# A Caching Relay for the World Wide Web

Steven Glassman

Systems Research Center
Digital Equipment Corportion
130 Lytton Ave.
Palo Alto, CA 94301
steveg@src.dec.com
http://www.research.digital.com/SRC/people/Glassman_Steve/bio.html

### Abstract

*We describe the design and performance of a caching relay for the World Wide Web. We examine how the behavior of users affects the performance of the relay. From an analysis of the data from the relay, we develop a model that describes some aspects of the Web as a whole. (A version of this paper is available at http://www.research.digital.com/-SRC/people/Glassman_Steve/-CachingTheWeb.html)*

## 1 Overview

In January 1994, we set up a caching World Wide Web [10] relay for Digital Equipment Corporation's facilities in Palo Alto, California. We use a relay to reach the Web because Digital has a security firewall that restricts direct interaction between Digital internal computers and machines outside of Digital. We added caching to the relay because we wanted to improve the relay's performance and reduce its external network traffic. Clients use the relay for accessing the Web outside of Digital; requests for internal Digital pages go directly to the servers. When the relay has a current version of a Web page in its cache, it handles the request locally without ever contacting the remote server.

The relay handles about 2000 requests from 60 users each day. To date, it has served over 100,000 HTTP [3] and Gopher [2] requests from 300 users for 40,000 different pages. FTP [1] (file transfer) requests are handled by an existing (non-caching) FTP relay. We don't allow WAIS [9] (Wide Area Information System) or NNTP [6] (network news transfer protocol) requests through the relay.

A caching relay has several potential advantages over a non-caching relay; it:

- Reduces latency on requests for cached pages
- Reduces overall network load and remote server load
- Provides availability when a remote server is unavailable

And possible disadvantages are that it:

- Returns stale versions of a page when the remote version has changed and the cache doesn't know
- Increases latency on requests for non-cached pages
- Increases local administrative complexity and cost for disk space

We found that 30-50% of all requests are satisfied from the cache (a cache hit). When there is a cache hit, the relay supplies the page in 1/4 to 1/6 the time of retrieving the page over the network. The reduction in network load is equivalent to the cache hit rate. As described below, we did not find any of the disadvantages to be significant.

## 2 Our cache design

We based our cache design on studies of the logs from an existing Digital Web relay. We ran a simulation of our cache design using these logs to estimate the cache size, the cache hit ratio, and the effectiveness of various heuristics for estimating the validity of pages in the cache.

All cached Web pages are stored as Unix files, with the file name formed from the URL [8]. We plan for our cache to hold up to 80,000 Web pages. Since pages have an average size of 20K-25K bytes, we expect our cache size to reach about

2 GBytes. Rather than use a single giant directory with 80,000 files in it, we hash the file name into one of 4,096 sub-directories so that each directory holds about 20 files. The sub-directories are organized into a 3 level deep hierarchy where each internal directory has 16 sub-directories. For example, the URL http://www1.cern.ch/WWW94/-Abstracts/GNA.html is cached under the file name www1.cern.ch%%WWW94%%Abstracts%%-GNA.html. The hashed value of the URL is 5C7 (hex) and so the file is found in the directory /www/cache/5/C/7.

In our original design, the cache was entirely protocol-independent because this was simpler and we weren't yet sure how many different protocols we would have to handle. Each page was cached under a name based on a 128-bit fingerprint of the *entire* request. This meant that cache hits occurred on textually identical requests, but not on non-identical but equivalent requests. Therefore the cache was not getting hits when it should. This problem led us to abandon being protocol-independent, since the parsing of the requests to determine equivalency is protocol-dependent.

Our relay only handles two protocols - Gopher and HTTP. The Gopher protocol is very simple and a simple approach suffices. HTTP requests are more complex and must be parsed, but understanding the additional fields of the request and the response assists in caching.

The main problem in designing the cache is to avoid returning a stale page from the cache when a newer version of the page is available on the remote server. Neither HTTP nor Gopher has any provision for a server notifying the cache when a page changes, so the cache must estimate a time period during which it believes the page will not change.

A cached page is termed *valid* if the cache believes that it has the newest version (i.e. it believes the page hasn't changed on the remote server). A cached page *expires* when the page's valid time period ends (although expired pages remain in the cache). A *stale* page is a valid page that has in fact changed (i.e. the cache's estimate for the valid period was wrong).

