TaBLA: A Client-Based Scheduling Algorithm for Web Proxy Clusters

Girija Narlikar Lakshman Y. N. Tin Kam Ho
{girija, ynl, tkh}@research.bell-labs.com
Bell Laboratories, Lucent technologies, 700 Mountain Ave, Murray Hill, NJ 07974

Abstract

As client populations in ISPs continue to rise, it becomes necessary for ISP proxy caches to efficiently handle large numbers of web requests. In this paper, we examine the performance of client-side load balancing schemes that help select a proxy from an array of proxies that are equidistant from the client. The current most popular solutions include choosing a random proxy based on either the URL requested, or the web server from which the URL is requested. Based on an analysis of proxy traces, we propose a new client-side scheduling algorithm "TaBLA". The algorithm creates a redirection table that can be loaded into the clients' browsers. Trace-driven simulations indicate that our algorithm significantly improves average response time and average slowdown compared to the purely randomized schemes.

1 Introduction

Caching web proxies play an important role in reducing latency and bandwidth usage in today's networks. The amount of sharing (and hence the increase in cache hits) has been shown to increase with the number of clients [25, 4, 11, 10, 7]. However, a single proxy host has a finite amount of capacity, limiting the number of clients that can be placed behind each proxy. Large Internet Service Providers (ISPs) are therefore adding several proxy hosts within their networks to provide an acceptable quality of service to an ever-increasing population of clients. Several vendors now offer products and services that supply hardware or software to manage clusters of web proxies [14, 17, 1, 9]. A typical solution includes Level-3/4 or Level-7 switches that intercept requests from multiple clients and redirect them to different proxies depending on the IP address of the target web server (at Level-3/4), or the target URL (at Level-7). The switches need to provide high redirection throughput, fault tolerance in the face of switch failure, and load balancing across multiple web proxies.

An alternate solution that avoids the high costs for the proprietary hardware, software, installation and management of the redirectors is to provide the redirection mechanism in the client (web browser) itself, which is the subject of study of this paper. Several client-side strategies for selecting a web proxy have been proposed earlier. The Cache Array Routing Protocol (CARP) from Microsoft [27, 23] applies a randomizing hash function to each URL at the client to determine which proxy from a set of equidistant proxies to redirect the web request to. Each client uses the same hash function, so requests for the same URL go to the same proxy; this preserves cache hit rates even though requests are distributed across multiple proxies. Furthermore, the load on each proxy is reasonably balanced due to the large number of URLs requested from each proxy. A drawback of the scheme is that requests to the same web server get redirected through different proxies. Typically, when a single client browses, she requests multiple objects from the same server (for example, images from one or more web pages) in quick succession. Because of this, HTTP 1.1 introduced persistent connections with pipelining. This has been shown to provide significant benefits in reducing the user-perceived latency [18, 8] due to temporal locality in the servers accessed by each client and reduction in the number of packet round-trips between the server and the client. In CARP, each URL is redirected to a potentially different proxy. Therefore successive requests for the same remote server may go through different proxies, thereby significantly reducing the benefit of persistent connections.

A possible solution to the persistent connections problem is to apply a randomizing hash function to the target web server name (domain) instead of the target URL. As shown in Section 5, this allows a significant number of cache misses to take advantage of persistent connections between the proxy and the remote server. For example, in an experiment with an ISP proxy trace, 81% of the cache misses can take advantage of persistent connections in a domain-level scheme, compared to 44% in a URL-level scheme. However, randomizing at a domain level also leads to load imbalance at high load levels, because of a small number of very popular domains. This trade-off between the two randomizing schemes is shown in Figure 1. These results indicate that a domain-level strategy with better load balancing is required to obtain consistently low response times.

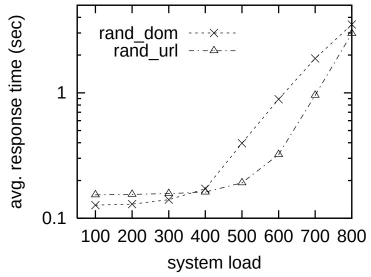


Figure 1: The trade-off between randomizing either the server name ("rand_dom") or the URL ("rand_url") when choosing a consistent proxy for outgoing web requests. "System load" represents the rate at which requests arrive at the proxy (defined more precisely in Section 5).

In this paper, we propose a new client-side scheme called ***TaBLA*** (Table-Based Load Assignment), which is based on a static analysis of the recent history of client request patterns obtained from proxy logs. We use the analysis to identify web sites attracting high traffic and file types with large mean sizes which are then assigned to the individual proxy machines according to a specific partitioning scheme. Each client is provided a small table with this information; the client (browser) looks up this table to determine which proxy to hit for each URL. The table can change with every round of static analysis of proxy logs.

To derive and validate the TaBLA scheme we study workloads from two different web proxies; Section 2 presents our main observations, which include invariants in the volume directed to popular domains, and the average file sizes of large mime types. Based on the analyzed data we create look-up tables that a client would use (Section 3). We use trace-driven simulation (Section 4) to evaluate the performance of TaBLA, and compare it to other client-based randomizing redirection strategies (Section 5). The results show that TaBLA improves average response times by up to factors of 3, and the average slowdown ("stretch") perceived at the proxy by up to a factor of 12. Related work is described in Section 6, and we conclude in Section 7.

2 Workload Characterization

We examined the traces from two proxies (see Table 2): a proxy from a major national ISP, and the sv.cache.nlanr.net proxy managed by NLANR [19].

	ISP	NLANR
Proxy Type	Lowest level	Upper Level
Clients	United States	Asia and Pacific
Dates	1/16/99–2/10/99	11/11/99–12/26/99
# requests	1.88 million/day	1.28 million/day

Figure 2: Details about the proxy traces we studied.

