A TESTBED FOR PERFORMANCE EVALUATION OF LOAD BALANCING SCHEMES FOR WEB SERVER SYSTEMS[*]

Dheeraj Sanghi[†], Pankaj Jalote, Puneet Agarwal
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur, INDIA
{*dheeraj,jalote*}*@iitk.ac.in*

ABSTRACT

To improve the response time of a web server, an obvious approach is to have multiple servers. Effectiveness of a replicated web server system will depend on how the incoming requests are distributed among these replicas. A large number of load balancing schemes for web server systems have been proposed in the literature. In this paper we describe a test-bed that can be used to evaluate the performance of different load balancing schemes. The test-bed uses a general architecture that allows different load balancing schemes to be supported easily. It emulates a typical world wide web scenario and allows variable load generation and performance measurement. We have performed experiments to compare the performance of a few schemes for load balancing using this test-bed.

KEYWORDS: Replicated web servers, Load balancing, Test-bed, Performance Comparison.

1. INTRODUCTION

The number of users accessing the Internet is increasing rapidly and it is not uncommon for popular sites to get hundred million hits a day. Replication of web servers is one way to deal with a large number of requests. However, to improve performance, effective load balancing schemes are needed. Many of the existing load balancing schemes assume that the replicated site has all the servers in one cluster. But beyond a certain amount of traffic, the connectivity to this cluster can become a bottleneck. Consequently, for serving heavier loads, multiple clusters are needed, which may be geographically distributed. This changes the load balancing problem to that of first selecting a cluster and then selecting a server within that cluster.

With a large number of proposed schemes for load balancing, there is a need to evaluate their performance under different conditions. Analytical models that will account for the real-life Internet scenario are very hard to build. Simulation is also hard as behavior of caches at various levels, of servers, DNS, network, etc., is not well understood and can not be easily simulated.

One approach is to build a test-bed in which any load balancing scheme can be implemented and its performance measured under different conditions. Besides evaluating performance, a test-bed will also thoroughly test the implementation of the scheme.

In this paper we describe a test-bed that we have developed for evaluating performance of load balancing schemes. The test-bed uses a general architecture which allows easy implementation

[*]This work is supported in part by a grant from Avaya Labs Research, New Jersey, USA.
[†]Author for correspondence.

and performance analysis of any load balancing scheme. Using the test-bed, we have performed experiments to evaluate the performance of some load balancing schemes and compared them. This is done to validate the test-bed. The experimental results are as expected, and show that for a geographically distributed web server system, schemes like random, round-robin, and weighted have similar performance, though a proximity-based scheme performs better.

2. LOAD BALANCING SCHEMES

Serving client requests using a cluster of replicated servers requires mapping a request to one of the servers. Cardelini *et. al.* [6] classify web server architectures based on the entity which distributes the incoming requests among the servers. Some schemes in each category may use feedback based algorithms, while others use non-feedback algorithms, as discussed in [2]. Some categories of load distribution schemes are briefly discussed here.

In **Client-based schemes**, the client side entity is responsible for selecting the server. The selection can be done by the client software (browser) [11] or Java Applet run by client [15] or client-side DNS [5] or proxy servers [4]. These entities are aware of existence of multiple servers and they select the server themselves either randomly or based on probe results or based on previous access performance reports.

In **DNS-based schemes**, the server-side DNS maps the domain name to IP address of one of the servers. Several DNS based schemes are discussed in References [6] and [7]. The *constant TTL algorithms* are those that assign a fixed time to live (TTL) value in all cases. An example of this type is the round-robin scheme (DNS-RR) [10], which was one of the first load balancing schemes. The selection of the server can also be based on the server load or based on proximity with clients or a combination of both.

The *dynamic TTL algorithms* are those that assign different TTL values to different requests. These are of two types. In *variable TTL algorithms*, as server load increases, TTL value is decreased in an attempt to increase the DNS control over request distribution. In *adaptive TTL algorithms*, a large TTL value can be assigned for a more capable server and less TTL value for replies to clients that have high request rate.

Dispatcher-based schemes give full control over client requests to a server side entity, called *dispatcher*. In these schemes, DNS returns the address of a dispatcher that routes all the client requests to servers in a cluster. Thus the dispatcher acts as a centralized scheduler at the server side that controls all the client request distribution, and presents a single IP address to the outside world for the entire cluster. Dispatching of requests can be done using various techniques, like *Packet single-rewriting*, or *Packet double-rewriting* [3], or *Packet forwarding using MAC address* [9], or using *ONE-IP address* [8], or using *HTTP redirection*.

