Challenges in URL Switching for Implementing Globally Distributed Web Sites*

Zornitza Genova and Kenneth J. Christensen
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
Email: {*zgenova, christen*}*@csee.usf.edu*

Abstract

URL, or layer-5, switches can be used to implement locally and globally distributed web sites. URL switches must be able to exploit knowledge of server load and content (e.g., of reverse caches). Implementing globally distributed web sites offers difficulties not present in local server clusters due to bandwidth and delay constraints in the Internet. With delayed load information, server selection methods based on choosing the least-loaded server will result in oscillations in network and server load. In this paper, methods that make effective use of delayed load information are described and evaluated. The new Pick-KX method is developed and shown to be better than existing methods. Load information is adjusted with probabilistic information using Bloom filter summaries of site content. A combined load and content metric is suggested for use for selecting the best server in a globally distributed site.

1. Introduction

The use of globally distributed Web servers for mirroring of popular content is commonly done. For example, apache.org maintains over 180 mirror sites around the world [1]. Such globally distributed server systems can reduce user response times and decrease Internet traffic load. URL, or layer-5, switches can be used to automatically switch or redirect requests to individual servers or server sites. URL switching makes possible a for single site (i.e., a single URL) to consist of locally or globally distributed servers. URL switches should send requests to the "best" server. The best server is the one that can give a satisfactory response time to the user and minimize network traffic. Selection criteria include server load, server contents, and network path characteristics.

In local server clusters, selecting the least loaded server works well because load information can be kept current. In geographically distributed server clusters, maintaining current load information is difficult given the constraints of the Internet connectivity between individual servers or sites. Given delayed load information, selection of the least loaded server can result in pathologically poor performance. This poor performance is due to the so-called "herd effect" (which has been studied in [8] and [12]). In the herd effect, requests rush to the currently advertised least-loaded server and cause unbalanced and oscillating loads. Dealing effectively with delayed load information is an open challenge.

Caching can reduce both Internet traffic and server load. Caching can be implemented local to the client (e.g., in the browser application or via proxy servers in the client LAN), within a caching hierarchy in the Internet (such as in the case of geographical push-caching [11]), or at the server site. A reverse cache is part of a local server site and handles commonly requested objects without having to burden the server hosts with additional connections. Caching hierarchies use compressed cache directory information, called digests, to coordinate cooperation between caches (e.g., SQUID [19]). Caching is largely limited to static content and the use of digests may present performance scaling problems. Having knowledge of server site cache contents is needed to make good selection decisions.

The remainder of this paper is organized as follows. Section 2 describes methods for building distributed content. Section 3 develops the new **Pick-KX** method for using stale load information. Section 4 develops a distributed Web server simulation model and presents experimental results for existing and new (**Pick-KX**) server selection methods. Section 5 describes adding content-based knowledge to server selection. Finally, Section 6 is a summary and describes future work.

* This work funded by an NSF CAREER grant (award ANI-9875177).

0-7695-0771-9/00 $10.00 © 2000 IEEE

2. Building Distributed Content

Web content is "globalized" via distributed server and caching infrastructures. Figure 1 shows a globally distributed web site comprised of multiple local sites. The local sites partially or fully mirror their contents. Each local site contains one or more servers (or possibly different capabilities) and a reverse cache. At any given time the load and reverse cache contents vary between the sites. Each local site is front-ended by a URL switch. The URL switches have knowledge of state information such as site load and contents and use this information to make switching decisions. In the figure, a request to local site #1 is shown as redirected to local site #2. The URL switches share state (e.g., site load and contents) information.

