

Diffusion-based Caching along Routing Paths

Position Statement

Abdelsalam Heddaya
heddaya@cs.bu.edu

Sulaiman Mirdad
mirdad@cs.bu.edu

David Yates
djay@cs.bu.edu

May 26, 1997

Boston University
Computer Science Department
111 Cummington Street
Boston, MA 02215

Phone: (617) 353-8919
Fax: (617) 353-6457

Abstract

Caching for the Web can be beneficial in different ways: not only can it reduce network traffic and client response time, but it can also enable large scale server load balancing. In this paper, we present preliminary simulation data to characterize the performance of *WebWave*, a diffusion-based caching protocol for server load balancing that we have recently proposed. Initial results suggest that *WebWave* indeed achieves load balance, even under self-similar request load. Furthermore, the number of cache copies created by *WebWave* appears to be within acceptable levels.

1 Introduction

Much current Internet research and development centers around caching as a means to eliminate the redundant retransmission of information over expensive communication lines. In contrast, we advocate the use of caching also for the purpose of deploying and harnessing additional computation and storage, in the network fabric itself. Our original concept—called WebWave [12]—and subsequent theoretical development of it [13], calls for employing caching to distribute and balance load among servers that are embedded in the network itself. We also propose that cache copy discovery occur transparently and with minimal overhead, a goal that is achievable if cache copies are assigned to serve requests that naturally fly-by them *en route* to the home server. This latter point deserves elaboration first.

A caching system, regardless of its aim, can present a special difficulty to clients that use it: that of finding an appropriate cache copy to use. All the existing caching systems, such as Harvest/Squid [9] and HTTP proxies [5], as well as research proposals [6, 7, 8, 11, 14], enable a client to find an appropriate cache copy by one of two methods:

1. A resource discovery protocol.
2. A directory lookup.

In both cases, the client's request must experience a potentially significant added delay in order to perform what we call off-route communication. That is, for a client request to discover a suitable cache copy, it must either be diverted away from its natural route to the home server (as in HTTP proxies), or wait until cache probe messages are exchanged (as in ICP [16]).

Turning the home server into a cache directory server does not help, nor does assuming the existence of a scalable distributed name service. In either case, the cache directory lookup consumes multiple round-trip network delays just in the process of discovering a cache site. Furthermore, cache copies come into being suddenly, and possibly in large numbers, then disappear equally abruptly. Any distributed directory service that attempts to maintain a mapping from document identifiers to cache servers, must therefore expend significant overhead in keeping up with the high rate of change in this mapping. To eliminate such overhead, we place cache copies on the path that client requests take towards the home server.

Under our caching protocol, neighboring cache servers cooperate so as to smooth sudden changes in demand. Periodically, cache servers gossip and diffuse excess load to their neighbors. Cache servers diffuse load by relegating a fraction of future requests to their children (push) or parents (release), in the routing tree. Load at a cache server is defined as the fraction of time the cache server is busy during the last gossip period (*i.e.*, utilization). Based on the diffusion method for load balance, an overloaded server i periodically relegates a fraction of the load differential with each of its under loaded neighbors. Because WebWave diffuses load towards the source of the corresponding demand, it reduces network traffic. Effectively increasing the aggregate network bandwidth. This, combined with the higher service throughput that results from server load balance, improves client response time.

A server can relegate (push/release) an arbitrary fraction of all future requests to any locally cached document. However, the number of requests that can be relegated from server i to its child j for document d cannot exceed the number of requests forwarded by j in the first place. In other words, we prohibit i from shifting load to child j , that originates from another child k .

2 WebWave Simulation

For this performance study we used MaRS (Maryland Routing Simulator) [1]. MaRS is an event driven simulator designed to evaluate routing algorithms. In MaRS, a network consists of store-and-forward entities connected by links, routing algorithms and workload generators (static source-sink pairs). To evaluate the performance of WebWave, we adapted MaRS by introducing clients, servers, and documents. Our modifications to MaRS account for protocol dependent tasks. Servers and communication links are taxed for both load gossip and the creation of new cache copies; additionally, request packets are charged 2 msec for passing through the filter. The documents requested by a collection of clients are determined using a synthetic self-similar trace generated by SURGE [4], and explained in more detail later in this section. Each event in the trace file represents a document request. Client requests are scheduled using exponential inter-arrival times [3]. For each request, a client generates a request packet to the home server (document publisher).