If a request comes in for a valid page, the cache returns its version without contacting the remote server. If a request comes in for an expired page, the cache *refreshes* it from the remote server. Depending on the remote server, a page is refreshed either by retrieving an entire new copy or by checking whether the page has changed and retrieving a new version if it has.

The cache's algorithm for estimating the valid period of a page must balance two concerns. If the estimate is too long, then the probability of returning stale data increases. If the cache's estimate is too short, then the cache will have fewer valid pages and thus a lower cache hit rate.

The HTTP protocol allows for an "Expires" time in the response. Ideally, the cache could use this value to set the valid period for the page in the cache and never risk returning a stale page. Unfortunately, we have found only 6 pages (out of 28,000) that have an expiration time on them (and 5 are from the finger gateway). Therefore, we use the "Last modified" time, which is provided in nearly all HTTP responses, as a simple estimate of when the page will change next. For example, if a page has not changed in a week, we assume it will not change in the next week. If there is no modification time (the Gopher protocol doesn't give one), we use a default of 6 hours. We limit every page to be valid for at least one hour and no more than two weeks.

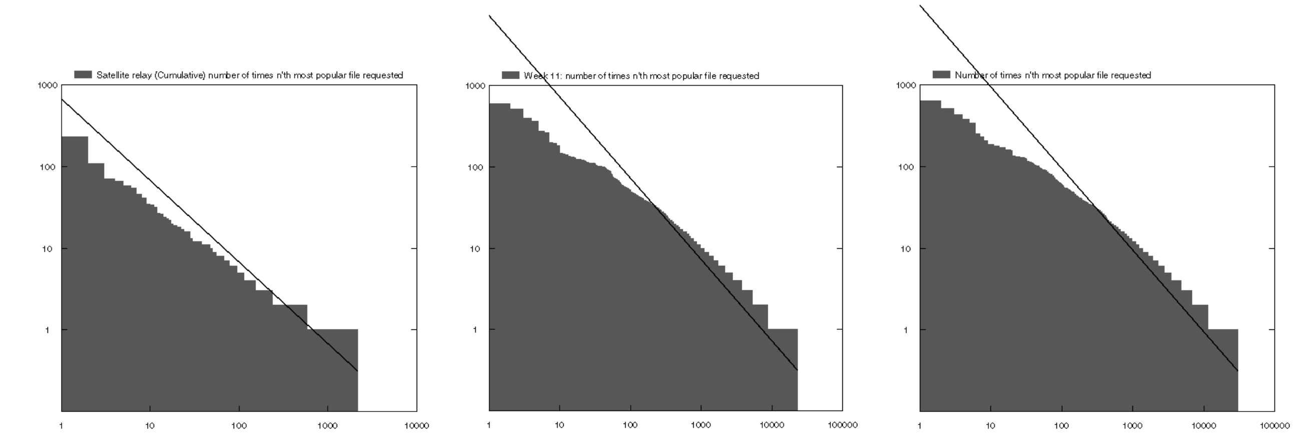
When an expired page is requested, we refresh it and calculate an expiration time as above. However, if there is no "Expires" or "Last modified" time, we compare the new version with the expired one that is still in the cache. If the page has not changed, we use the time of the expired version as the "Last modified" time and increase the valid period. If the page has changed, we reduce its valid period.

HTTP requests allow "Accepts" fields that list the acceptable MIME [4] (multimedia) formats of the response. (This is another reason why we must parse HTTP requests.) A cached page that is not in an acceptable MIME format cannot be returned. Fortunately, the HTTP specification is very generous about when a format is acceptable. The cache returns a cached page if it matches *any* acceptable format (and since most requests will accept any format, matching the "Accepts" is usually trivial). We have yet to find a cached page that was not in a format acceptable to a request.

One of our original goals for the cache was to minimize the external network load of the relay. We therefore did not implement any strategies such as pre-fetching (retrieving pages from the server speculatively) or refreshing cache pages as they expire. Now that we have data from operating the cache, we plan to examine these possibilities and compare the costs of the extra network load with the benefits of improved cache performance.