2.1 Load prediction

We first measured the load of traffic targeted to each remote web site; the target web site (also referred to here as the "domain") is defined as the symbolic name that is part of the URL recorded in the logs[1]. We define the load as a combination of the number of requests and bytes transferred for each request. For a domain with r requests and b bytes, the load is computed as $r + b/4000$. This definition of load was determined from the system parameters, and is explained in Section 4. Figure 3 shows the load targeted to the remote web sites sorted in decreasing order of load, over the entire period of the trace. The top few domains each account for a moderate fraction of the load. For example, the top 1000 domains account for 55% of the total load in the ISP logs and 51% for the NLANR logs. However, the important observation is that the remaining domains each individually contribute to around 0.01% of load or less. Therefore, a random assignment of these light domains to proxies should result in a well-balanced load. In contrast, a naive or random assignment of the popular domains can result in a significant load imbalance (see Section 5). We therefore chose to examine the popular domains more closely.

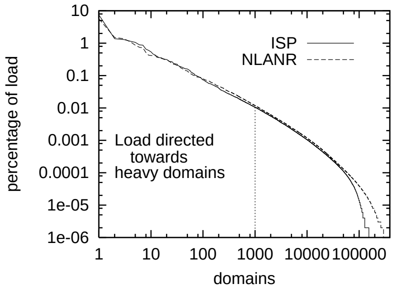


Figure 3: Plot of total load over the entire period for each domain in the trace for both data sets.

[1]For example, "www.foo.bar.com" is the "domain" for the URL "http://www.foo.bar.com/index.html".

Data reduction and normalization. The top 1159 sites with highest load in ISP trace (0.7% of all sites) and the top 1079 sites in NLANR trace (0.28% of all sites) were labeled as ***heavy domains***[2]. The per-domain load was computed as a timeseries with totals on an hourly as well as a daily basis. The series for each domain was normalized with respect to the total load for that domain.

Analysis of daily totals by sites. A cluster analysis for the NLANR data was performed on the 1079 heavy domains sites, using the k-means procedure with Euclidean distance. This is a widely used unsupervised learning procedure for discovering natural groupings in a dataset [3, 26]. The domains were grouped into 20 clusters (see Figure 4). For many sites the access patterns are nonstationary. They experienced a sharp burst a few times during the month, but negligible load at other times. Prediction of traffic for these sites is difficult.

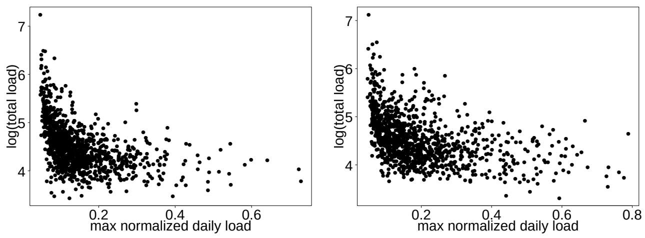
In contrast, the two largest clusters (clusters 1 and 2 in Figure 4) do not contain very sharp bursts, and are potential targets for strategic load prediction. The maximum normalized daily load (the peak height) is a good discriminator for identifying such sites with stable traffic. Analysis of the 1159 heavy domains in the ISP data shows similar characteristics. We did not detect any stable, intra-day complementary patterns between pairs of domains which would have allowed packing the domains together.

Figure 5 shows the plots of total load (at each site) for the month versus the maximum normalized daily load. These plots show that accesses to the sites having highly concentrated (unpredictable) traffic do not contribute heavily to the total load through the respective proxies. The bulk of the traffic was from those sites having less than 20% of their total load occurring in one day (maximum normalized daily load < 0.2); this traffic can be reasonably predicted in our load balancing strategy.

2.2 Analysis of file types and sizes

We decided to study if the less common, but very large files are predominantly of different mime types than the typical text and images. We attempted to guess the file types by the suffixes of the URLs. As expected from a heavy tailed distribution of file sizes [4, 6], the most popular file types are typically not large (with mean and median in the 2–12 KB range). To look for those file types that deserve special treatment due to their large sizes, we found those with an average of at least 10 requests per day, and with a median file size of 20 KB or above. We call these file types the ***heavy types***; examples of heavy file types include .mp3, .dll, .zip, .exe, .mpg, etc. Such a list of heavy types is used in our scheduling algorithm to detect and separate requests that are likely to incur a large response (Section 3).

[2]The heavy domains were selected by applying a simple filters with low cut-offs for the total byte traffic and number of requests on the set of all domains.



(a) ISP (b) NLANR

Figure 5: Plot of period total load (log scale) vs. maximum normalized daily load for (a) ISP data and (b) NLANR data.

In summary, we drew the following conclusions from our study of the web traces.

1. A fine-grained prediction of load beyond well-known daily variations is difficult. However, at an aggregate level, such as over a day, the load directed at most heavy domains does not show significant variation, and can be reasonably predicted. The remaining domains do not *individually* contribute significant load.
2. File types (as derived from the URL) for large files are fairly consistent across days. This gives us a reasonable chance of predicting if an outgoing request will return a response significantly larger than the average file size.

3 A Table-Driven Proxy Selection Scheme

We now present details of TaBLA which are based on the analysis from the previous section. For both logs, we pick 500 of the heavy domains that show the least variability (smallest normalized daily maximums), as described in Section 2.1. From now on we refer to these 500 domains as the ***heavy domains***. The heavy types and the heavy domains are assigned to the individual proxy machines with the aim of separating out the big requests as well as balancing the overall load.

