Performance Issues of Enterprise Level Web Proxies

Carlos Maltzahn and Kathy J. Richardson
Digital Equipment Corporation
Network Systems Laboratory
Palo Alto, CA
carlosm@cs.colorado.edu, kjr@pa.dec.com

Dirk Grunwald
University of Colorado
Department of Computer Science
Boulder, CO
grunwald@cs.colorado.edu

Abstract

Enterprise level web proxies relay world-wide web traffic between private networks and the Internet. They improve security, save network bandwidth, and reduce network latency. While the performance of web proxies has been analyzed based on synthetic workloads, little is known about their performance on real workloads. In this paper we present a study of two web proxies (CERN and Squid) executing real workloads on Digital's Palo Alto Gateway. We demonstrate that the simple CERN proxy architecture outperforms all but the latest version of Squid and continues to outperform cacheless configurations. For the measured load levels the Squid proxy used at least as many CPU, memory, and disk resources as CERN, in some configurations significantly more resources. At higher load levels the resource utilization requirements will cross and Squid will be the one using fewer resources. Lastly we found that cache hit rates of around 30% had very little effect on the requests service time.

1 Introduction

Enterprise web proxies service large user communities, typically large corporations or Internet service providers. They forward HTTP requests from an internal network to the Internet and relay the corresponding responses to the internal network, thus acting as a firewall. Web proxies can be configured to cache relayed responses. This saves network bandwidth and reduces average response latency.

For our work we must define the main workload characteristics and basic performance requirements for enterprise level web proxies. The main workload characteristics are high traffic volume (currently over one million requests per day), high cost for degenerated proxy service or outage (24 hours a day, seven days a week), and all the uncertainty and error propagation associated with the Internet. Thus, the basic performance requirements of enterprise web proxies are fast crash recovery, scalability under high load, availability, reliability, robustness, and quality of service. Some enterprises may have additional constraints or specific feature requirements, but all require high quality service.

To appear in the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, June 15-18, 1997, Seattle, WA

While the performance of web proxies has been analyzed based on synthetic workloads, little is known about their performance on real workloads. The goal of this research is to understand how the state of the art in web proxy design performs under heavy real workload, how their resource requirements might scale to larger workloads, and to identify promising directions of improvement. In this paper we present the performance of two widely used public domain web proxies under real workload conditions for both caching and cacheless configurations. These web proxies are the web server "httpd" developed at CERN [12], which can also be run as a proxy, and the public domain successor of the Harvest Object Cache [6, 7] called "Squid" [22]. Two versions of Squid are evaluated: Squid 1.0.beta17 and Squid 1.1.5. For the rest of the paper we will refer to the CERN proxy server as *CERN* and to the two Squid proxy versions as *Squid 1.0* and *Squid 1.1*. The proxies are evaluated on their resource requirements (CPU, memory, and disk I/O) and how well there requirements scale with load. The request service time and hit rate are used to evaluate the quality of service each provides.

We chose these proxies because initial tests seemed to indicate that CERN with its simple architecture performs better at high workloads than Squid which was designed for high performance. We suspect that many of Squid's sophisticated architectural features were designed prior to the work on operating system behavior and implementation effects on high Internet server performance [15, 16, 17, 10]. Common web server and web proxy benchmarks do not include realistic network latency profiles [18, 19]. As we will show in this paper, network latency can have a significant impact on a proxy's resource utilization and therefore its performance. Resource utilization is also influenced by the hit rate of a caching proxy, and other environmental factors which increase the network latency.

The most related work is the paper about the design of the

Harvest Object Cache [6]. The authors present a performance analysis based on a simulated load of 10 local clients each requesting in parallel 200 unique objects in random order. Their measurements indicate that Harvest is orders of magnitude faster than CERN. We will complement that paper with measurements on a much higher and non-simulated load.

There have been a number of web traffic characterization [2, 3, 9, 16, 15]. However, all of them focus on web server traces and often at a much lower load level than the workloads we studied. The uncertainty and variance of Internet services impact web servers and proxies in different ways. While web servers only accept connections from clients, proxies additionally generate connections to a multitude of web servers with varying service times and often over slow or poor connections.