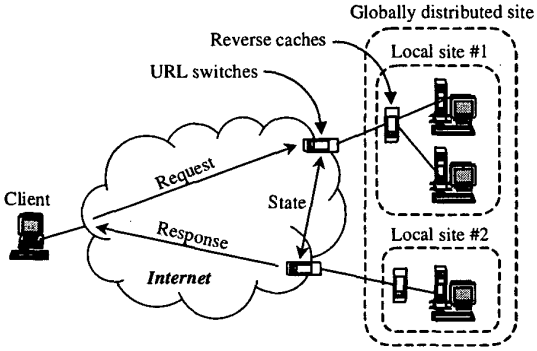

Figure 1 - Globally distributed site using URL switches

2.1 Implementing local server clusters

In a local server cluster, a high-speed switched LAN connects the servers and caches that form the local site. Local server clusters can be implemented with a front-end URL switch. The simplest and most common policy for server selection is for the URL switch to direct a request to the least loaded server in the local site. The high-speed LAN enables state information to be frequently shared and least-loaded server selection to be effective. If the URL servers "spoofs" for the multiple servers of the local site, Network Address Translation (NAT) methods can be used to switch requests to servers. TCP connection handoff and splicing are other methods.

Pai et al. investigate content-based server selection in the context of a local server cluster [13]. A URL switch distributes requests to servers in a local cluster based on a Locality Aware Request Distribution (LARD) technique which combines load balancing and content knowledge to achieve both load balancing and increased cache hits at back-end servers. The data structures used for identifying servers with cached objects are unclear. A load factor of 2x is used to determine when to favor an "uncached" server over a server with the requested object in cache.

Local clusters can be implemented without front-end URL switches. With Distributed Packet Rewriting (DPR) [3], Bestavros et al. let individual servers in a cluster route connections to less loaded servers. Thus, DPR implements routing functions within a server. The most effective metric for server load was found to be number of TCP connections [3].

2.2 Implementing distributed server clusters

Scaling URL switching from a local cluster to a distributed cluster adds several new complexities:

1. Internet connections between local sites have less bandwidth and greater delay than a local LAN. Thus, state information must be shared with less frequency and be smaller in size.
2. Network path characteristics between a client and a server site must be considered if a request is switched to a distant site.
3. Switching techniques that work in a local site would result in a "dog leg" routing between two distant sites (e.g., DPR and NAT). Thus, redirection techniques need to be explored.

In this paper we focus on mitigating the effects of delay and bandwidth constraints of Internet connections between distributed sites. Caching of load information is necessarily based on an update interval that can be seconds to minutes in length. Content information can best be shared as compressed digests (e.g., as the caching research community is exploring in SQUID [19]).

The following methods for server selection based on load information have been previously studied:

- **Pick-1** - Uniform random selection of a server (random selection - oblivious to load or delay).
- **Pick-N** - Selection of the least loaded server from the full set of N servers.
- **Pick-2** - Selection of the least loaded server from two randomly chosen servers.
- **LI** - Load Interpretation (LI) algorithms for interpreting old load information from [8]. Two variants of **LI** are described below.

The general **Pick-K** (K = 1, ..., N) methods are analyzed in [12] using a fluid-limit approach to study the limiting system. The behavior of the system is then studied via a set of differential equations and via simulation methods. It is shown that systems that attempt to use global

information too aggressively will suffer in performance and that using limited information performs very well.

In [8] two **LI** algorithms are developed as an improvement to **Pick-2**. For the **LI** algorithms, define:

L_i .....Reported load information (typically this is the queue length) for server i, $i = 1, \ldots, N$

T_{post} .....Posting interval for load information L_i

λ .....Mean per-server request arrival rate

P_i .....Probability of an arriving request selecting server i

The **Basic LI** algorithm attempts to balance the load across all servers by the end of a posting period. The **Aggressive LI** algorithm subdivides the posting period such that load is balanced during the initial part of the period. Assume that servers have been sorted by L_i with server i the ith least loaded ($i = 0, \ldots, N-1$ and $L_N = \infty$ as a sentinel). The posting period is then subdivided into N intervals. In interval j ($j = 0, \ldots, N-1$) requests are evenly distributed across machines $0, \ldots, j$ to bring their loads up to L_{j+1}. A subinterval j is of length,

$$T_j = \frac{j \cdot (L_{j+1} - L_j)}{\lambda \cdot N}, \quad (1)$$

and,

$$P_i = \begin{cases} \frac{1}{j+1} & \text{if } i \le j \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

In [8] it is shown that the **Aggressive LI** algorithm results in lower response times than the **Basic LI** algorithm for periodic posting of load information.