## 3 Satellite relays

Our caching relay can also be used as a *satellite* relay. A satellite relay is a secondary relay usually located at a site with poor connectivity or bandwidth to the net. It helps by providing a cache near the users, so they have access to cached pages at local network speeds.




(a) (b) (c)

Figure 2: Here are three examples of the distribution of page requests showing how they fit the pattern of a Zipf distribution. Each graph shows the page requests sorted in order of popularity. The horizontal axis is the log of the number of the page. The vertical axis is the log of the number of times the page was requested. The straight line in each graph shows the number of requests predicted by a Zipf distribution. The graphs show: (a) the cumulative distribution of requests (over 4 weeks) for the satellite relay in Australia, (b) the requests for the eleventh week on the primary relay, and (c) the cumulative distribution of requests (over 12 weeks) for the primary relay. Notice how similar the graphs look.

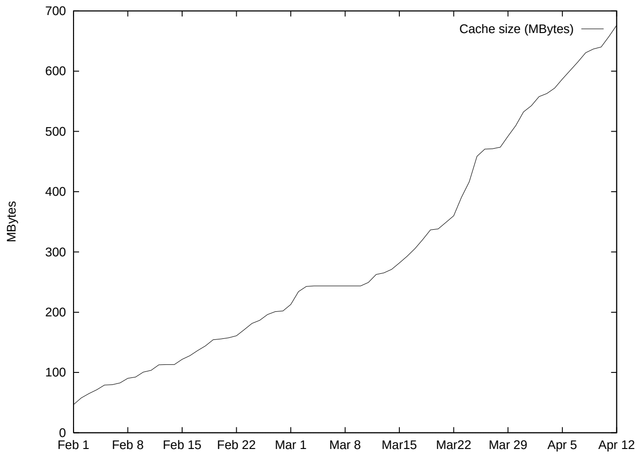



Figure 1: The growth of our cache. No page has ever been deleted. There are flat spots in the curve each weekend and from March 3-10 when caching was turned off.

Clients at the site of a satellite relay use it for all of their Web accesses (including Digital internal pages). When the satellite relay doesn't have a requested page in its local cache, it contacts the primary caching relay for the page (or contacts the server directly for internal pages). Although the cache hit ratio of a satellite relay is somewhat lower due to the smaller population of users, there is a much greater gain from cache hits.

Satellite relays can be organized into a tree-structured hierarchy with cache misses in lower relays percolating up through higher relays until the requested page is found. In practice, though, it seems sufficient to have a two-level structure consisting of many satellite relays sharing one, or more, primary relays.

# 4 How Caching Affects the Socks Protocol and Mosaic

The Socks protocol used by our relay is a simple protocol that encapsulates another protocol (in our case, Gopher or HTTP) to allow it to pass through an intermediate relay. The Socks protocol precedes each request sent to the relay with a header identifying the client and the intended external server. The relay checks to make sure the client has permission for the request, makes the connection to the external server, forwards the request, and relays the response back to the client.

We modified the Socks protocol in two ways to support the relay:

1. We added a special marker sequence to denote end-of-request. As described above, the relay was originally protocol-independent and needed a reliable way to know when it had received the entire request. Since the relay is now protocol-dependent, the marker is no longer necessary, but it is still convenient and it also supports the next change.

2. As part of the end-of-request marker, we include two flags for the client to control the cache. The first flag is the "refresh" flag, which forces the relay to refresh the page from the remote server. This flag is used by clients when they believe the cached page is stale. The second flag is the "don't cache" flag, which tells the relay not to cache the result of this request. This flag is used if the request is somehow private to this particular user and should not be cached.

We made small modifications to NCSA's Mosaic [5] to send the marker and flags at the end of the request. We also modified Mosaic's "Reload" command to send the flag telling the relay to ignore the cache. We don't have a command to set the "don't cache" flag. The "Reload" command is the only direct user interaction with the cache.

## 5 Cache performance

We keep full logs of all requests to the relay and of the relay's response. Every day we extract statistics from the logs describing the cache performance, cache size, number of requests, sizes of results, and number of users. We also publish that day's most popular pages and the pages that were first referenced on that day.

The cache gets hits on 30-50% of all requests. It can service a request resulting in a cache hit in about 1.5 seconds, while a cache miss takes 6 to 9 seconds. We have found that the caching relay adds no appreciable latency to client requests on cache misses.