In [1] the authors perform application and system level measurements of web server performance. Their measurements are based on sampling and event-driven techniques that resulted in less than 3% overhead. They use a synthetic workload generated according to the WebStone benchmark. Results show that a server saturated by requests spends over 90% of the time in system calls. We complement these results with our results from real workloads.

Recently, SPEC published a standard for web server performance [18]. This benchmark acknowledges the fact that it does not simulate network latency. We will show in this paper that network latency is a crucial factor in server and proxy performance.

In section 2 we present an overview of CERN and Squid architectures. We proceed in section 3 with a description of the methodology of our performance measurements, including a workload characterization. Next, the results section (section 4) presents resource requirements for CERN and Squid under different configurations and compares their quality of service. In section 6 we identify general performance issues based on our results, and conclude the paper with future research.

2 Proxy Architectures

The function of a web proxy is to relay a request in the form of a Uniform Resource Locator (URL)[8] from a client to a server, receive the response of a server and send it back to the client. If the proxy is configured to have a disk or memory cache, the proxy tries to serve a client's request out of its cache and only contacts the server in the case of a cache miss. A cache miss occurs when the object is not in the cache or it has expired. When the request is relayed to a server, the proxy translates the server name contained in the URL into an IP address in order to establish a connection. This usually requires a query to the Domain Name Service (DNS) [13, 14], which is typically implemented on a separate host on the same network as the proxy to service all external host name to IP address mappings for the enterprise.

CERN The CERN proxy forks a new process to handle each request. In caching configurations CERN uses the file system to cache both data (web pages) and proxy meta-data (expiration times, content type). It translates the request into an *object file name* which it derives from the structure of the URL: each URL component is translated into a directory name. The resulting file name is a path through one or more directories. Thus, the length of the path depends on the number of URL components. We call this path without the last component the *URL directory*. The meta-data is stored in a separate file for each URL directory. To find out whether a request can be served from the cache, CERN tries to open the meta-data file in the URL directory. Every component of the URL directory name needs to be resolved. If the meta-data file exists and it lists the object file name as not expired, CERN serves the request from the cache. In any other case, CERN relays the request to the appropriate server and passes the server's response to the client and stores it in its cache. In a cacheless configuration, CERN only relays requests to server and passes responses to clients. Processes are created to serve a single request after which they terminate. Objects are removed from the cache by a separate "garbage collection" process that checks for expired objects and deletes them.

Squid The Squid proxy is the public domain network object cache portion [6] of the Harvest system [5]: "tools to gather, extract, organize, search, cache, and replicate relevant information across the Internets". The architecture was designed to be portable and to overcome performance weaknesses of CERN: It uses its own non-blocking network I/O abstractions built on top of widely available system calls and it avoids forking new processes except for relaying FTP requests. "For efficiency and portability across UNIX-like platforms, the cache implements its own non-blocking disk and network I/O abstractions directly atop a BSD select loop" (section 2.8 in [6]). In managing its own resources, Squid attempts to isolate itself from the operating system. Squid keeps meta-data about the cache contents in main memory. This enables Squid to determine whether it can serve a given request from its cache without accessing the disk. Squid maps URLs to cache object names using "fingerprinting" [20]. The cache has a LRU expiration policy which is activated once the cache size reaches a configurable high mark, and deactivated once the cache size falls below a configurable low mark. Squid also uses main memory to cache objects that are currently in transit, to cache the most recently used objects in a *hot cache*, and to cache error responses which resulted from bad requests. In-transit objects have priority over error responses and *hot cache* objects. Squid implements its own DNS cache and uses a configurable number of "dns server" processes to which it can dispatch non-blocking DNS requests.

The choice of this architecture has some interesting consequences:

- A large number of file descriptors must be managed by a single process,
- Many operating system facilities must be replicated within the proxy,
- Storing the meta-data for each cached object in memory means that main memory utilization grows with the number of objects cached or the proxy cache size. Increasing the cache size requires increasing both disk and main memory.