In our model, a client request can be intercepted and serviced by an intermediate WebWave server caching the requested document. To achieve this goal, routers under MaRS were modified so as to be able to interact with the attached cache server and exchange cache related information. Each server provides its underlying router with filter code. As a packet moves through the network, routers inspect its header and determine its type. All request packets are passed through the filter to determine if the document will be served locally (a filter injected by a home server will intercept all requests to documents published by it). If a packet is intercepted, the filter composes a pseudo header, attaches it to the request and hands it over to the local server. Pseudo headers contain information—inserted by routers—such as the identity of the attached WebWave server that this packet flew-by. Servers are defined using two attributes: capacity (in bytes/sec) and gossip period length.

For each server, the simulator periodically computes and records the number of hits to each document and the load (utilization). For each client request, the simulator records the document requested, its size, the home server publishing the document, the server intercepting the request, number of hops to this server, and the response time.

To drive our simulation, we employ SURGE (Scalable URL Request Generator), a synthetic Web load generator designed and implemented by Barford and Crovella [4]. SURGE generates a sequence of file requests that satisfies the same statistical properties that characterize experimentally measured Web loads for a set of clients. In particular, the resulting synthetic request trace has Zipf file popularity distribution and heavy-tailed file size distribution [10]. Furthermore, the load mimics empirically measured temporal and spatial locality properties, which are critical for the accurate evaluation of cache performance [2]. This means that the trace exhibits the following characteristics:

- The fraction of requests for each file is inversely proportional to its rank by popularity.
- File size distributions show heavy tails with parameter $\alpha < 2.0$. As α declines, traffic generated becomes increasingly self-similar [15].
- When each request in the stream is expressed as the number of requests since the same file was last requested before, the resulting *stack distance* distribution is lognormal. This causes the generated load to mimic the locality of reference observed in real traces.