3. TEST-BED DESIGN

To compare various schemes for request distribution at the server side, we have designed and implemented a test-bed which emulates real network scenarios and follows all steps in HTTP request service. All standard components used in the Internet are also used in this test-bed, for example, *BIND* for DNS and *Apache* web server. We have used *WebStone* for generating HTTP requests. We have modeled WAN delays and bursty packet losses which are common on Internet links.

3..1 Design Goals

The test-bed was designed to facilitate easy measurement of various parameters of web server performance like average response time for requests and the throughput of Web Server system. While setting up the test-bed following goals were kept in mind:

1. It should *emulate real Internet scenario* in the lab environment.
2. It should be *general* enough so that different schemes for load balancing can be implemented.
3. It should permit an *easy configuration* of load balancing schemes.
4. It should allow *dynamic schemes* that make their decision based on system state information.
5. It should be *scalable* and allow a large number of servers and clients.

3..2 Architecture of the Web Server System

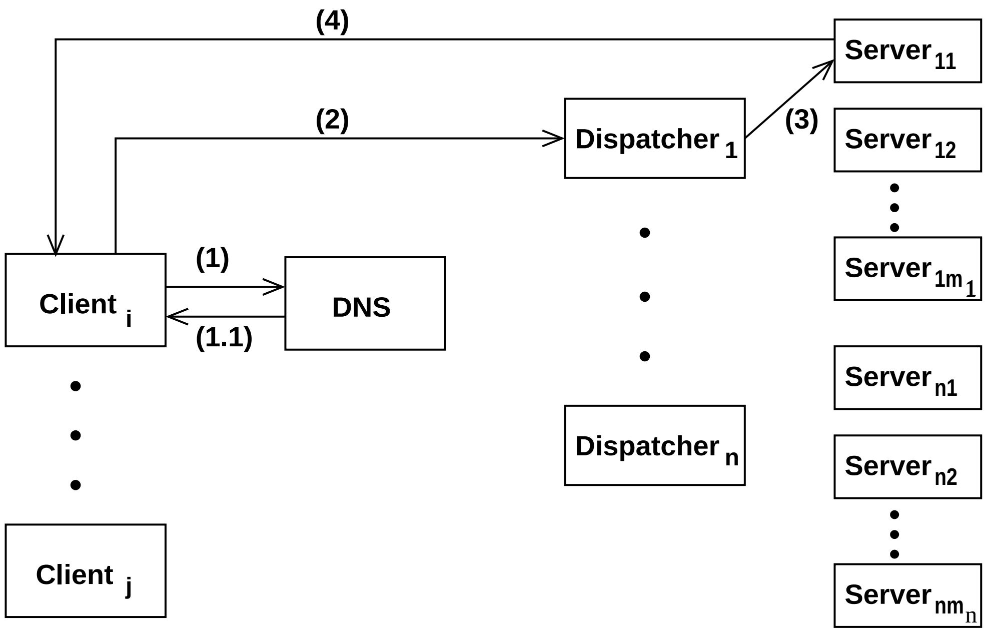

Figure 1: System model

To permit implementation of various schemes, we have chosen a general architecture shown in Figure 1 for the web server system. In this architecture, the replicas are organized as a set of clusters, each cluster having a dispatcher and some servers. A request is directed to a server in two steps. In the first step, the request is directed to one of the clusters by the DNS by returning the IP address of the dispatcher of that cluster. In the second step, the client sends the request to the dispatcher of that cluster, which decides the server that should serve this request, and forwards the request to that server.

This architecture allows the implementation of both DNS-based and dispatcher-based schemes as well as combinations of the two. It permits modeling the situation where servers are located in different geographical locations to provide better service to customers in different regions. It is possible to model a scenario where there is only one cluster, or a scenario where there is no dispatcher. It can also model single server systems easily.

3..3 Support for Load Balancing Schemes

We permit different schemes for request distribution at two places - DNS and the dispatchers. At DNS, the cluster IP address can be selected according to the desired scheme. We have already implemented four schemes: random, round-robin, weighted and proximity-based cluster selection [1]. In the software, new schemes can be implemented very easily. At the dispatcher, each new TCP connection from a client can be scheduled on the desired server. We have implemented three schemes for server selection at the dispatcher: random, round-robin, and weighted round-robin.