If there are P proxies and the heavy types account for fraction $1/h$ of the total load, then we assign $P \times 1/h$ of the proxies to exclusively serve heavy file types. The heavy domains are sorted in increasing order of their average file sizes; we then split this list into $P \times (1-1/h)$

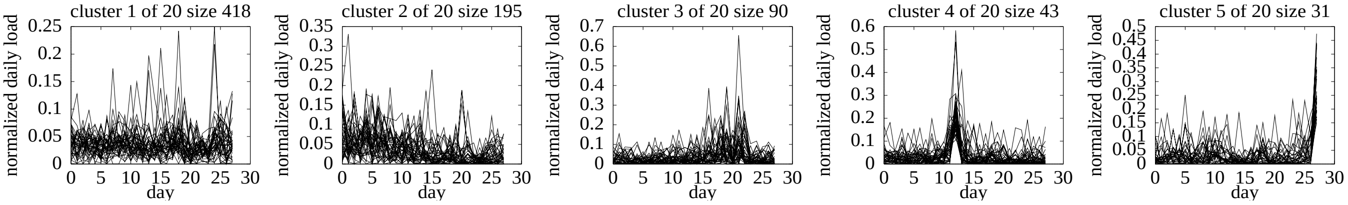


Figure 4: 5 of the 20 clusters formed by normalized daily totals for the NLANR data. The cluster size is the number of domains that fall within that cluster; the remaining 15 clusters each have at most 30 domains. Only clusters 1 and 2 contain domains with low variability in load (measured as maximum normalized daily load); these are selected by TaBLA as heavy domains.

partitions of equal load, and assign one partition to each of the remaining proxies[3]. We assume that all proxy machines have identical capacities; otherwise, the load can be spread in proportion to their capacities and the scheme works with no significant variation. The motivation for separating heavy types and sorting heavy domains by size is to reduce the variance in request sizes arriving at each proxy; large variances can affect the slowdown of tasks in the request queue[4].

Each client has a table encoding this assignment of the heavy types and heavy domains to individual proxies. When a client needs to access a web resource, it consults the table. If the resource type is a heavy type, it goes to the proxy responsible for that heavy type; else, if it belongs to a heavy domain, it asks the proxy responsible for that domain for the resource; else, it applies a simple hash function on the domain part of the URL to compute the identity of a proxy from which to seek the desired resource.

Our traces showed significant load variations from day to day, especially between weekends and mid-week. However, the total load pattern from week to week was highly consistent. Therefore, in our experiments the tables were computed once from a week's worth of logs to be used for the next day. Considering periods longer than a week yielded no additional accuracy in load prediction. Experiments with a variety of different 7-day periods for load prediction yielded results consistent with those reported in the paper. The choice of the prediction period (one week for our traces) and the frequency of table updates (once a day for our traces) may vary for different trace data. They are both parameters fed to the TaBLA algorithm that are derived from the data itself. After each round of static analysis of logs from all the proxies, the updated tables are placed in a specific path on each proxy.

Table distribution. A key issue here is how to distribute the tables to the clients. The simplest way (one that does not require large modifications to existing client software and other web infrastructure) is to use the automatic proxy configuration facility supported by the major web browsers such as Netscape[20] Our tables can be encoded within a standard function in a JavaScript file. When a browser starts up, it obtains the latest version of the table by downloading the JavaScript file (with a direct connection to any of the proxies). For all subsequent requests, the browser executes the FindProxyForURL function in the JavaScript file to determine which proxy it should contact. A time-to-live field can be attached to the JavaScript file and the functions in the JavaScript file can directly obtain the latest table if the current table is stale. In an ISP context where the service provider typically provides clients with all the software needed to connect (including a fully configured browser), this should not present a logistical problem. More dynamic update scenarios are possible if the browser and proxies can understand a header field indicating when the last table update took place.

Dynamic Issues. Another issue is non-availability of one or more proxies. This can be because of a proxy machine failing or getting swamped by a "hot spot"—an unexpected deluge of requests to a group of web pages. Since our tables are constructed based on recent history, and we have deliberately avoided any active collusion among the proxies, it is not possible to predict transient hot spots. When a client fails to get a response from a proxy for a time-out period, it attempts to get the same resource from another randomly chosen proxy; it tries to revert to the table-based policy for the troubled proxy after a specified amount of time. If the service delay is indeed caused by a hot spot or proxy failure, this has the effect of spreading out the responsibility for serving that proxy's traffic

[3] Here the load for heavy domains is computed after excluding the requests that are of heavy types.

[4] The effect of task size variance on the slowdown depends on the scheduling policy at the request queue. For example, with a FCFS policy, slowdown is proportional to variance [16].

throughout the proxy bank. The advantage of the table based scheme will be diminished during hot spots or proxy outages. If an entirely new domain becomes highly popular and stays that way for an extended period of time, its presence will be captured during the log analysis and the subsequent table updates will reflect it.

4 Experimental Setup

The client-side scheduling algorithms that we evaluated were: applying a randomizing hash function to the requested URL (called ***Rand$_{url}$***), applying a randomizing hash function to the domain from the requested URL (called ***Rand$_{dom}$***), and TaBLA (as described in Section 3). All clients share the same randomizing hash function, so all requests to a URL (in Rand$_{url}$) or all requests to a domain (in Rand$_{dom}$ or TaBLA) map to the same proxy. This reduces the storage requirement for each proxy cache and maintains good hit rates. Rand$_{url}$ is identical to CARP [27] except that we use a different randomizing function. We developed a trace-driven simulator to study the performance of the different scheduling algorithms. A network simulator was necessary to avoid replaying the proxy logs on a real network (many pages are not accessible in such a replay). We also chose to simulate the proxy instead of using a real proxy implementation for two reasons: (a) a simulator allows us to study the effects of various system parameters on the relative performance of the scheduling schemes, and, (b) we can more easily modify scheduling policies for the pending web requests at the proxy. We only measure response times *at the proxy*, that is, the difference between the time it takes for a request to arrive at a proxy, and the time the proxy finishes serving the request. All default values were measured on a 400 MHz Pentium PC. We did not implement dynamic handling of hot spots in our simulation.