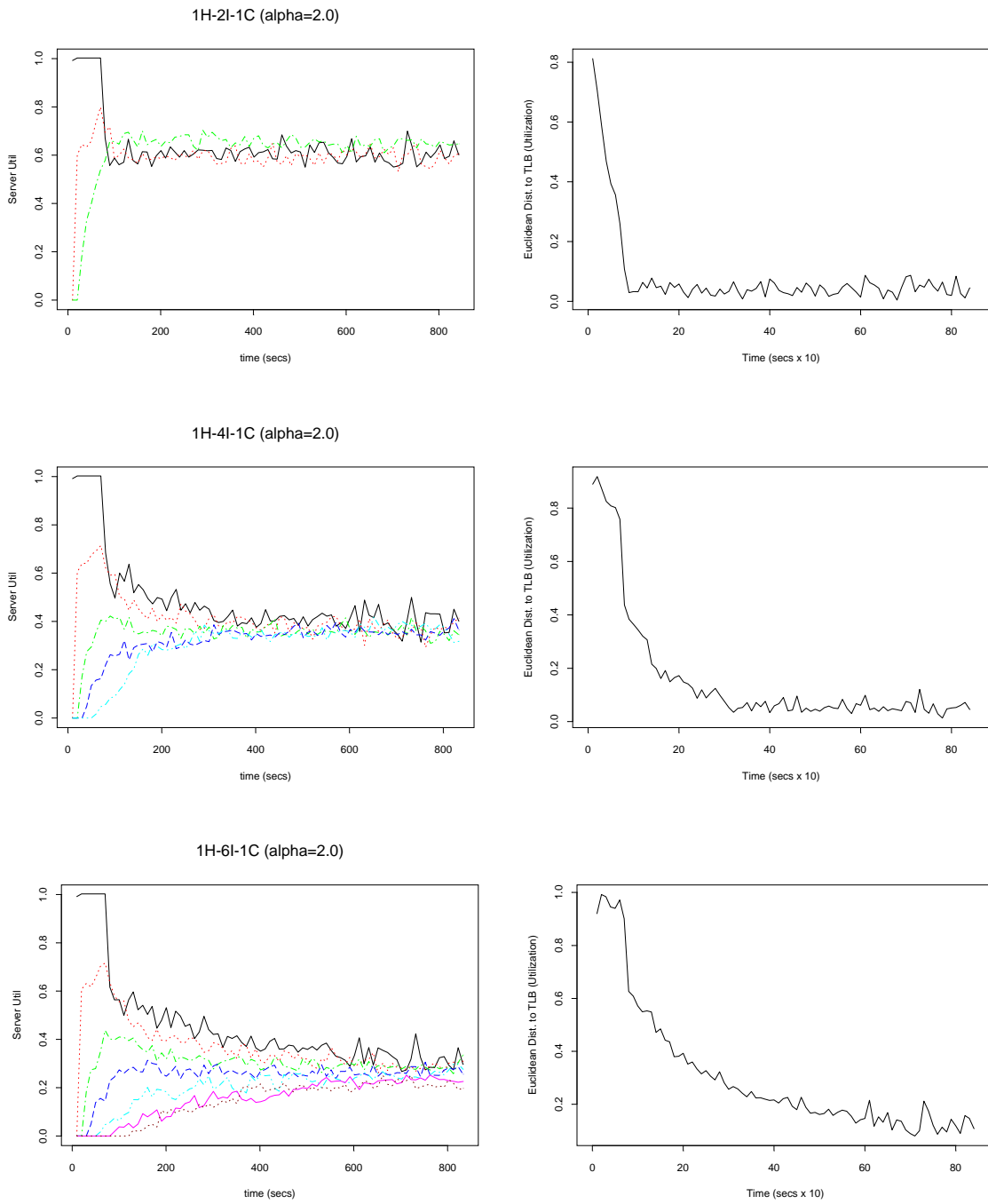


Figure 1: WebWave converges to approximate load-balance on a linear topology with three (top), five (middle), and seven (bottom) servers. The request stream in this case is exponentially distributed.

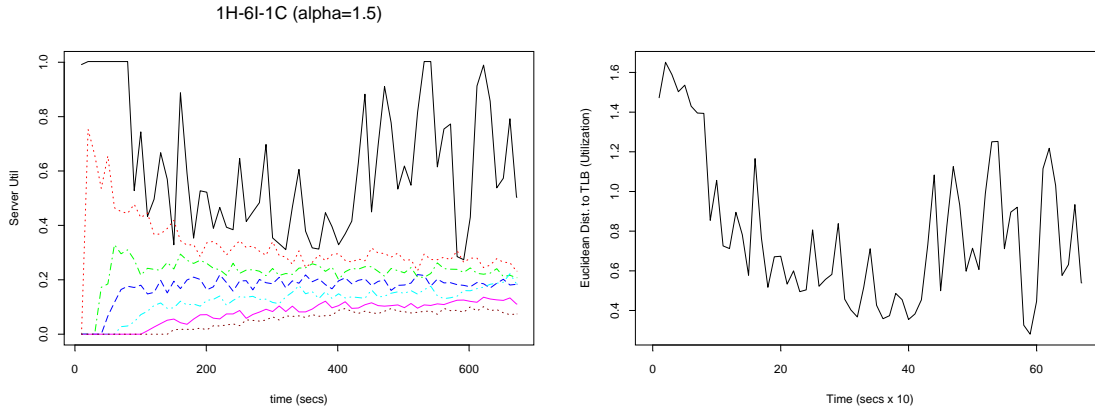


Figure 2: WebWave's convergence under self-similar Web request load.

3 Preliminary Results

Our simulations track the performance of WebWave by monitoring both a servers' utilization and its memory usage. Server utilization is defined as the fraction of time a server is busy over a 10 second interval (the gossip period). The data presented in this section arises from an implementation of WebWave in which load differences ≤ 0.05 are ignored, and in which the minimum amount of load that can be shifted is restricted to files requested at least twice during the last gossip period (set to 10 seconds). A linear network topology is chosen, with the home server and the client load source situated at both ends. A cache server runs at each intermediate node. While far from typical, this topology enables us to observe the ability of our protocol to split load along a path from a client to a home server. We are currently working on other topologies, including a single tree, multiple overlapping trees, and general graphs.

3.1 Convergence to Load Balance

Given that the primary goal of WebWave is to achieve load balance among home servers and cache servers placed inside the network, we present in this section preliminary results illustrating load convergence under different conditions. Figure 1 shows that cache server utilizations converge under a file request load that is not self-similar. The synthetic load generator produces this type of load for a parameter value of $\alpha = 2.0$ in the heavy-tailed file size distribution. This data agrees with our previous results [13], obtained under a much simpler simulator that assumes constant request rate, and that does not account for individual documents. Increasing the length of the path between client and home server, from three hops to seven, slows convergence considerably. This is not surprising, given the additive nature of load differences along a single path. Nevertheless, we are working on a quantitative analysis and explanation of convergence rate.

