For minimizing the response time, RTT is perhaps the best metric to use along with the cluster (or server) load information. However, use of RTT requires its periodic measurement. But this overhead is within limits. When used at the client side, results in [8] indicate that the overhead was less than 1% in terms of additional network traffic. Also use of three ping messages for measurement of RTT gave better results. They found that RTT has high correlation with latency perceived by the client.

For a server side approach, measuring RTT for each client can result in a huge overhead as a heavily loaded cluster may get requests from thousands of client. Arlitt et al [3] found out that 75% of total HTTP requests to a server come from 10% of networks. So if we measure the RTT for only the high load generating clients, we can minimize the overhead.

Our approach is to dynamically distribute requests based on the current system state information. All servers in a cluster report state information to the dispatcher. Dispatcher reports aggregate cluster information to the DNS. Servers also count the number of requests coming from each IP address. This information is also sent to the dispatcher, which aggregates this information and reports IP addresses of clients having very high request rate to the DNS. Using the request rate information, DNS asks a subset of clusters to measure round-trip delay with only those clients which generate the heavy load. Round-trip delay is measured by sending ping messages to clients.

While selecting the cluster, DNS ensures that clusters have enough free capacity to serve the client requests.

4 Implementation

We have implemented the scheme on PCs running Linux.

We run *Apache* web server on all the server machines. Another process collects state information like load average, CPU and memory utilization, number of active connections, etc. The system load is obtained periodically, and sent to dispatcher. Using the web server access log, the number of requests from each IP address is determined and this information is also propagated to the dispatcher.

As discussed earlier, DNS-based schemes for load-balancing require that DNS returns the IP address of server/dispatcher, based on the state information like load, and information about the client like its round-trip time from various clusters. We have extended BIND to permit this flexibility.

Dispatcher receives regular load updates from the servers. It also receives client request rates from each server. The aggregate information is sent to the DNS. Dispatcher may also receive requests from DNS to measure RTT of select clients. It can send ICMP echo request messages to those clients and report RTTs between clients and itself.

5 Experiments

We compare the performance of the proposed scheme with some existing schemes using our test-bed [1]. The results of these experiments are presented below.

5.1 Setup

For conducting our experiments, we have setup a test-bed as shown in Fig. 1. It has three clusters on different logical networks representing three different geographical regions. Each cluster has one dispatcher (or front node, represented by 'F' node) and two servers ('S' nodes).