4.1 Network model

The network model implemented is a simple one, similar to previous work [21]: each request of size x bytes that results in a cache miss at the proxy takes a round-trip time L_s to set up a connection to the remote server (if necessary), followed by time $L_s + x/b$; here b is the bandwidth available between the proxy and the server. We assume that all connections between a proxy and the server get an equal share of a common link of bandwidth B_p that connects the proxy to the Internet, but each individual connection can only use a maximum bandwidth B_s (see Figure 6). Similarly, all clients share a common link of bandwidth B_p to the proxy. If a request from a client for an object from a server S arrives at the proxy when a persistent connection to S is currently alive, it does not incur the additional L_s round-trip delay to set up a connection. If the request arrives while a connection to S is being established for an earlier request, it must wait for the connection to be opened before the proxy can start receiving data from S for either request. The default parameter settings used are B_p = 12MB/s[5], $L_s = 0.1sec$, and B_s = 150KB/s for all remote servers. We do not expect a real network to behave in such a uniform, time-invariant manner. However, we are only interested in studying the high-level effects of the different scheduling policies averaged over a large number of web requests; for such a study we believe a coarse-grained network model is sufficient.

We assume that a remote server keeps a persistent TCP connection open to a proxy for at most 10 minutes, and closes the connection before this period if no request has been sent by the proxy for the last 30 seconds. Deciding when to close a persistent connection at the server remains a difficult problem. We have not seen definitive solutions, and different servers adopt different timeout periods. Our parameters are in ranges deemed reasonable by several researchers (e.g., [8]).

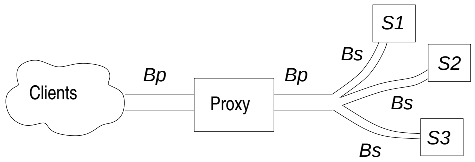


Figure 6: Link bandwidths in the network model for the network connecting the clients, each proxy, and the remote servers ($S1, S2, S3$ in this figure).

4.2 Proxy model

There are two network queues at each proxy host: the ***server queue*** for requests with replies pending from servers and the ***client queue*** for replies to be (or being) sent to the client. The requests in the server queue continuously share bandwidth as described above, while replies in the client queue share bandwidth depending on the request scheduling policy employed at the proxy CPU. We assume that the CPU writes replies to the network buffers at a throughput equal to the bandwidth B_p of the shared link to the clients. The proxy CPU requires a time of $150\mu s$ to establish a new TCP connection to either the client or the server (as measured on a 400 MHz Pentium Linux PC). The size S_{buf} of the network buffer is set to 64KB (the Linux default).

[5] Essentially the maximum throughput of a 100Mbits/s Ethernet link.

The scheduling policies for servicing the client queue studied here are first-come-first-serve (FCFS), round robin (RR), shortest-processing-time-first (SPT), and shortest-remaining-processing-time-first (SRPT). Processing time is assumed to be proportional to the size of the remaining reply. Both RR and SRPT use a time quantum equal to the time required to fill one network buffer (of size $S_{\text{buf}} = 64\text{KB}$). Thus, RR approximates a multithreaded server with an unlimited number of threads[6]. There is no limit on the length of either network queue. The sequence of events for a web request at the proxy is shown in Figure 7. Priority is given by the CPU to incoming requests over replies to clients.

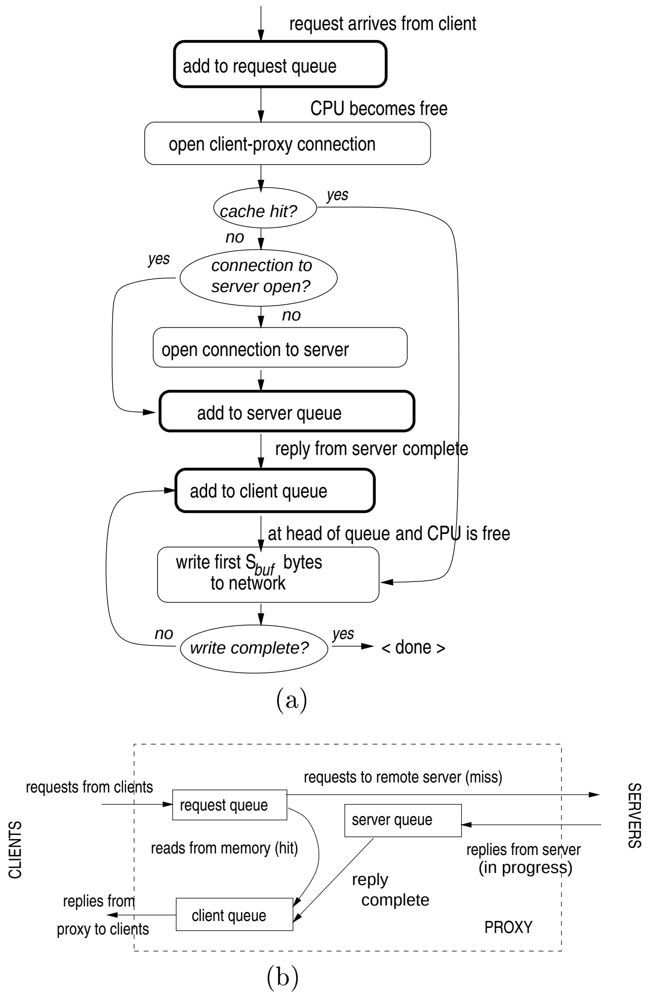


Figure 7: (a) Sequence of events at the web proxy. Actions that may result in delays due to system load are outlined in bold. (b) Flow of requests and replies between the queues.