4. IMPLEMENTATION

We are using PCs running Linux for all components in the test-bed. The request generation and collecting of statistics is done by *WebStone*. DNS Server has been enhanced to return one of the many possible IP addresses based on the load balancing scheme. Finally, we have implemented dispatcher software to distribute requests transparently within a cluster, based on the given scheme. All the components have been implemented in a way that new schemes can be implemented as modules which can be added to the existing test-bed easily.

In this section, we describe the implementation of all these components.

4..1 Client-side Software

WebStone is installed on one host, where the master process will be executed. The software also contains a program called *webclient*, which is copied on all those hosts, which are going to generate web requests during the experiments.

WebStone configuration file, *filelist*, includes the names of the remote machines, and the number of *webclient* processes that should be executed on those remote machines. Note that *WebStone* does not allow direct control over the number of requests generated per second. One can only specify the number of processes. But more number of processes imply more number of requests. Further *WebStone* does not generate differential load on different machines. But this can be implemented by mentioning the name of the client machine in the configuration multiple times. One can also configure the distribution of file size that is being downloaded through an HTTP request.

Clients also simulate delays and losses to mimic the Internet behavior. The configuration file has parameters to control the amount of delay and losses, and their respective distribution.

4..2 DNS Software

As discussed earlier, DNS-based schemes for load-balancing require that DNS returns the IP address of server/cluster, based on the state information like load, and even information about the client like its round-trip time from the various clusters. Current implementation of the domain name server (BIND-9.1) does not provide such support. It only supports random and round-robin selections of IP address. We have extended BIND to permit this flexibility.

4..3 Server Software

We run *Apache* web server on all the server machines. But one could use any other server software without necessitating any change in the test-bed. In addition to the web server, we execute another process that collects state information like load averages, CPU and memory utilization, number of active connections, number of server processes running to handle client requests, etc.

The system load is obtained periodically (every 500 msec) and if the load has changed since the last update considerably, the change is propagated to the dispatcher. A update is sent after every T time units (T = 4 seconds) anyway.

Using the web server access log, the number of requests from each client (IP address of client) is determined and this information is propagated to the dispatcher periodically.

4..4 Dispatcher Software

Dispatcher receives regular load updates and asynchronous alarm messages from the servers. It also receives client request rate in the last T seconds from each server. The aggregate load information is sent to the DNS. Dispatcher also sends the request rate information of clients that have a high request rate to 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.

Dispatcher is responsible for distributing requests within a cluster. Depending on the scheme, it can take into account loads on various servers and previous request rate of the clients, to choose a particular server. It also maintains a table of client's IP address and port number so that successive requests from one client can be sent to the same server. Dispatcher also introduces delays and losses based on parameters in the configuration file.

5. EXPERIMENTS

We have conducted preliminary experiments to validate our test-bed. We have implemented several schemes at the DNS, as well as at the dispatcher. We have conducted experiments with varying amount of load. The results are on expected lines. This validates the test-bed as a good research tool to compare different schemes under varying Internet conditions.

5..1 Setup

For conducting our experiments, we have setup a test-bed as shown in Figure 2. 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).

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 [12]. Half of delay and losses occurred in one direction and the other half in the reverse direction. 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. In our experiments, round-trip delay in the range of 10-50 ms was introduced for the same region and delay in the range of 50-250 ms was introduced across the 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%).

5..2 Experimental Results

The master process of *WebStone* software was executed on one of client PCs. During the experiments, we varied the number of client processes from 20 to 120 in steps of 10. *WebStone* tries to execute roughly equal number of processes at each client. Different DNS-based schemes were tested with all network and load parameters kept identical. We have evaluated the performance of four schemes - round-robin, random, weighted capacity and proximity based cluster selection.