We have designed our relay around a 2 GByte cache of 80,000 pages, but the cache does not need to be this big. We have never deleted any page from our cache and it currently holds about 28,000 pages occupying 630 MBytes. Figure 1 shows the growth of our cache.

The valid portion of the cache (those pages that are not expired) currently consists of only 5,400 pages and takes 140 MBytes. When we simulate a very conservative algorithm to clean out the cache, it removes about 16,000 pages and frees up 360 MBytes.

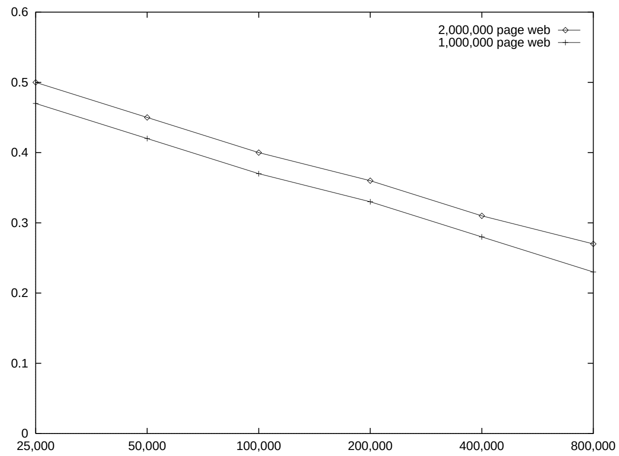



Figure 3: The cache miss rate predicted by caching samples taken from a Zipf distribution. The horizontal axis is the number of samples that formed the cache. The vertical axis is the predicted fraction of samples that cause cache misses. The upper curve is for a domain of 2,000,000 entries, the lower curve for a domain of 1,000,000 entries.

We are still collecting data about the cache. We will continue to determine values for the cache size and algorithms for cleaning out the cache.

Recently, we added a satellite caching relay at a Digital site in Australia. It has a much smaller user population (20) and fewer requests per day (a few hundred), but it still gets respectable cache hit rates (25-35%). A cache hit takes about 0.5 seconds while a cache miss takes 10-20 seconds. (The satellite relay goes faster on cache hits because all of its clients are local and responses go at local network speeds. The primary relay has clients all over; some are in Australia.)

## 6 Limits to caching

We have found that there is a fairly stable hit ratio for our cache. About 1/3 of all requests are for pages that have never been requested before (cache miss), 1/3 of all requests are for pages that are valid in the cache (cache hit), and 1/3 of all requests are for pages that have been requested before, but the result in the cache is no longer valid (cache expired).

A Zipf [11] distribution is a distribution where the probability of selecting the $i'th$ item is proportional to $1/i$. Zipf distributions are empirically associated with situations where there are many equal-cost alternatives (e.g. referring to books in a library). Figure 2 shows how the distribution of the number of requests per Web page compare to a Zipf distribu-

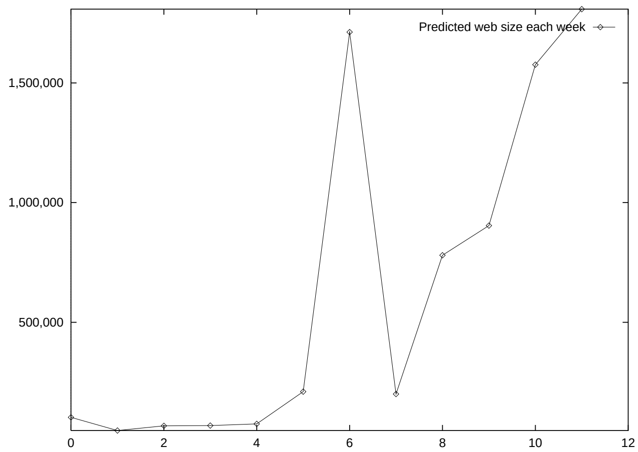



Figure 4: The weekly predictions of the size of the domain of the Zipf distribution based on the number of pages viewed each week. We calculate the size $N$ of the domain of the Zipf distribution, such that $k$ samples from Zipf distribution result in $d$ distinct values. For each week, we measured $k$ (the number of requests) and $d$ (the number of different pages requested). The calculation is very sensitive to the value of $d$, and so the spike at week 6 is due to one user visiting a *lot* of pages once.

tion.