[6]It also simulates a more realistic event-driven proxy with a small number of threads writing pending replies to the clients in a round robin manner.

All experiments assume that we have a bank of 10 proxies to choose between for any web request; the proxies do not communicate with each other. We assume for simplicity that all request hits are served out of the main memory of the proxy (and not the disk). Because our experiments look at total footprints of under 2 GB per proxy, this is a reasonable assumption for typical high-performance proxies. The simulation run begins with a cold cache; the first request to every object (URL) is assumed to be a miss, while subsequent requests are hits.

The definition of load defined in Section 2 is based on the above system parameters. If, as an approximation, we assume each request is a cache miss, then the proxy spends $300\mu s$ to open connections to the client and remote server, and writes the reply at the rate of 12MB/sec. Thus writing 3774.6 bytes of data require the same amount of time as one request. Therefore, we assign a load of $r + b/3774.6$ to r requests that add up to b bytes.

4.3 Workload and table generation

The NLANR and ISP traces that we had access to are from single proxy hosts, respectively containing on average 15 and 22 requests per second. This traffic load is insufficient to stress one or multiple proxies. To simulate higher loads, we shrank the time scale by successively larger factors. We also accordingly shrink the 30-second and 10-minute windows over which a persistent connection may be left open, to avoid exaggerating the benefit of persistence at higher loads.

We ran the load prediction scheme described in Section 3 on both traces from an arbitrary choice of seven consecutive days; the resulting redirection table was used to route traffic for the eighth day. The table was not dynamically updated while the eighth day's traces were being simulated.

We measure the average response times for requests once they arrive at the proxy. We also measure the average ***stretch factor***, which is the ratio of the actual response time for a request divided by the time it would have taken in the absence of any load on the system. Stretch is a metric independent of file size. Since clients typically do not like to wait long for small requests (like a typical html document along with small embedded images), but are willing to wait longer for larger downloads, we expect that a low stretch factor is an appropriate indicator of a good scheduling algorithm.

5 Experimental Results

For the ISP logs, we dedicate 1 out of the 10 simulated proxies to exclusively handle requests for the heavy types. According to our prediction, the top 46 largest files types contribute around 10% of the total load; these were therefore selected as the heavy types. The NLANR logs contained a higher number of files of the heavy types, and so we dedicated 2 proxies to handle load from the 72 largest file types; these accounted for around 20% of the total load in the previous week's logs.

5.1 Variation of performance with load

Here the ***system load*** is the factor by which the original trace is speeded up. For example, the ISP logs contain on average 21.76 requests/sec, and a speedup of 500 results in an average of 10,800 requests/sec directed to the 10 proxies together.

Figure 8 shows the relative performance of the three schemes, namely, Rand_{url}, Rand_{dom}, and TaBLA, for both the proxy logs. The scheduling policy at the proxy for sending back replies to clients is FCFS. For both logs, under extremely low or extremely high system load, all three schemes exhibit similar performance. However, as expected, Rand_{url} suffers from a slightly higher response time even at low system loads because fewer of the cache misses take advantage of persistent connections. As the system load increases, the imbalance in the load assigned to each proxy by Rand_{dom} begins to play a major role in the overall performance of the system. TaBLA outperforms both the randomized schemes.

With the FCFS scheduling policy at the proxy, the stretch factor is significantly affected by variance of file sizes [16]. TaBLA successfully partitions files by size (see Figure 12), resulting in lower variation in the sizes of replies at each proxy. Therefore it achieves a lower stretch when the system is loaded. However, our prediction schemes for load partitioning are not as close to ideal as Rand_{url}; therefore, at extremely high system loads, even this slight imbalance causes the performance of TaBLA to approach that of Rand_{url}.

In practice, when a proxy sends back replies to clients, the replies get interleaved, instead of being sent out in strict FCFS order. For example, when multiple processes attempt to write to sockets in Linux, the processes get scheduled in a round robin (RR) order, with each process writing data equal to the size of the socket buffer (64KB). We therefore also simulated this RR policy for scheduling of replies to clients at the proxies. (The buffer size assumed was 64KB.) The results are shown in Figure 9. The RR policy significantly reduces the observed stretch for the Rand_{url} and Rand_{dom} policies, because a large number of small replies no longer get queued up behind a few large replies. This policy improves the TaBLA scheme to a lesser extent, because TaBLA already reduces the variance in file sizes that arrive at each proxy. For the ISP logs, TaBLA still results in up to a factor of 3 reduction in response time compared to Rand_{url}, and up to a factor of 12 reduction in average stretch. For the NLANR logs, the performance improvements due to TaBLA are reduced to around 20% in response time, and 30% in stretch compared to Rand_{url}. TaBLA shows lower improvement over Rand_{url} for the NLANR logs because its prediction of load is less accurate compared to the ISP logs.