Squid versions At the time we started our experiments Squid 1.0 was the most recent version available. Half a year later Squid seemed to have matured significantly and we repeated the experiments with Squid 1.1. The most significant differences between Squid 1.0 and 1.1 are the following:

- Squid 1.1 introduces an extra level of directories to keep the individual directory size small. Squid 1.0 had only a single level of directories with a large number of objects which caused the directory objects to grow beyond a file system block. According to Squid developers this slowed down directory searching and caused significantly more disk traffic due to directory operations;

- Squid 1.1 switches from a Time-To-Live based expiration model to a Refresh-Rate model. Expiration dates are no longer assigned to objects when they enter the cache. Instead, the "freshness" of an object is tested at hit time based on the object's age in the cache, it's last modified date and its expiration date (if it exists). The last modified date and expiration date are shipped from the server with the original object. If an object is not fresh, or "stale", the proxy asks the server whether the object has been modified. Thus, objects are not purged from the cache when they expire. In practice the only difference between the two schemes is that the Refresh-Rate model keeps objects after they have *expired* and is able to use the object if the server reports that the object has not been modified.

3 Methodology

3.1 Workload

Our workload is taken from the web traffic at Digital Equipment Corporation's (Digital) Palo Alto Gateway which has a web proxy located at and managed by the Network Systems Lab at Palo Alto, CA. The gateway relays web communication between much of Digital's intranet and the Internet. A large fraction of the North American and Asian sites use this gateway. A measurement infrastructure allows us to collect system and application performance data on a daily basis in a fully automated fashion. We have collected almost a year's worth of data during the deployment of various commercial and non-commercial web proxies[1].

Real world workloads are by definition *not repeatable*, and contain a multitude of errors. The chosen workload samples strive to represent *best case* workload patterns because it is easier to find comparable best workloads than comparable failure modes. For the analysis presented in this paper we decided to select workloads based on the following criteria:

- The load occurred during a business day. We are interested in high load testing - business days exhibit a two to three time higher load than weekends.
- The proxy under test delivers 24 hours of uninterrupted service. This was a surprisingly limiting criterion: especially in a caching configurations the proxies were unreliable.
- Little detectable anomalous network behavior. We used the the length of the system network tcp queue for pending connections to the Internet (SYN_SENT queue) and the access level for indicators of network problems. Unusually large SYN_SENT queues or unusually low access levels are generally caused by Internet service failures.
- The Domain Name Service (DNS) average service time is reasonably short for the entire 24 hour period. Occasionally, the DNS degenerates, which increases proxy service time and skews our measurements.

[1]Colleagues have collected proxy request traces that are now available for public use [11]. The current traces contain data taken between 29 August 1996 and 22 September 1996. This is a total of 24,477,674 references.

Selecting workloads based on the above criteria results in a selection which represents best cases instead of average cases. The curves of the selected workloads are shown in Figure 1. Each workload is taken from a 24 hour time period. The selected workloads are from days which span almost six months over which the number of daily requests almost doubled.

3.2 Measurement Framework

The proxy experiments used two dedicated Digital Alpha Station 250 4/266 machines with 512 MB of main memory and 8 GB of proxy cache disk space. DNS round-robin split the load between the two to insure that each had more than sufficient hardware resources. An additional process logged system statistics every 15 minutes; once a day all logs were shipped to other machines for log processing and archiving.

A set of standard Unix tools ran every 15 minutes to measure proxy resource consumption. Among other things, these tools provided information about the CPU idle time (iostat), the memory allocated by processes (ps) and by the network (netstat), and the total number of disk accesses per second (iostat). Each of these measurements are snapshots and do not summarize the activity of the whole 15 minutes.

This sampling approach allows us to continuously monitor the overall system behavior, collecting data for months on end. From this we know the baseline performance of the system, the expected load for a given day and time, and have the ability to detect network problems that are unrelated to proxy yet affect its performance or the service seen by the clients. By monitoring the length of the tcp (SYN_SENT queue) we can detect quality of service failures to portions of the Internet. Monitoring the length of the tcp (SYN_RCVD queue) we can detect failures on the corporate Intranet.