In proximity-based scheme [1], dispatchers inform DNS of clients who are sending a large number of requests. DNS requests all dispatchers to send *ping* messages to the client, and estimate the round-trip time. All dispatchers inform DNS of the RTT values. This is then used for selecting the

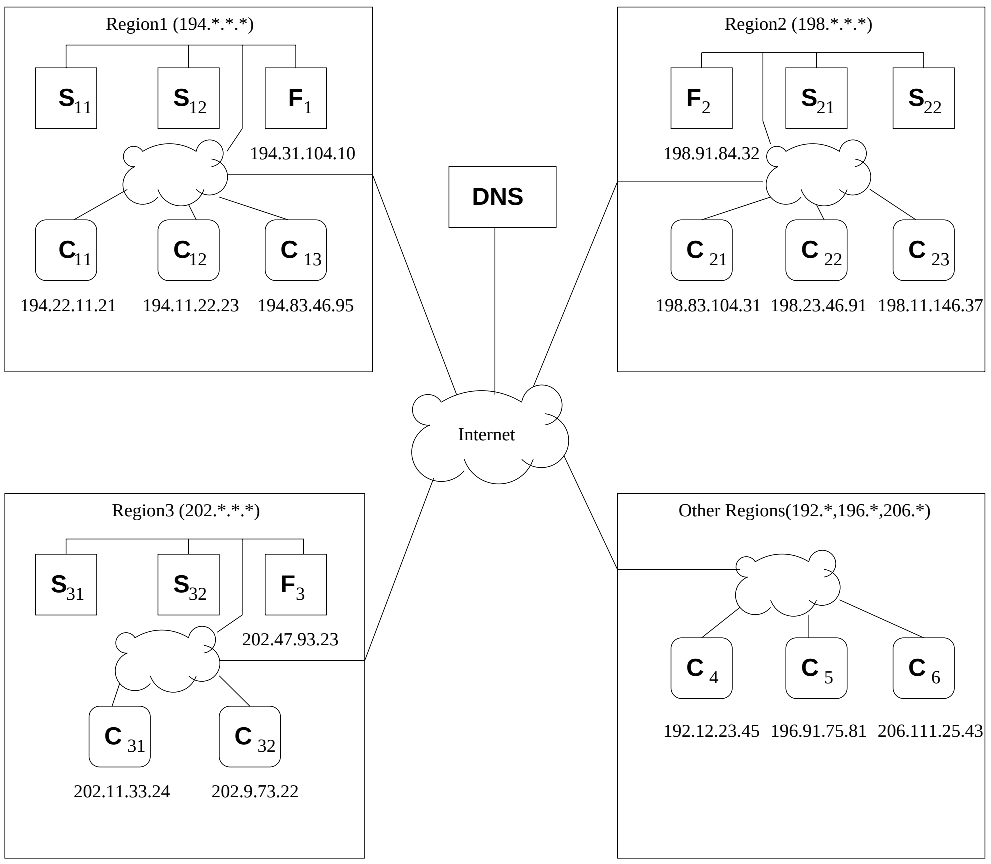

Figure 2: Test-bed used in experiments

nearest cluster.

Average response time of different schemes is shown in Figure 3. For random, round-robin, and weighted schemes, we see that there is not much variation in average response times, since servers were not a bottleneck in these experiments. Since the proximity-based scheme saves on round-trip time, it gives a lower response time on the average. The average throughput achieved with different DNS-based schemes is shown in Figure 4. We once again notice that throughput is much higher for the proximity-based scheme.

6. CONCLUSIONS

Many popular websites receive millions of hits everyday. To provide better response time to all clients, replication of web content on multiple servers is used. A large number of schemes for load balancing have been proposed. We have designed and implemented a test-bed, to evaluate the performance of load balancing schemes. This test-bed is quite flexible and schemes for load balancing (using server side components) can be compared with each other very easily. This test-bed will help in understanding trade-offs and impact of different parameters on a distributed and replicated web server system.

We have implemented support for four DNS-based schemes in our test-bed. These are: random, round-robin, weighted, and proximity-based. We compared these schemes using our test-bed, and