During periods of overload, a server runs at maximum utilization so long as it has pending requests in its queue. This is true regardless of how short the overload period may be, since during that period, the queue of pending request can grow appreciably. Because WebWave only relegates future load to neighbors, requests thus enqueued have to be served at the local server. This is evident in Figures 1 and fig:lb-self-similar, initially all requests are serviced by the home server resulting in a large number of pending request.

Self-similarity in Web request streams can have a significant impact on a caching system, since the intrinsic burstiness of such loads creates the possibility of long streaks of cache misses. Figure 2

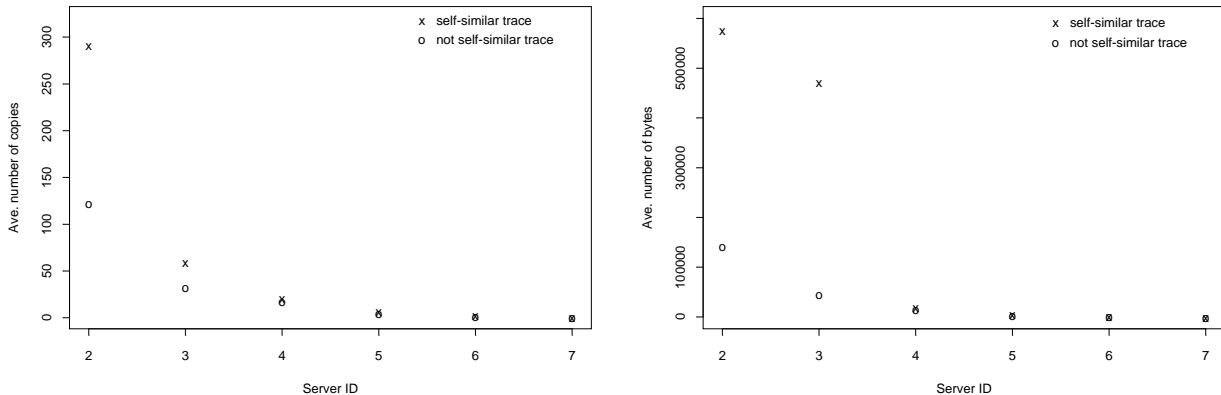


Figure 3: The number of copies created by WebWave decays quickly with distance from the home server. Self-similar request streams require more copies. Server 1 is the home server, and server 7 is the closest to the client.

shows the convergence of load under a self-similar synthetic load, obtained by setting $\alpha = 1.5$ in SURGE. This value of α , which corresponds to a *Hurst parameter* value of 0.75, is within the range estimated in [10] as typical of Web loads recorded by a variety of empirical studies. The results in Figure 2 support the hypothesis of convergence for all intermediate servers but apparently not for the home server. We suspect that the erratic behavior of the home server stems from the disproportionate contribution of the largest files (which tend to be the least popular, and hence not cached elsewhere) to its load. In other words, the document requests that account for the heavy tail of the file popularity distribution by size, tends to be concentrated at the home servers. Figure 2 can also be seen to suggest that the WebWave caching protocol, shields the small popular files from the erratic impact of heavy-tailed demand for large files.

3.2 Cache Memory Overhead

Another metric of performance is the memory overhead associated with cache copies. To measure this overhead, we compute the average number of copies cached by each server over 10 second intervals (the load gossip period). Figure 3 shows a comparison of this overhead with and without self-similarity in the request trace (at different cache servers in the 7-node network). Even though the number of cache copies decreases as the distance to the home server increases, self-similarity has a curious effect. As the distance from the home server increases, the number of cache copies in both simulations converge. The reason is that highly popular documents tend to move nearer to the client. Under the self-similar trace, however, the number of copies at caches closer to the home server, is approximately double that of the non-self-similar case. This effect is more pronounced when we take the number of bytes into consideration. The heavy tail of the popularity distribution by size, implies that large files—even if relatively unpopular—are requested frequently enough to have a strong effect on overall performance. Consequently, more of these files tend to be diffused from the home server to its child and grandchild (Figure 3 right), but not further.