Snapshot measures provide an accurate measure of system behavior at a single point in time; this preserves details that might be lost when aggregating the performance over large periods of time. Collecting sufficient samples over long periods of time produces a full range of expected behavior and errors. The drawback of this technique is that it is not possible to tightly correlate events. This would be difficult even with precisely correlated measurements because proxy service is a pipeline within which arbitrary delay and queueing occur. Thus, the request rate is decoupled from the serviced request rate.

The regular logs of CERN and Squid did not give us precise information about the duration of the proxy service times. To obtain more accurate data we instrumented CERN and Squid 1.0. The service time duration is the time from receiving a request from a client to terminating the connection to a client, effectively the time that the end user waits for a request to be completed. We summarize the service time durations and the number of requests serviced per second (rps) every 15 minutes taking the the mean or distribution of all measurement points.

4 Results

Two different configurations were evaluated for each of the three proxies: a proxy without cache and a proxy with 8 GB disk space for caching. For the caching configurations we set the time-to-live or refresh-rate to 50% of the time since last modification.

For cache configurations the performance is also dependent on cache hit rate. CERN's hit rate was 35%, Squid 1.0 was 24%, and

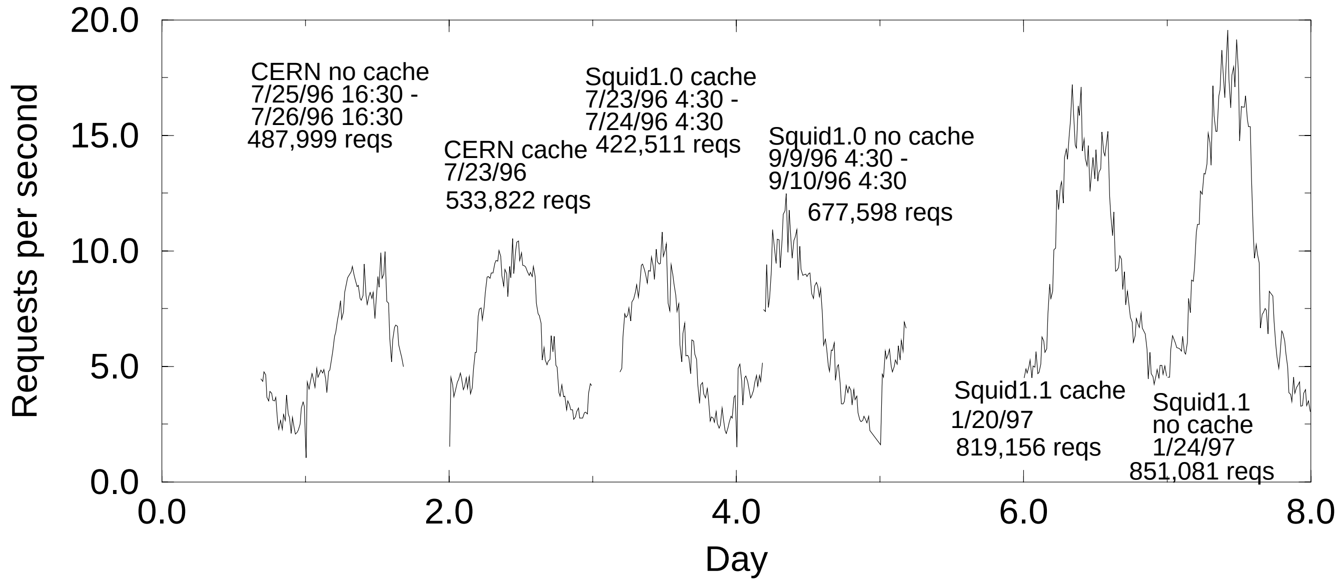

Figure 1: The load profiles of days selected for our experiments - for each proxy we measured one day with and one day without caching. The selected days span almost 6 months over which the number of requests serviced per day doubled, from a low of 422,511 requests to a high of 851,081 requests/day. The CERN cache hit rate was 35% Squid 1.0 was 24%, and the Squid 1.1 hit rate was 28%. The highest load during a day occurs between 10 and 11 a.m. and the low load period is between 4:30 p.m. and 4:30 a.m.