We can analyze the performance of a cache of entries taken from a Zipf distribution. If we assume that the total size of the domain of the distribution is $N$, then we can predict $d$, the number of distinct entries, that will occur in $k$ samples from the distribution. Even if we never remove a cache entry, the cache will get $d$ cache misses (one miss the first time each entry occurs). The cache miss rate is then $d/k$, i.e. the number of different entries divided by the total number of samples. In Figure 3, we see the predicted cache miss rate for different $N$ and different numbers of samples $k$.

If we use a Zipf distribution to model page requests on the Web, then $N$ is the size of the Web, $k$ is the number of requests, and $d$ is the number of different pages requested. At the current operating point of the caching relay (100,000 requests, 30,000 entries in the cache) and assuming a Web size of one million pages, we expect about 40% of all requests to cause misses. This is fairly close to our observed value. In addition, this predicted miss rate is pretty stable. Doubling the size of the Web, or doubling the number of requests, only makes a 3-4% difference in the predicted cache miss rate.

We can reverse this calculation and use the Zipf distribution to calculate $N$ from $k$ and $d$. Using the logs from the relay, we can count the number of requests and the number of different pages requested each week. When we use these values as $k$ and $d$ respectively, we can calculate $N$ (very roughly). Figure 4 shows the size of the Web as predicted by these numbers each week.

There are a several points to make about this analysis:

- The predictions assume a Zipf distribution; slightly different assumptions can give very different results.
- The value $N$ that we are predicting is the size of the domain of the Zipf distribution. This is not necessarily the size of the actual Web. For instance, there may be large portions of the Web that our users can't or don't access (e.g. pages behind network firewalls, pages we are not authorized to view, and pages in languages we don't understand). In addition, there may be large portions of the Web that our users haven't found yet. Our users can only request pages that they have already visited or that are directly reachable from those pages. As our users explore the edges of their known Web, the effective size of the Web grows.
- The Web itself is growing, se we are estimating a moving target.
- Our weekly data are noisy and the calculations are very sensitive to the values given. Reducing $d$ in for the twelfth week by 4% reduces the predicted value of $N$ by 33%. Or, if the user who caused the spike in Figure 4 at week 6 had visited 500 fewer pages (6% of all requests that week, 11% of the distinct pages for the week), the predicted size of the Web would have been 1/3 as big.
- Our data from the primary relay (which we used in the calculations) only covers requests for pages outside of Digital. However, users request internal pages most often, so we are missing data about an important part of the distribution of requests. If our data included all of the internal pages (which the data for the satellite relay in Figure 2(a) does), then we believe that it would appear even more like a Zipf distribution. We are working to gather data about user requests for internal pages so we can make more accurate predictions.

## 7 Do Users Notice Caching?

As an experiment to gauge the impact of caching on users, we turned off caching in the relay for one week. (We left the relay running, but we did not use the cache to satisfy any requests, nor did we

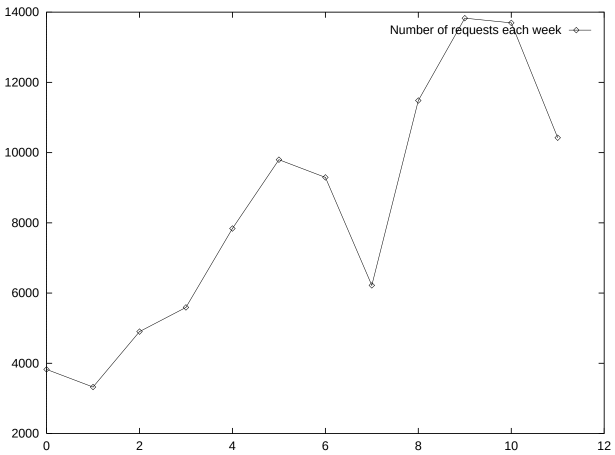



Figure 5: The number of requests made on the relay each week. The drop in requests at week 7 corresponds to the week when the cache was turned off.

cache any new results.) We were somewhat surprised when no user commented on a change in the cache's performance. On the other hand, when we examine the number of requests made on the relay each week (Figure 5), there was a noticeable decline (33-45%) in that week when compared to the preceding and following weeks.