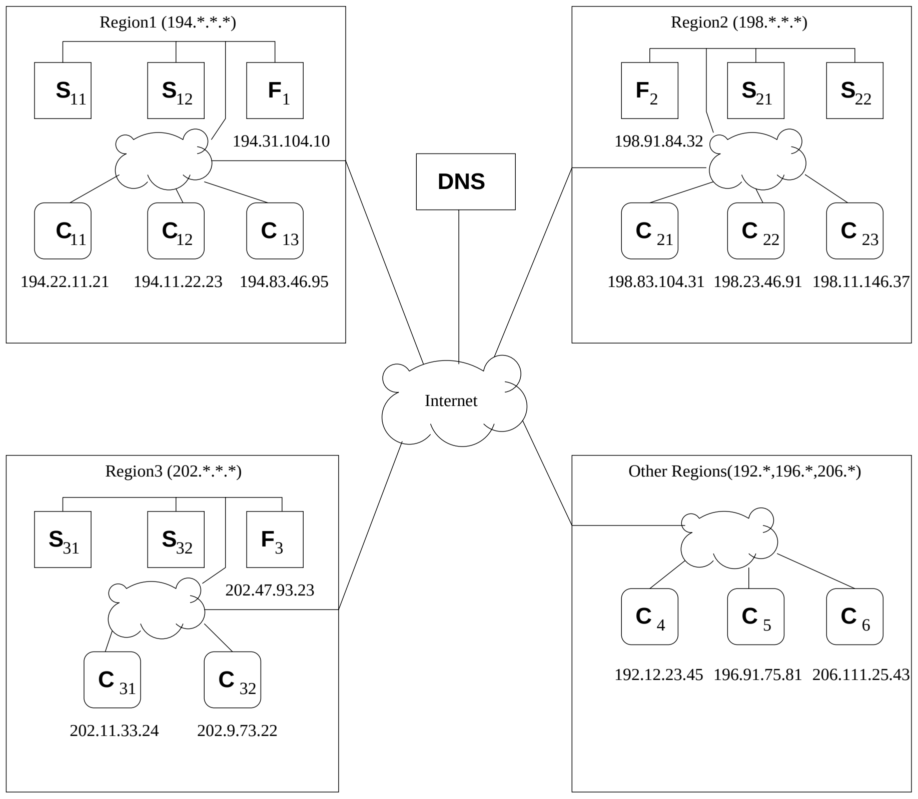

Fig. 1. Test bed used in experiments

In this setup, we have eleven clients ('C' nodes) to generate requests. Note that each client-node in the test-bed actually generates a very large number of requests in the simulation, and is equivalent to more than 10 clients in real environment. Three clients each were present in region 1 and region 2 and two clients in region 3. We also had three clients in other regions which represent mix of clients not close to any of the three clusters. A DNS was also setup to resolve IP addresses of clusters.

To model WAN effects, artificial delays and packet losses were introduced using Nistnet software [14]. We have introduced smaller delays for IP packets sent and received between clients and servers in the same region and higher delays for packets between clients and servers in different regions. We have assumed less packet loss (2%) while they travel between the same region. For packets traveling across the regions, the packet loss is assumed to be higher (5%).

To generate load and measure performance in experiments, we have used *WebStone*. This is a standard software used to benchmark web servers. Different schemes were tested with everything kept identical except policy for cluster selection at DNS.

5.2 Experimental Results

Since minimizing the response time was the primary objective of the scheme, we use average response time as the metric for performance evaluation. We have compared the proximity-based policy with three other policies for load balancing, viz., round robin (RR), random (R) and Weighted capacity (WC).

We conducted experiments to measure performance under two different conditions for RTT on links. In one condition, there was less variations in RTT (delays were mostly in the range of 20 to 45 milliseconds with a mean of 30 milliseconds for links within the same region and in the range of 200 to 500 milliseconds with a mean of 300 milliseconds). In the other condition, there were high variations in RTT (delays were mostly in the range of 10 to 70 milliseconds, with a mean of 30 milliseconds for links in the same region, and in the range of 100 to 600 milliseconds with a mean of 300 milliseconds). We have performed experiments with file sizes of 5KB and 50KB.

We varied the number of client processes from 20 to 100 in steps of 20. We also measured the request rate with the number of client processes. The request arrival rate increases linearly with the number of client processes.

Average response time of the different schemes for file sizes 5KB and 50KB are shown in Figures 2 and 3, when there were high variations in round-trip time on links. As seen in the figure, the RR, R, and WC schemes have almost similar performance, while the proximity-based scheme has a significantly lesser response time.

For 5KB files, while the average response time with other three policies was between 0.98 to 1.1 seconds, the average response time with our policy was between 0.49 to .51 seconds. For 50KB files, the average response time with other three policies was between 1.96 to 2.75 seconds, and the average response time with our policy was between 1.02 to 1.13 seconds. The average response time was reduced to half when our policy was used as compared to other policies.