Squid 1.1 had a hit rate of 28%. Different hit rates have an impact on average service times and resource utilization. We will discuss these hit rate differences in Section 4.3.

4.1 Basic Enterprise Proxy Criteria

Recall that to be considered an *enterprise proxy*, a proxy must be capable of operating twenty-four hours a day under high load, while being resilient to intranet and internet failures. It must have fast failure recovery, scalability to handle load peaks, availability, reliability, robustness, and provide a high quality of service. None of the three proxies meet these requirements.

All the proxy cache-configurations require constant attention. With a cache Squid tends to lock up and quit serving requests for no apparent reason. CERN's cache garbage collection is much slower than its fill rate, so that cache can reach capacity, at which point both the proxy and the system lock up. Early versions of Squid had long restart times during which they were unavailable. For the caching configurations the mean-time between failures starting with a clean cache was on the order of 3 days; restarting the cache after a failure resulted in mean-time between failures of under a day.

The proxies have no mechanisms to protection themselves from Internet failures, like limiting the number of outstanding connections to sub-nets that have poor response times.

We know of no publicly available proxy (commercial or public domain) that meets these criteria. There are enterprises, such as large Internet Service Providers, that successfully run caching web proxies based on these CERN and Squid proxies. As the body of knowledge and code base matures, *enterprise proxies* will emerge. For this reason we continue to evaluate the potential and relative performance of available proxies.

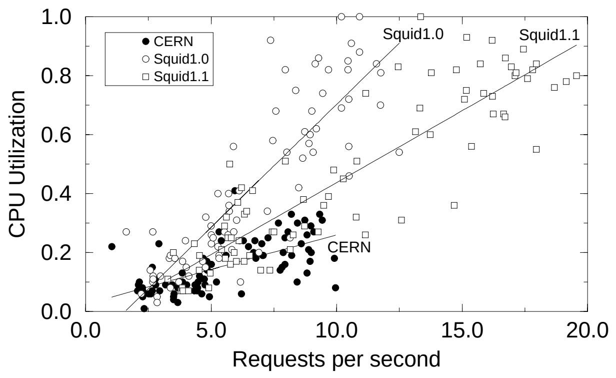

Figure 2: CPU utilization for cacheless configurations. Each data point represents one sample. The lines are a linear approximation for the data, and are meant to visually help group the points around an axis - not to model the data.

4.2 Resource Requirements

CPU utilization Proxy CPU requirements determine the basic load that a workstation or server can handle. If the CPU requirements scale linearly with load, then CPU load characterization can establish server requirements for expected workloads. Understanding the components of the CPU requirements allow one to predict the CPU requirements on other systems or other configurations.

The CPU utilization of CERN and Squid are shown in Figures 2 and 3. In these figures the CPU usage is not as tightly correlated with the workload request per second service rate as one would hope. For the cacheless configurations, Figure 3, the CPU utilization is erratic and it is impossible to draw any conclusions. One explanation, is that a multitude of environmental factors ef-

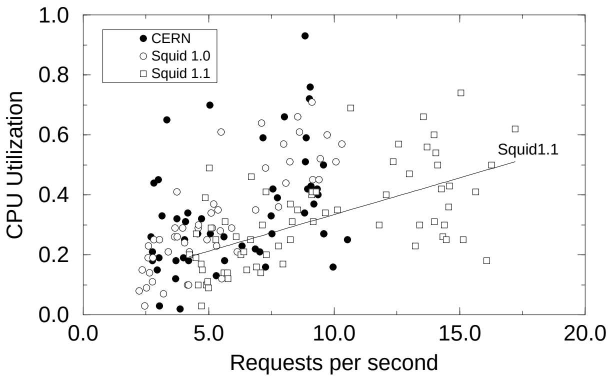

Figure 3: CPU utilization for proxy cache configurations. The Squid 1.0 and CERN results are too scattered to for an approximation.

fect average Internet request service time which in turn effects the amount of simultaneous state that must be maintained and managed by the proxy and the operating system. The cache adds additional variation to the request processing and state requirements. The amount of simultaneous state combined with the request process rate determines the total CPU requirements.