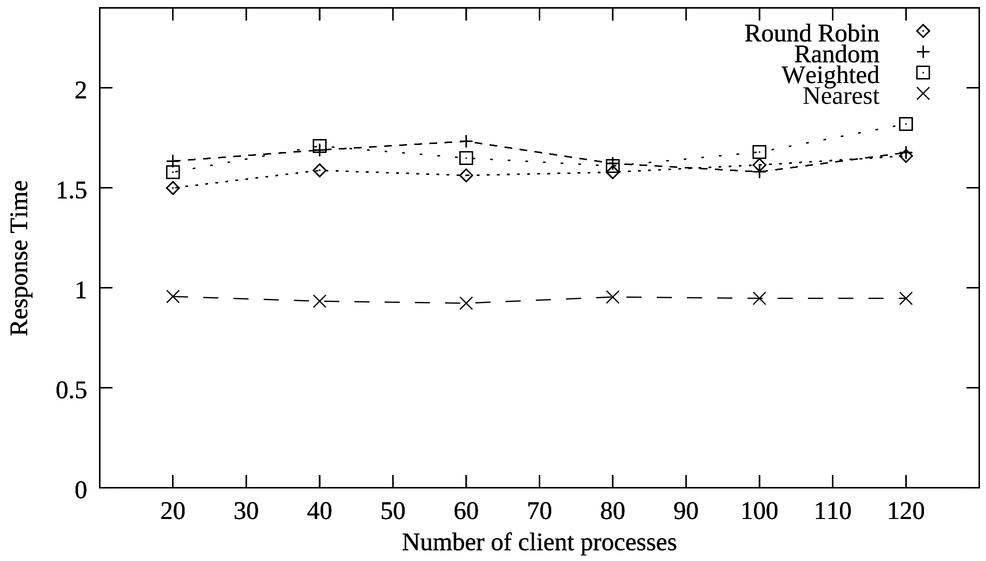

Figure 3: Average response time with different DNS-based schemes

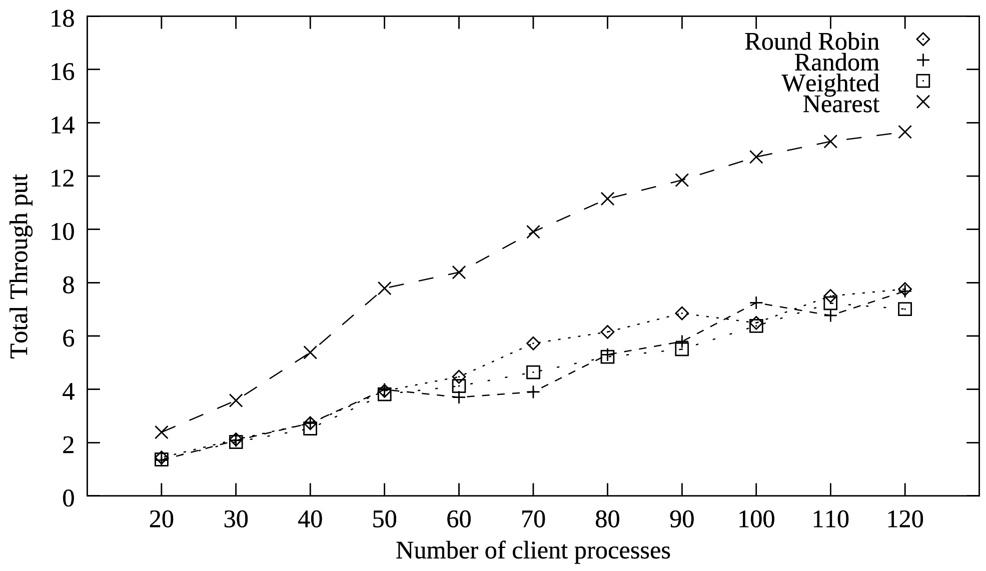

Figure 4: Throughput achieved with different DNS-based schemes

the results are on the expected line. The proximity-based scheme results in least response time.

We plan to add support for more schemes in this test-bed. We also plan to upgrade the hardware, and conduct more experiments with this test-bed. We are currently working on a web-interface, where one could submit one's request for running simulation with given parameters.

References

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

[2] A. Aggarwal and M. Rabinovich. Performance of dynamic replication schemes for the Internet hosting service. Technical report, AT&T Labs., Oct. 1998. http://www.research.att.com/∼misha/radar/tm-perf.ps.gz.

[3] 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.

[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. http://wwwcache.ja.net/events/workshop/18/mbeck2.html.

[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 *Proc. of IEEE 18th Int'l Conf. on Distributed Computing Systems*, pages 295–302, May 1998.

[8] 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.

[9] 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.

[10] T. T. Kwan, R. E. McGrath, and D. A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, Nov. 1995.

[11] D. Mosedale, W. Foss, and R. McCool. Lessons learned administering Netscape's site. *Internet Computing*, 1(2):28–35, March-April 1997.

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

[13] T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable web-server clustering technologies. *IEEE Network*, May-Jun 2000.

[14] R. Vingralek, M. Sayal, Y. Breitbart, and P. Scheuermann. Web++ architecture, design and performance. *World Wide Web*, 3(2):65–77, Apr. 2000.

[15] 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.