We have some explanations for the results of this experiment:

- We have very good (Ethernet quality) access to the internet. Connections to remote WWW servers (even through the network security firewall) have good bandwidth and low latency. For clients outside of the cache's area, accesses to external pages are often dominated by the bandwidth of their connection to the relay.

- The average response time is 6-9 seconds for accessing uncached remote pages. This appears to be acceptable for most users. There is also high variance in response times. (We have measured a factor of 30 difference in response time for accesses to one page just a few minutes apart.) Users are prepared for unpredictable delays when accessing a page.

- Given the cache's hit rate, most requests are still handled by remote servers. The user is used to the response times of remote servers.

- Much of our Web usage is attributable to users browsing and the Web. Users may believe (maybe even want to believe) that they are the first ones to request a page. They are not surprised when a (seemingly) obscure page is not in the Web.

- Despite not explicitly noticing the presence or absence of caching, users do appear to be sensitive to the performance of the relay. In the week when caching was turned off, the average response time increased and users did look at fewer pages.

There are two ways to increase the impact of caching on users - increase the cache hit rate and increase the relative performance difference between a cache hit and retrieving a page from the net. We are examining various strategies to improve the cache's hit rate. For sites with poor net connections, a satellite relay has better relative performance.

## 8 Pages change

As we described above, we use the heuristic of keeping pages valid in the cache for a time equal to the time since the page last changed. This seems to be a reasonable assumption that would balance the competing desires for a high hit rate and a low stale data rate.

To verify this assumption, we randomly check 5% of our cache hits to see if they have returned stale data. For those pages, we return the cached page and then retrieve the page from the remote server and compare it with our current version. We have found that roughly 8% of our cache hits return stale data. When we looked at the cases where the page had changed, we found that the time the page changed was distributed roughly uniformly over the period we predicted the page would remain unchanged. This means that to reduce the stale data rate by half, we would have to reduce the time a page is valid in the cache by half and that, in turn, would reduce the cache hit ratio by half.

Another viewpoint is that most users look at a given page only once. This limits good caching opportunities to pages that attract many users, but for caching to be useful, these pages must not change too frequently. Unfortunately, our relay's 20 most popular pages include "What's new" pages, the home page for a local newspaper, and weather reports; all of which change regularly.

There are whole classes of requests that can't cache successfully: requests that include cursor positions over images, database lookups, filled-in forms that are being sent to a remote server, and other requests that are created by the user. These requests are unlikely to ever be repeated and are not cached. In addition, the relay doesn't cache HTTP requests that include "authorization", or "charge-to" fields.

## 9 Characterization of our work load

[We are still gathering data for this section. We will characterize the work load on each relay and see how the work load influences cache performance.] Here is a very rough summary of the workload of our first relay:

| HTTP requests to Gopher requests: |
|---|
| 11,314 to 3,333 (77%) |

| Average file sizes (bytes) | |
|---|---|
| by request: | 13,781 |
| in cache: | 17,999 |
| http vs. gopher: | 17,646 vs. 19,196 (in cache) |
| hits vs. misses: | 11,199 vs. 15,558 (by requests) |

| Size distribution (files in the cache) | | | | |
|---|---|---|---|---|
| 0 | - | 1KB | 5,290 files | 2.78 Mbytes |
| 1KB | - | 5KB | 8,843 files | 21.25 Mbytes |
| 5KB | - | 10KB | 2,711 files | 19.20 Mbytes |
| 10KB | - | 50KB | 3,052 files | 67.67 Mbytes |
| 50KB | - | 100KB | 722 files | 51.67 Mbytes |
| 100KB | - | 500KB | 618 files | 118.55 Mbytes |
| 500k | - | 1MB | 86 files | 58.91 Mbytes |
| 1MB | - | 5MB | 45 files | 79.92 Mbytes |
| 5MB | - | 10MB | 1 files | 5.26 Mbytes |

User requests are log-linear. The ratio of the number of requests made by the i'th most frequent user to the (i+1)'st most frequent user is approximately 1.02. See Figure 6.