For the cacheless case, Figure 2, CERN requires significantly fewer CPU cycles than either Squid version, except in the lowest load regimes. The CERN proxy scales well with a load to CPU scaling factor of about 2.5%/rps (requests per second). Squid 1.0 has the worst scaling factor somewhere between 8.75%/rps (requests per second) and 5.9%/rps. One reason that Squid uses excessive CPU in the cacheless case is that it performs much of the same in memory cache maintenance, regardless of the cache size (zero in this case).

Squid 1.1 uses considerably fewer CPU cycles than Squid 1.0. In the cacheless case, Squid 1.1 still takes more cycles than the CERN proxy, but with cache it outperforms CERN with caching. Surprisingly, the Squid 1.1 caching configuration outperforms the Squid 1.1 cacheless configuration.

The differences in CPU performance can be explained by examining the two proxy architectures and the system cost of various operations which all proxies rely on heavily. CERN forks a new process for each request, and keeps no meta-data or state internally in the process (the cache is implemented entirely on disk). Each process has very little state to scan or to pass into the kernel for network related system calls. Forking a process for each request incurs a large overhead which is eliminated in the Squid architecture. The Squid architecture eliminates process forking; it implements asynchronous I/O within a single thread and stores all the cache meta-data in main memory in order to improve cache response time. This results in additional CPU cycles to manage all the network connections in a single process, to process much more state for network system calls, and to manage the in-memory meta-data.

With the Digital Continuous Profiling Infrastructure (DCPI[2]) [4] we compared the CPU usage profile of CERN and Squid 1.1. For the DCPI results, we collected two sets of samples, of 20 minutes for each configuration. The two cacheless configurations were run in parallel, and the two cached configurations were run in parallel. This eliminates differences in external network behavior between the samples. Prior to the final DCPI run, many other samples were taken; the cycles/request varied somewhat but the conclusions were consistent reguardless of load or time of day. Furthermore, these results are consistent with the measurements shown in Figures 2 and 3 and with the related work in [1]. The latter demonstrates that a vast majority of the CPU time is spent in kernel routines and implies that a proxy's major function is to manage network connections and pass data.

[2]DCPI: The Digital Continuous Profiling Infrastructure for Digital Alpha platforms permits continuous low-overhead profiling of entire systems, including the kernel, user programs, drivers, and shared libraries. The system is efficient enough that it can be left running all the time, allowing it to be used to drive on-line profile-based optimizations for production systems. The Continuous Profiling Infrastructure maintains a database of profile information that is incrementally updated for every executable image that runs. A suite of profile analysis tools analyzes the profile information at various levels.

	CERN		Squid 1.1	
	Cacheless	Cache	Cacheless	Cache
Proxy	16.0	19.8	22.6	9.4
Kernel	87.9	128.6	89.7	55.4
Shlib	1.5	2.9	13.4	10.2
Total	105.4	151.3	125.6	75.0

Table 1: DCPI results - measured in 100,000 cpr (number of cycles required to process a request). The configurations vary between 7.5 Million cpr to 15.1 Million cpr.

Table 1 shows the profiling results normalized to the number of cycles required to process a request (cpr) (kernel idle cycles were filtered out). The most obvious result is that the native proxy executes only 12% – 18% of the cycles required to process a request. CERN relies directly on the kernel to manage resources while Squid manages many of its own resources via the shared libraries.

For CERN, the differences between the cacheless and cache configuration are predictable. The cache configuration requires additional CPU to manage and lookup both data and meta-data in the cache. The proxy translates URLs into file system calls; the kernel processes the additional file system calls and the associated memory managements; shared libraries support miscellaneous proxy cache lookup and time-stamp evaluations.