3. The New Pick-KX Selection Method

It is possible that adding further, limited randomization to an existing server selection method can improve performance. To explore this direction, we define a new variant of **Pick-K,** called **Pick-KX**, where the probability, P_j, that an incoming request selects server j ($j = 1, \ldots, K$ and $1 \le K \le N$) for N servers is:

$$P_j = \frac{X_j}{\sum_{i=1}^{K} X_i} \quad (3)$$

for

$$X_j = \frac{(L_{total} - L_j)}{L_{total}} \quad (4)$$

where L_{total} is the sum of the queue lengths of the uniformly randomly chosen K servers. The **Pick-KX** method weights the K randomly chosen servers by load. For example, given two servers (i.e., for **Pick-2X**) with loads of 10 and 5, the probabilities of selection would be 1/3 and 2/3, respectively. With existing **Pick-2** the server with the least load would always be selected. Unlike **Aggressive LI** or **Basic LI**, **Pick-KX** does not use time information in its selection (i.e., the load weightings are not a function of the update interval).

4. Simulation Evaluation of Server Selection

Effective server selection is critical during periods of high user demand for documents stored on replicated servers. Such high load situations would be difficult to study in the Internet using experimental methods. Even with simplifying assumptions, simulation techniques are likely the best way to gain insights into the general system behavior of server selection in high load situations since they can allow controlled and replicated generation of such situations.

A simulation model with the following attributes was built to study system behavior:

- There exist N replicated and distributed servers, each of equivalent capacity.
- Each server can support a maximum of M simultaneous connections. If the limit of M connections is reached, an incoming request is discarded and can retry as part of the overall stream of requests.
- A very large population of clients independently generates requests to these servers.
- Every T_{post} seconds, the server load (the number of active connections) for each replicated server is posted in an array of size N elements. This global array models periodic updating of load information at a URL switch.
- The limiting shared resource (the performance bottleneck) is the outgoing link from each server to the Internet. Network effects (e.g., latency as a function of path length) are not modeled.

The simulation model was developed as a queueing system model using the CSIM18 simulation engine [18] and is available from the authors. A validation of the simulation model was performed using shared data from [8]. The validation results can be seen at [5].

The overall stream of client requests is modeled as Poisson with a rate λ. A Poisson assumption is probably reasonable for user behavior [14], but does not model accurately the machine requests that spawn from each HTML page requested (e.g., for imbedded images). Future work will use both burst arrivals and trace-driven workloads. Due to the effects of the server selection methods, the arrival process at the individual servers will generally not be Poisson (except for uniform random server selection). Of special interest is the modeling of

service time. A simple exponential service time (for validation of the simulation model with [8]) and a heavy-tailed service time distribution based on measured sizes of Web objects are modeled. For actual Web servers it is recognized that Web object sizes are distributed with a heavy tail [17]. This would suggest that service times will also be heavy-tailed. A heavy-tailed distribution of Web objects was used with a mean file size of 21-Kbytes (as measured in [2]) and with Bounded Pareto, $BP(k, p, \alpha)$ distribution as used in [7]. In a $BP(k, p, \alpha)$, k is the smallest value taken $(k \le x \le p)$, p the largest, and α the shape parameter. Parameters were chosen as k = 1 Kbytes and p = 10 Mbytes as reasonable minimum and maximum file sizes. The resulting $\alpha = 0.8037$ is within the range of α values measured in [2] and [6]. Figure 2 shows a histogram of ten million generated file sizes. This histogram is similar in shape to the measured Web server file size histogram in [17].