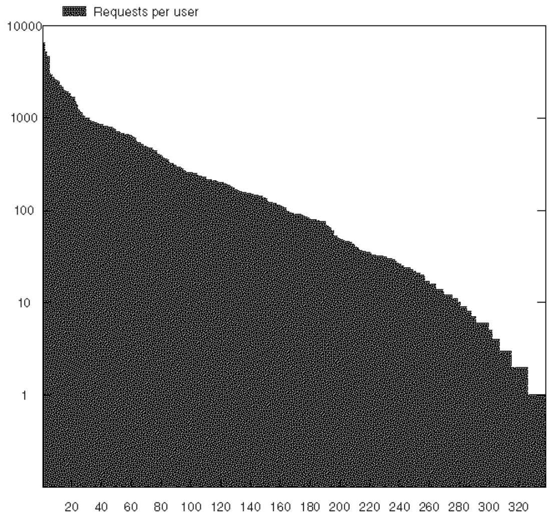



Figure 6: The number of requests per user, sorted in order of the number of requests. The horizontal axis gives the i'th most active user. The vertical axis gives the log of the number of requests made by the user.

## 10 Other caching efforts

There are a number of other efforts to provide caching to the Web. Although there are no published papers, there have been various announcements and discussions about caching in news groups and on the Web itself:

The caching server from CERN, which can run in a gateway mode where it relays requests to external servers and caches the responses. (http://info.cern.ch/hypertext/-WWW/Daemon/Status.html)

The Unix Hensa project at the University of Kent at Canterbury runs a cache on their archive and on certain entry points in the Web. (http://unix.hensa.ac.uk)

On www-talk, Guenther Fischer announced an experimental caching server at the Technical University of Chemnitz. (http://www.tu-chemnitz.de)

On comp.infosystems.www, Gertjan van Oosten announced a HTTP caching proxy CGI script. (gvoosten@isosa1.estec.esa.nl)

## 11 Open Issues

We can improve the cache's performance by speculatively retrieving pages from remote servers before they are requested. This involves automatically refreshing some pages in the cache when they expire, or pre-fetching pages that might be requested soon. How do the costs of additional network load compare to the improved performance of the cache?

We need to develop good algorithms for managing the cache when it gets full. How big should the cache be? How small can it be and still be effective?

Is the distribution of page requests really a Zipf distribution? If not, what kind of distribution does it have? As the Web evolves, will the distribution of requests change? Will better search and navigational facilities affect caching? How will the distribution change when the Web is 10 times bigger? 1000 times bigger?

Is our use of the Web peculiar in some way? Do strategies that work for our cache apply to other caches? Would a much larger or much smaller user population give different results?

How do legal issues like access control, copyrights, pornography, and export licensing affect a caching relay?

## 12 Conclusions

Our experience shows that caching provides added value for little cost. It decreases latency for users accessing common pages and does not add significant overhead for users accessing new pages. For sites with poor network connectivity, the gains of caching are more significant.

## References

[1] RFC 959: File Transfer Protocol. http://info.cern.ch/-hypertext/WWW/Protocols/rfc959/Overview.html

[2] Gopher protocol. http://info.cern.ch/-hypertext/Gopher/gopher_protocol/protocol.rtf

[3] HTTP: A protocol for networked information. http://info.cern.ch/hypertext/WWW/Protocols/-HTTP/HTTP2.html

[4] RFC 1341: Mutipurpose Internet Mail Extensions http://info.cern.ch/hypertext/WWW/Protocols/-rfc1341/0_TableOfContents.html

[5] Mosaic. http://www.ncsa.uiuc.edu/SDG/-Software/Mosaic/NCSAMosaicHome.html

[6] RFC 977: Network News Transfer Protocol http://info.cern.ch/hypertext/WWW/Protocols/-rfc977/rfc977.html

[7] Socks tar file. ftp://s1.gov/pub/firewalls/socks.tar.Z

[8] Uniform Resource Locators. http://info.cern.ch/hypertext/WWW/Addressing/-URL/Overview.html

[9] Wide Area Information System http://info.cern.ch/hypertext/-Products/WAIS/Overview.html

[10] The World Wide Web Initiative: The Project http://info.cern.ch/-hypertext/WWW/TheProject.html

[11] Donald Knuth. *The Art of Computer Programming, Volume 3. Sorting and Searching* Page 397 Addison-Wesley, 1973