5.2 Sources of improvements due to TaBLA

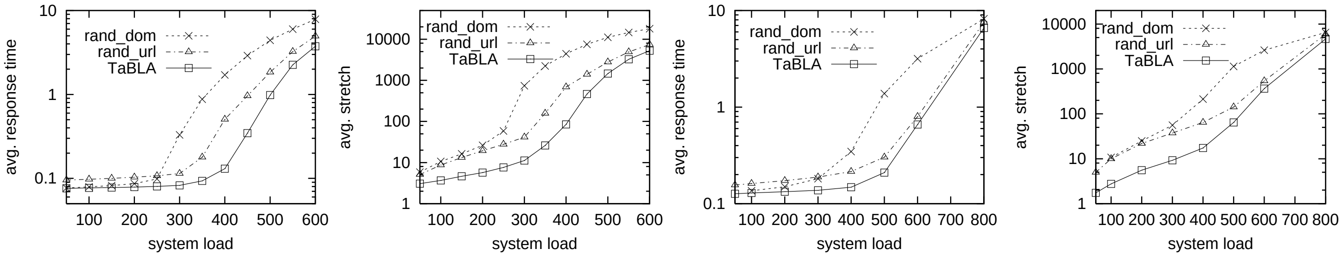
Recall that our scheme TaBLA uses two different optimizations: ordering of requests to proxies by file size, and balancing of load due to the most popular domains. We first demonstrate the combined effects of these optimizations across file sizes and cache hits or misses, and then attempt to separate out the performance benefits due to both optimizations.

Table 10 shows a breakdown of response times and stretch factors for files of different sizes, for both cache hits and cache misses, at a system load (speedup) of 350 for the ISP logs. Stretch factors of less than 1 arise due to persistent connections; the service time (under no load) for each job is computed assuming that a new connection must be opened to the server. The results show that TaBLA significantly reduces both the response times and stretch factors of cache hits; this improvement is due to a good balance of load and reduced variability in request sizes that are directed at each proxy. For cache misses, the network delay dominates the total response time; however, due to a better use of persistent connections, TaBLA results in lower response times for cache misses compared to Rand_{url}. In TaBLA at most $p-1$ domains get split across the p proxies while partitioning the expected load. Consequently, for the dataset in Table 10, TaBLA gets 0.8% fewer connection hits than Rand_{dom}.

Figure 11 shows the performance benefits when only load balancing is applied at the domain level, using RR scheduling at the proxy. Here domains are not ordered by average request size, and large file types are not separated out. We see that the majority of improvement over Rand_{dom} is due to the prediction and balancing of load, particularly as the system load increases. Because large files cause a bigger disruption with the FCFS policy at the proxy (compared to RR), the size-based separation contributes a bigger improvement to TaBLA for FCFS[7].

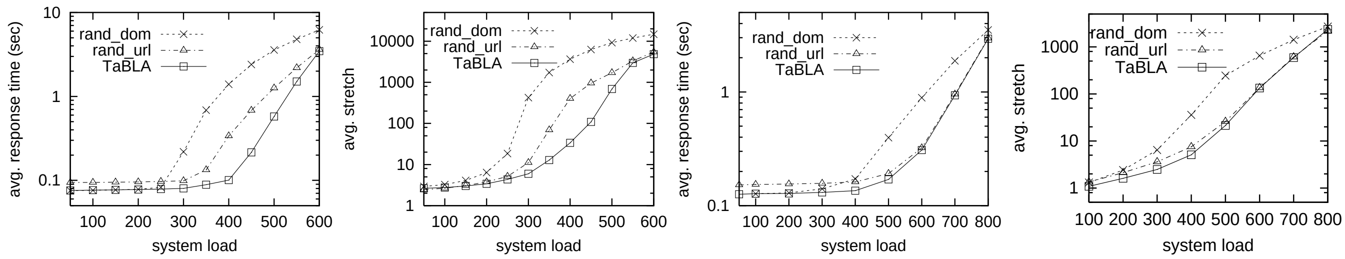
Figure 12(a) compares the load directed to the 10 proxies by each of the three schemes over the entire day's ISP logs (NLANR data looks similar). Although

[7]Results for FCFS not shown due to space constraints.



(a) Response time ISP (FCFS) (b) Stretch ISP (FCFS) (c) Response time NLANR (FCFS) (d) Stretch NLANR (FCFS)

Figure 8: Performance of the three client-side scheduling schemes for both NLANR and ISP proxy logs. The FCFS algorithm was used at the proxy to schedule replies to clients. Metrics for comparison are the average response time and the stretch factor.

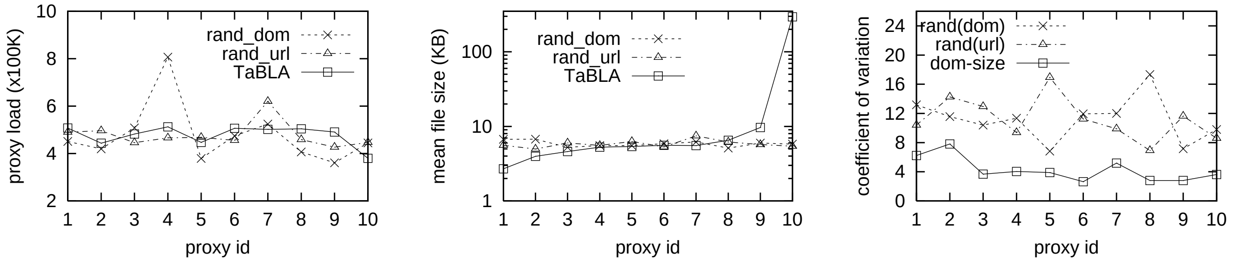


(a) Response time ISP (RR) (b) Stretch ISP (RR) (c) Response time NLANR (RR) (d) Stretch NLANR (RR)

Figure 9: Performance of the three client-side scheduling schemes for both NLANR and ISP proxy logs. The RR algorithm was used at the proxy to schedule replies to clients. Metrics for comparison are the average response time and the stretch factor.