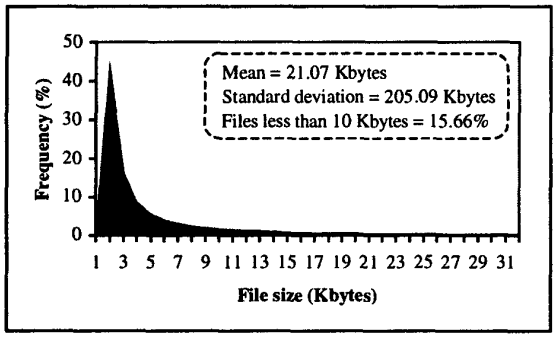

Figure 2 - Histogram of ten million generated file sizes

A model of a system with 10 servers each bottlenecked by a 1.544-Mbps (T1) communications link was built. Each server had a limit of maximum of 80 simultaneous connections (e.g., setting `MaxClients` = 80 in Apache server) with rejected connections being recycled back into the original stream of incoming connections. Service time is $BP(1, 10, 0.8037)$ distributed and request arrivals are Poisson.

Figure 3 shows the response time for a 90% offered load. It can be seen that **Pick-2X** is better than **Pick-2** and **Pick-NX** is significantly better than **Aggressive LI** for large periodic posting intervals. **Pick-NX** is less complex than **Aggressive LI** (e.g., it does not require sorting of server loads or precise timing of request intervals). It is somewhat surprising that a method that uses less knowledge of the system performs better - this further shows the need for more analysis. The lower complexity of **Pick-NX** may make it more tractable for analysis (than **Aggressive LI**).

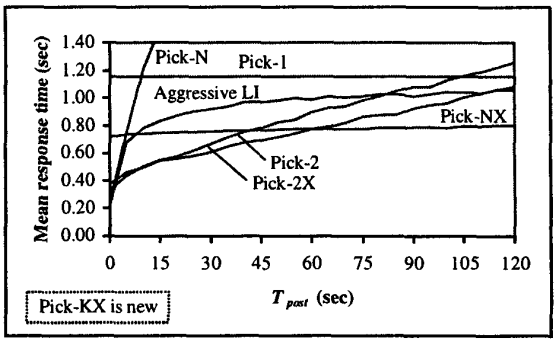

Figure 3 - Response time for heavy-tailed service times

Figure 4 shows the number of connections for an arbitrary server in the modeled 10-server system over a period of 15 minutes. Note that **Pick-N** in Figure 4(a) is unstable with oscillation between zero connections and the maximum allowed (80 connections). **Pick-NX**, in Figure 4(b), is stable.

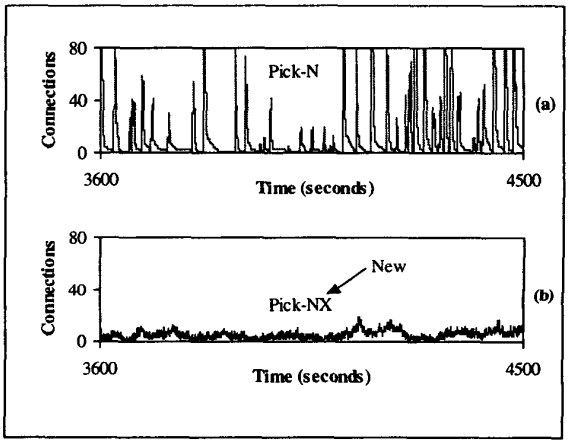

Figure 4(a)(b) - Oscillations in **Pick-N** and **Pick-NX**

We have also tested exponential smoothing of server load and found the results to be very poor. Response times were high and oscillations, similar to that of Figure 4(a) were observed.

5. Exploiting Content Knowledge

The server selection algorithms discussed in the previous section all rely on delayed load information only. We now add estimated partial content information to the switching decision (e.g., of the contents of a reverse cache at a remote server site).

Caching has been proven beneficial from two major points of view: improving response time to client and reducing network traffic. Web caching measurements from the network perspective have already been done. In 1999 several commercially available products were tested using the Polygraph proxy benchmark [16]. Participating vendors in the 1999 "bake-off" were: InfoLibria, Novell/Dell, Peregrine, and Squid. Only two products were capable of achieving 1500 - 1600 requests per second peak throughput, while the rest were operating at rates lower than 700 requests per second. These results do not seem promising for high-speed implementations. Thus, specialized capabilities (i.e., as can be implemented in a switch) are needed. Caching can reduce network bandwidth consumption. A 1995 trace-driven study [15] points out that out of 656 files (total size of 36.5 Mbytes) only 10% of all blocks account for 91% of the requests, so that taking advantage of a hierarchical caching system would be reasonable.