4 Discussion

Diffusion-based caching, as employed in WebWave, presents specific advantages in terms of provably optimal load-balancing [13]. In this paper, we show preliminary detailed simulation results that support the theoretical analysis, not only in terms of the convergence of load-balance under exponential and self-similar request streams, but also with regard to the use of cache servers in the network itself. However, it is possible, in principle, to imagine a caching protocol that also achieves high quality *network load-balancing*. We believe that this is achievable within our framework, and we are currently pursuing this avenue.

We focus above on load balance and on memory consumption, yet there are other metrics of major importance that also benefit from WebWave caching. Both *network throughput* and *client response time* improve, albeit not to levels as certifiably high as load balance does. When the load is balanced, bytes are served at an equal rate at all servers. Therefore, the average number of hops traversed by a single byte equals half the maximum. If network throughput is measured by the total number of byte hops incurred, then, our simulations indicate that the network throughput is reduced by a factor of two, by virtue of the load-balance achieved by WebWave.

In contrast, response time is determined by the average number of hops that the *first byte* of every request must travel, not any byte. Given that the vast majority of requests access the smallest files, and given that those files are preferentially cached farther from the home server than larger files (see Figure 3), we deduce that the average number of hops per request is much smaller than the average number of hops per byte.

Acknowledgments. We would like to thank Paul Barford for supplying us with the SURGE code to generate our reference streams. Mark Crovella kindly supplied us with useful suggestions about self-similarity and helpful comments on an earlier draft of this paper.

References

- [1] C. Alaettinoglu, I. Matta K. Dussa-Zieger, and A. Shankar. MaRS (Maryland Routing Simulator)—version 1.0—programmer’s manual. Technical Report UMIACS-TR-91-107, University of Maryland at College Park, July 1991.
- [2] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *Proc. IEEE Intl Conf. on Parallel and Distributed Information Systems, Miami Beach, Florida*, Dec. 1996.
- [3] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants. In *Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 126–137, 1996.
- [4] Paul Barford and Mark Crovella. Generating representative web requests for network and server performance evaluation. Technical Report BU-CS-97-06, Boston Univ., Computer Science Dept., <www.cs.bu.edu/techreports>, May 1997.
- [5] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol—HTTP 1.0. Technical Report Internet Draft, Internet Engineering Task Force, Feb. 1996. Expires Aug. 1996.
- [6] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proc. 12th IEEE Intl. Conf. on Data Engineering, New Orleans, Louisiana*, Mar. 1996.

- [7] M.A. Blaze. *Caching in Large-scale Distributed File Systems*. PhD thesis, Princeton Univ., Dept. of Computer Science, Jan. 1993.
- [8] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report BU-CS-96-007, Boston Univ., Computer Science Dept., <www.cs.bu.edu/techreports>, Mar. 1996.
- [9] A. Chankhunthod, P.B. Danszig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A hierarchical Internet object cache. In *Proc. USENIX Annual Technical Conference, San Diego, California*, Jan. 1996.
- [10] M.E. Crovella and A. Bestavros. Self-similarity in World-wide Web traffic: Evidence and possible causes. In *Proc. ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, pages 160–169, May 1996. Accepted for publication in *Transactions on Networking*.
- [11] James Gwertzman. Autonomous replication in wide-area internetworks. Technical Report TR-17-95, Harvard University, April 1995.
- [12] A. Heddaya and S. Mirdad. Wave: Wide-area virtual environment for distributing published documents. In *Electronic Proc. ACM SIGCOMM'95 Workshop on Middleware, Cambridge, Mass.*, Aug. 1995. <www.acm.org/sigcomm/sigcomm95/workshop>.
- [13] A. Heddaya and S. Mirdad. Webwave: Globally load balanced fully distributed caching of hot published documents. In *Proc. 17th IEEE Intl. Conference on Distributed Computing Systems, Baltimore, Maryland, USA*, May 1997.
- [14] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems -or- your cache ain't nuthin' but trash. In *Proc. Winter USENIX Technical Conf., San Francisco, CA*, pages 305–314. USENIX, January 1992.
- [15] Kihong Park, Gi Tae Kim, and Mark E. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. 4th Intl. Conf. on Network Protocols*, pages 171–180, Oct. 1996.
- [16] D. Wessels and K. Claffy. Internet cache protocol (ICP), version 2. Technical Report Internet-Draft, IETF Network Working Group, Apr. 1997. <<ftp://ds.internic.net/internet-drafts>>.