The average response times of different schemes when there was low variations in round-trip time on links are shown in Figures 4 and 5. In this case also, our scheme performed better than the other schemes. For 5KB files, the average

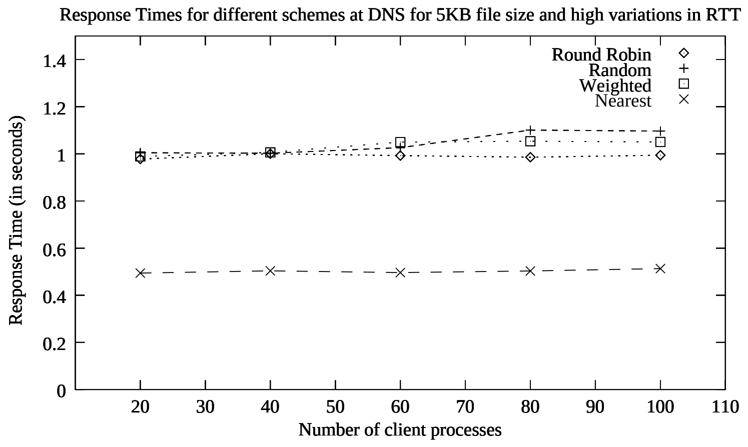

Fig. 2. Average response time with 5KB file size and high variations in RTT

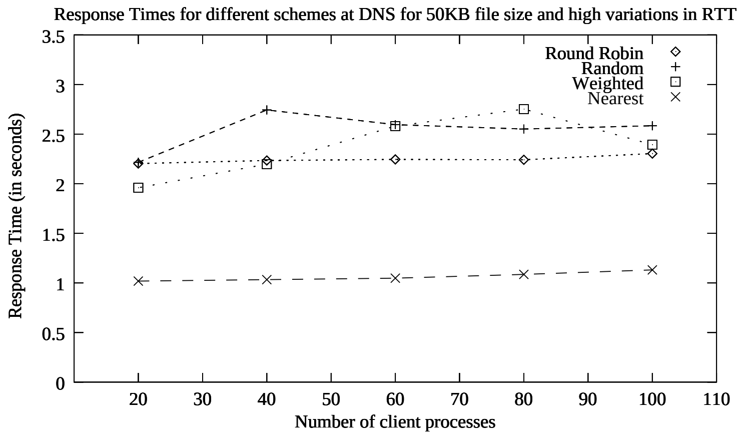

Fig. 3. Average response time with 50KB file size and high variations in RTT

response time with first three policies was between 0.86 to .99 second, the average response time with our policy was between 0.45 to .48 second. For 50KB files, the average response time with other three policies was between 1.7 to 2.27 seconds, and the average response time with our policy was between 0.8 to 0.93 seconds. Thus we see that the response time is reduced by a large factor when our policy is used and clients are distributed geographically across the globe.

We have also measured throughput (in Mbps) in all simulation experiments. We find that the throughput achieved by our policy is approximately twice in most cases.