All this justifies adding content knowledge to the selection decision while still utilizing hierarchical caching infrastructures. Moreover, there are a number of enhancements over traditional query/reply protocols such as Internet Cache Protocol (ICP) [20]. Summary Cache, which uses Bloom filters to share knowledge of cache contents, reduces network bandwidth consumption by 50% over ICP [9] and cache digests reduce median service time by 100 milliseconds [15].

From the client point of view, caching aims to reduce response times. We next measure the effects of trivial server main memory caching on file transfer rates.

5.1 Experiments on transfer rate improvement

A test bed of two 700MHz Pentium III Windows NT 4.0 machines was used for our caching experiments. Each machine had a main memory of 128 Mbytes, Level I cache of 32-Kbyte SRAM (16-KBbyte data cache; 16-Kbyte instruction cache), and full-speed Level II cache of 256 Kbytes. Dynamically allocated is a disk cache (a main-memory caching structure) of about 20 Mbytes. The two machines were connected via a 1-Gbps switched Ethernet. File sizes of 1.1 Kbytes to 10 Mbytes were used. Using the Gigabyte Express file transfer program [10] each file was transferred 30 times when it was only disk resident, and 30 times when it was cache and main memory resident. Figure 5 shows the improvement in transfer rate based on case, 1) when a file is only found on disk, and 2) when some blocks of the same file are cache and main memory resident. It can be seen that the most significant improvement is obtained for files between about 23 Kbytes and 345 Kbytes.

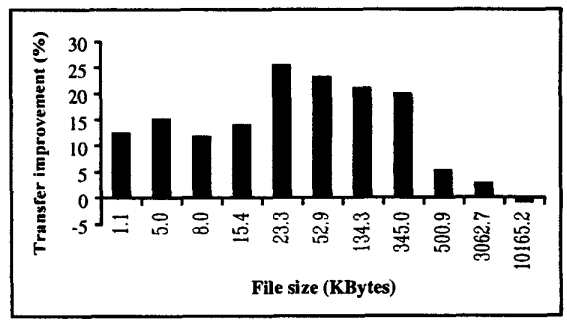

Figure 5 - Effects of caching on file transfer rate

5.2 Adding content knowledge to the selection

Content information should be shared between URL switches. The information consists of a data structure that contains `<server, content>` tuples. We are investigating the use of *Bloom filters* [9] as a compact data structure that can be both updated and read quickly. A key issue is cache validity time - how long a previously requested object will remain in a site cache. The Bloom filters will need to be updated on a frequency approximating cache validity times. Thus, an updated Bloom filter guess can be used to adjust the known load of the server. Both the server load value and partial knowledge of cache contents are estimates - the load due to its delay in reporting and the cache contents due to both the probabilistic nature of Bloom filters and the possibility that an object previously known (or guessed) to be in cache has been since removed from cache.

If **Pick-KX** is used, content knowledge is used to adjust the calculation of the X_j term in (4) as,

$$X_j = \frac{(L_{total} - (L_j - \alpha_j))}{L_{total}} \quad (5)$$

where α_j is an adjustment factor that is positive when it is guessed that server j may contain the requested object in its cache. The α_j value is updated periodically. The value of α_j is a function of how much serving a cached object requires less server load than serving a disk-resident object. For example, α_j can be the number of connections that a cache can independently handle (and thus relieving the server of this load).

6. Summary and Future Work

We have shown that how a server is selected from a set of distributed servers has a significant impact on response time. We have observed how caching can result in an improvement in transfer rates of about 10% to 25%

for a range of file sizes for a Windows NT platform.