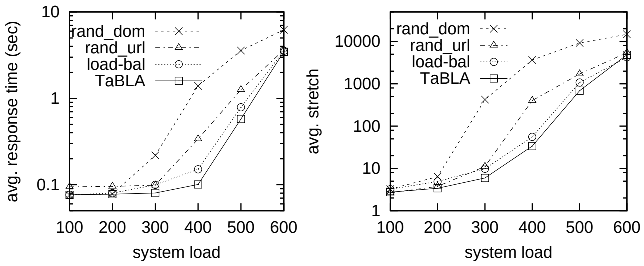
File size (B)	Cache hits			Cache misses				All requests	
	#reqs	resp.T	stretch	#reqs	resp.T	stretch	conn-hits	resp.T	stretch
(a) Rand$_{url}$ + RR									
$\leq$ 500	572908	0.026	169.048	212953	0.207	1.029	94547	0.075	123.518
0.5K-8K	331831	0.027	89.766	429985	0.220	0.997	206468	0.136	39.664
8K-128K	126261	0.046	25.653	200352	0.355	1.050	67308	0.236	10.561
128K-2M	882	0.305	6.947	2246	2.821	1.077	309	2.112	2.732
$\geq$ 2M	80	1.446	5.945	130	27.483	1.054	5	17.688	0.633
(b) TaBLA + RR									
$\leq$ 500	578485	0.006	40.670	207376	0.137	0.683	161027	0.041	30.120
0.5K-8K	336855	0.008	25.135	424961	0.152	0.688	364370	0.089	11.499
8K-128K	128335	0.011	7.320	198278	0.282	0.821	147836	0.175	3.375
128K-2M	918	0.093	2.800	2210	2.687	1.011	597	1.926	1.536
$\geq$ 2M	81	0.704	2.505	131	26.849	1.015	8	16.860	1.584

Figure 10: Breakdown of average response times and stretch factors by file size and cache hit status using RR with (a) Rand$_{url}$ and (b) TaBLA at the clients for the ISP trace. Here "conn-hits" is the number of cache misses that took advantage of an existing, open (persistent) connection to the remote server.



(a) (b) (c)

Figure 12: (a) Total daily load, (b) mean file size and (c) coefficient of variation ($= \sqrt{\text{variance}}/\text{mean}$) of file sizes directed to each proxy by the three client-size schemes, for ISP data.



(a) Response time (b) Stretch

Figure 11: Performance of simply balancing the load (labelled "load-bal") at the domain level across proxies for the ISP data, compared to the performance of Rand$_{dom}$ and TaBLA.

this only shows loads at an aggregate level, we see that TaBLA and Rand$_{url}$ both balance the load better than Rand$_{dom}$. Figure 12(b,c) shows how successful TaBLA was in separating requests by sizes for the ISP data; the NLANR results look similar.

5.3 Size-based scheduling at proxies

An alternate way to tackle the queueing of small requests behind larger requests is to reorder the replies at the proxy based on the sizes of the replies. We tried the Shortest-Processing-Time-first (SPT) and the Shortest-Remaining-Processing-Time-first (SRPT) at the proxy for sending replies to clients [16]. In SPT, the smallest pending reply is written to completion over the network. For SRPT we preempt the sending of replies after a buffer-size (64KB) worth of data is written, and select the reply with the smallest remaining size next. We found no improvement to any of the client-side schemes due to these proxy-based policies; RR remains the best scheduling policy at the proxy for all the client-side schemes.

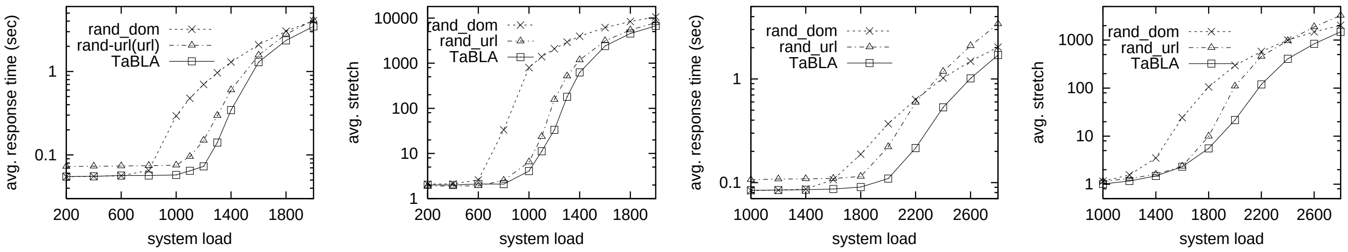
5.4 Effect of a faster network

We experimented with higher bandwidth (an order of magnitude higher) links in the system. We set B_p to 1Gbits/s (128MB/s), while B_s was increased to 1.5MB/s; the latencies and other overheads were kept unchanged. The definition of load due to requests had to be changed to reflect a lower weight for the number of bytes transferred compared to the number of requests. The results are shown in Figure 13. TaBLA on the NLANR logs now shows a bigger improvement over Rand$_{url}$ compared to Figure 9, because a faster network can better handle a load imbalance due to unpredicted, large-sized requests. Further, the advantage of persistent connections is more pronounced because the network latency was not changed. NLANR logs incurred more cache misses, and therefore a bigger percentage of requests benefit from persistent connections (compared to the ISP logs). For both traces, with a faster network, TaBLA improves the response time over Rand$_{url}$ by up to a factor of 2, and the stretch factor by up to a factor of 5.