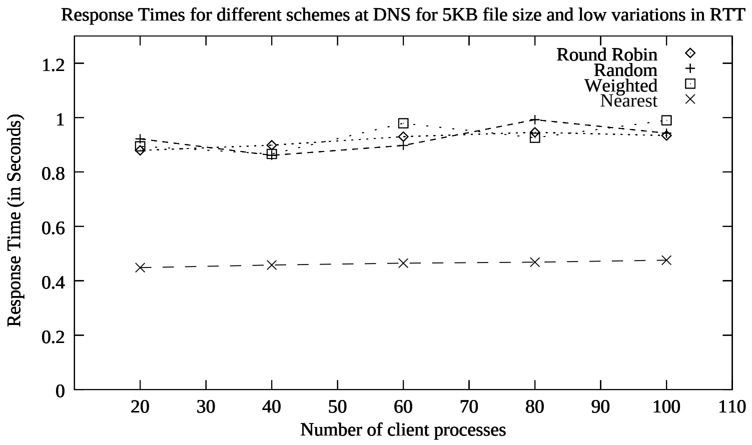

Fig. 4. Average response time with 5KB file size and low variations in RTT

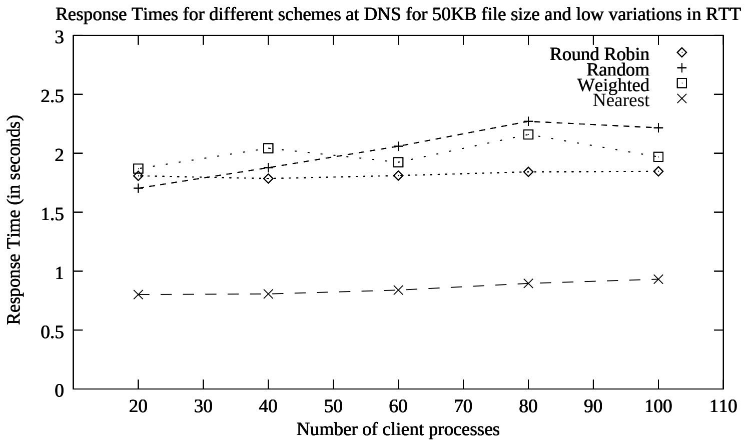

Fig. 5. Average response time with 50KB file size and low variations in RTT

6 Conclusions

We have proposed a proximity-based scheme for request distribution for a very large website. The scheme also takes server load into account. In the proposed scheme, the round trip time (RTT) for the client is measured, and the request is sent to the cluster that has the least RTT. This is done only for clients generating a lot of requests.

We have implemented the scheme on Linux machines. Using a test-bed, which simulates the delays of a real network, we compared the performance of our proposed policy with round robin, random, and weighted capacity. We found that the proposed policy gives better performance in terms of response time and throughput as compared to other policies.

References

[1] P. Agarwal. A test-bed for performance evaluation of load balancing strategies for web server systems. Master's thesis, CSE Dept., IIT Kanpur, India, May 2001.

[2] E. Anderson, D. Patterson, and E. Brewer. The Magicrouter: An application of fast packet interposing. http://www.cs.berkeley.edu/~eanders/projects/-magicrouter/osdi96-mr-submission.ps, May 1996.

[3] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Trans. on Networking*, 5(5):631–644, Oct. 1997.

[4] M. Baentsh, L. Baum, and G. Molter. Enhancing the web's infrastructure: From caching to replication. *IEEE Internet Computing*, 1(2):18–27, March-April 1997.

[5] M. Beck and T. Moore. The Internet-2 distributed storage infrastructure project: An architecture for Internet content channels. In *Proc. of Third Int'l WWW Caching Workshop*, Manchester, UK, June 1998.

[6] V. Cardelini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web server systems. *IEEE Internet Computing*, 3(3):28–39, May-June 1999.

[7] M. Colajanni, P. S. Yu, and V. Cardelini. Dynamic load balancing on geographically distributed heterogenous web servers. In *IEEE 18th Int'l Conference on Distributed Computing Systems*, pages 295–302, May 1998.

[8] M. E. Crovella and R. L. Carter. Dynamic server selection in the internet. In *Proc. of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS '95)*, June 1995.

[9] O. P. Damani, P. Y. Chung, and C. Kintala. ONE-IP: Techniques for hosting a service on a cluster of machines. In *Proc. of 41st IEEE Computing Society Int'l Conference*, pages 85–92, Feb. 1996.

[10] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proc. of IEEE INFOCOMM '98 Conf.*, 1998.

[11] J. Guyton and M. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of ACM SIGCOMM '95 Conference*, pages 288–298, Oct. 1995.

[12] G. D. H. Hunt, G. S. Goldzsmit, R. P. King, and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *Proc. of 7th Int'l World Wide Web Conference*, Apr. 1998.

[13] T. T. Kwan, R. E. McGrath, and D. A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, pages 68–74, Nov. 1995.

[14] NIST. NistNet network emulator. http://snad.ncsl.nist.gov/itg/nistnet/.

[15] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. In *Proc. of the Workshop on Internet Server Performance*, 1998. http://www.cs.wisc.edu/˜cao/WISP98/final-versions/mehmet.ps.

[16] C. Yoshilakawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *Proc. of Usenix Ann. Tech. Conf.*, Anaheim, CA, Jan. 1997.

[17] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated service. *IEEE/ACM Transactions on Networking*, 8(4):455–466, Aug. 2000.