If a least-loaded criterion (or **Pick-N**) is used to select a server, the load information must be current for this to be a "good" selection method. If load information is delayed, pure random selection (**Pick-1**) is better than **Pick-N**. A new method, built on the **Pick-K** and **LI** algorithms, called **Pick-KX** was evaluated and shown to result in lower server response times than existing methods when load information is highly delayed.

Future work will focus on employing the ideas discussed in this paper in the context of high-speed URL switching. This work includes:

1. Improving upon the simulation model to include more realistic server models and workloads.
2. Developing an analytical model - possibly based on a control theoretic view - of the observed behaviors to understand inherent instabilities in server selection with delayed load information.
3. Gaining a better understanding of how using cache content information, in conjunction with server load, can be used for selection.
4. Investigating methods of high-speed URL switching suitable for future 10-Gigabit Ethernet networks.

This future work will enable the scaling of Web services from localized server clusters to Internet-wide distributed server systems with automatic selection of servers.

Acknowledgements

The authors thank Mike Dahlin of the University of Texas at Austin for making available his simulation data and source code for model validation and for his helpful suggestions. The authors also thank the anonymous referees for their many helpful suggestions.

References

[1] The Apache Software Foundation, 2000. URL: *http://www.apache.org*.

[2] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants," *Proceedings of ACM SIGMETRICS*, pp. 126 - 137, 1996.

[3] L. Aversa and A. Bestavros, "Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting," *Proceedings of the 2000 IEEE International Performance*, Computing, and Communications Conference, pp. 24 - 29, February 2000.

[4] A. Bestavros, "Demand-Based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems", *Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing*, pp. 338 - 345, October 1995.

[5] K. Christensen, 2000. "URL Switching with Delayed Load Information.". URL: *http://www.csee.usf.edu/~christen/res3.html*.

[6] M. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *Proceedings of ACM SIGMETRICS*, pp. 160 - 169, 1996.

[7] M. Crovella, M. Harchol-Balter, and C. Murta, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load," *Performance Evaluation Review*, Vol. 26, No. 1, pp. 268 - 269, 1998.

[8] M. Dahlin, "Interpreting Stale Load Information," *Proceedings of the 19th International Conference on Distributed Computing Systems*, May 1999.

[9] L. Fan, P. Cao, J. Almeide, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", *Proceedings of ACM SIGCOMM*, pp. 254 - 265, 1998.

[10] Gigabyte Express, (v 2.93) Niwot Networks, Inc., 1999. URL: *http://www.niwot.com/Products.htm#Top*.

[11] J. Gwertzman and M. Seltzer, "The Case for Geographical Push-Cashing," *Technical Report HU TR-34-94*, Harvard University, DAS, Cambridge, Massachusetts, 1994.

[12] M. Mitzenmacher, "How Useful is Old Information? [Load Balancing in Dynamic Distributed Systems]," *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pp. 83 - 91, 1997.

[13] V. Pai, M. Aron, B. Gurav, M. Svendsen, H. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster based Network Servers," *Proceedings of the ACM Conference on Architectural Support for Programming languages and Operating Systems*, San Jose, October 1998.

[14] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, pp. 226 - 244, June 1995.

[15] A. Rousskov and D. Wessels, "Cache Digests," *Computer Networks and ISDN Systems*, Vol.30, No.22-23, pp. 2155 - 2168, November 1998.

[16] A. Rousskov, D. Wessels, and G. Chisholm "The First IRCache Web Cache Bake-off - The Official Report", April 1999. URL: *http://bakeoff.ircache.net/N01/*

[17] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection Methods for Replicated Web Servers," *Performance Evaluation Review*, Vol. 26, No. 3, pp. 44 - 50, December 1998.

[18] H. Schwetman, "CSIM18 - The Simulation Engine," *Proceedings of the 1996 Winter Simulation Conference*, pp. 517 - 521, December 1996.

[19] SQUID Web Proxy Cache, *URL: http://www.squid-cache.org/*

[20] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP) Version 2", *RFC 2186*, September 1997. URL: *http://ircache.nlanr.net/Cache/ICP/rfc2186.txt*