6 Related Work

A number of client-based strategies for proxy selection have been proposed before. For example, CARP [27, 23] and a scheme by Karger et al [15] involve hashing the target URL, similar to the Rand$_{url}$ scheme with which we compare TaBLA. Pai et al [22] proposed using a front-end redirector to provide load balance and locality of reference for web server clusters; this work was subsequently extended to handle persistent http connections [2]. We have attempted to do away with front end machines/switches in our approach; also, the concerns at a cluster of proxies are somewhat different.

Several systems have been proposed for distributed or hierarchical web caching [7, 10, 5, 12]. They re-



(a) ISP (RR) (b) ISP (RR) (c) NLANR (RR) (d) NLANR (RR)

Figure 13: Performance of the three client-side scheduling schemes for both NLANR and ISP proxy logs, at higher network bandwidths. The RR algorithm was used at the proxy to schedule replies to clients. Metrics for comparison are the average response time and the stretch factor.

quire moderate to large amounts of communication between proxies to share content, which is not required in our client-side scheme. Round-robin DNS is commonly used to distribute load across web servers. A similar scheme could be applied to web proxies, but that would result in different proxies serving the same web pages. This would reduce the hit rates and distribute content across all the proxies, which we avoid in TaBLA. Server-side mechanisms such as reverse proxies or DNS-based redirection are complementary to the client-side schemes described here.

Our size-based scheduling scheme is similar to the SITA-E policy proposed by Harchol-Balter et al [13]. SITA-E was specifically designed for non-preemptive distributed server systems where the task size distribution is heavy tailed; it significantly outperformed other more common scheduling schemes. However, we do not see as large a gain because in our case large requests are prevented from blocking small ones for a too long due to task preemption at the proxy.

7 Conclusions and Discussion

Based on analysis of recent traces from two different web proxies, we have proposed a new client-side table-based scheduling algorithm to redirect web requests to a cluster of web proxies. Our simulations demonstrate that the scheme provides significant improvement over pure randomizing hash-based schemes, which apply a consistent hash function to the web URL or domain name to determine the target proxy.

A weakness of our experimental methodology was the unavailability of proxy traces for sufficiently heavy web traffic. Shrinking the timescale was our solution. We could also scale up the size of the set of files requested according to the increasing load. However, this approach, which is used by workload generators like SpecWeb [24], involves adding synthetic requests to the real workload. Validating our scheme on originally heavy traffic is a subject of future work. Also, we did not study the effect of cache misses due to domains being reassigned to different proxies when the redirection table is updated. Our workload analysis indicates that the table need not be recomputed often, thereby mitigating the effect of the initial cache misses when a new table is installed.

Our scheme can replace the currently popular redirecting switches that need to be installed into an ISP network. These switches rely on proprietary hardware and software to provide load balancing and fault tolerance. Our scheme requires client-side adaptations to proxy failures or hot spots; this may result in slower response times during such conditions compared to the redirectors. However, we contend that the switches themselves add cost and complexity to the network, and are additional points of failure that can affect a large number of clients.

References

[1] Alteon Web Systems. Web OS Traffic Control Software. `http://www.alteonwebsystems.com/`.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient support for p-http in cluster-based web servers. In *Proc. USENIX Annual Technical Conference*, June 1999.

[3] G. Ball and D. Hall. A clustering technique for summarizing multivariate data. *Behaviorial Sciences*, 12(2):153–155, March 1967.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. INFOCOM*, March 1999.

[5] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *Proc. Usenix Technical Conference*, San Diego, CA, January 1996.

[6] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based traces. Tech. Rept. 1995-010, Boston University, 1995.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. SIGCOMM*, pages 254–265, September 1998.

[8] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of the INFOCOM '99*, March 1999.

[9] Foundry Networks. Serveriron switches. `http://www.foundrynet.com/`.

[10] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *6th Workshop on Hot Topics in Operating Systems*, Cape Cod, MA, May 1997.

[11] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proc. Usenix Symp. Internet Technologies and Systems (USITS)*, Monterey, CA, December 1997.

[12] C. Grimm, H. Pralle, and J.-S. Vöckler. Load and traffic balancing in large scale cache meshes. In *Proc. Intl. WWW Caching Workshop*, June 1998.

[13] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *IEEE J. Parallel and Dist. Computing*, 59:204–228, 1999.

[14] Inktomi Corporation. Traffic Server. `http://www.inktomi.com/`.

[15] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. *Computer Networks and ISDN Systems*, 31(11-16):1203–1213, May 1999.

[16] L. Kleinrock. *Queuing Systems*, volume II: Computer Applications. Wiley, 1976.

[17] Lucent Technologies. IPWorX Web Performance Solutions. `http://www.lucent.com/`.

[18] J. C. Mogul and V. N. Padmanabhan. Improving WWW latency. In *Intl. WWW Conference*, October 1994.

[19] National Laboratory for Applied Network Research (NLANR). A distributed testbed for national information provisioning. `http://ircache.nlanr.net/Cache/`.

[20] Netscape Communications. Automatic proxy configurations. `http://home.netscape.com/`.

[21] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, 26(3), July 1996.

[22] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. ASPLOS*, October 1998.

[23] K. W. Ross. Hash routing for collections of shared web caches. *IEEE Network*, pages 37–44, November 1997.

[24] Standard Performance Evaluation Corporation. Specweb99 release 1.01, November 1999.

[25] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond hierarchies: Design considerations for distributed caching on the internet. In *Proc. ICDCS*, Austin, TX, May 1999.

[26] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 1998.

[27] V. Valloppillil and K. W. Ross. Cache array routing protocol v1.0. Internet draft, February 1998.