At first glance the Squid results make little sense. Cacheless Squid 1.1 requires almost twice as many processor cycles as it does with a cache. The reason for this is many fold. First, the cache is highly integrated into the Squid architecture, so a cacheless configuration performs all the same work that a cache configuration would except for writing data to disk. Since the cache configuration has to fetch fewer objects from servers (28% fewer for the measured workload) it averages less work per client request. Secondly, Squid manages its own memory space, allocating and freeing memory as needed (much of the shared library contribution deals with memory management). Without a cache it seems to repeatedly free and reallocate buffer space to process requests. Lastly, the per connection computational cost of the Squid network polling scheme is a function of the number of open connections.

Table 2 breaks out the *process*, *memory*, and *network* components from the kernel cycles. This shows the relative importance of the architectural choices in each proxy configuration. CERN has high process management costs but inexpensive network management costs because each process has a single connection on which it can block. Squid has high network costs that increase super-linearly with the network load, additional memory management costs, and inexpensive process management costs. This is probably exagger-

The tools used for this analysis show the fraction of cpu cycles spent executing the kernel and each user program procedure.

	CERN		Squid 1.1	
	Cacheless	Cache	Cacheless	Cache
Process Mgnt	24.3	30.0	5.3	6.7
Memory Mgnt	10.0	14.8	22.0	11.9
Network Mgnt	2.6	3.5	28.1	13.1
Other Kernel	51.0	80.3	34.3	23.7
Total Kernel	87.9	128.6	89.7	55.4

Table 2: Process, Memory and Network Management contributions to the kernel cycles per request-processed (100,000 cpr). Memory management functions primarily associated with process spawning were included in the Process Management category. (All relevant kernel procedures accounting for at least 0.33% of the cycles were summed into the results.)

ated by higher network polling rate in the cacheless configuration.

Over the measured range of operation, CERN clearly requires less CPU in the cacheless configuration; it is a very inexpensive firewall proxy. Squid 1.1 requires the least CPU in the cache scenario. The network management scheme used in the Squid architecture passes state for a large number of connections back and forth on kernel system calls. This makes it hard to predict how it will scale beyond the measured workload range.

Because most of the processing time is spent in the kernel, a proxy implementation is not operating system independent. The CPU requirements are dependent on the relative use of each operating system facility and the relative cost of each operating system function used by the proxy. The cost of each operating system function will vary across vendors and across releases. Proxies should not be considered to be operating system independent.

Memory utilization Figures 4 and 5 show the overall memory usage for the proxy configurations. The total memory usage is calculated by summing the residence memory size for each process running on the system. Of this the kernel process typically uses 23M Bytes of physical memory, and the daemons, monitoring utilities, and proxy related utilities, typically use another 5M Bytes.

Squid's memory utilization is largely load independent. This is due to Squid's main memory cache management. Squid maintains its own memory pool which it pre-allocates and extends when needed.

This memory pool includes the Squid process state, disk cache meta-data, and a memory cache. The core Squid process uses about 35M Bytes; this grows with the number of simultaneous connections. At peak loads, the memory pool is extended to accommodate additional process state. The disk cache meta-data uses roughly 10M Bytes for each Gigabyte of proxy disk cache in use. (The cache *high water mark* was set to 80%, so the cache was typically just over 6G Bytes.) The memory cache is used for in-transit objects, meta-data, and hot cache objects; its maximum size was configured to 128M Bytes for the experiments. Under peak loads Squid is supposed to remove hot cache objects from the memory cache to make additional room for the in-transit objects. Squid also requires 3 to 5M Bytes for DNS server and Ftp server processes. Once memory is allocated it is permanently added to the memory pool, unless there are insufficient system resources. If the memory pool pages begin to swap, Squid will reduce the memory pool size if possible to avoid page faults. (We did not evaluate this; all experiments had 512M Bytes of physical memory to avoid limited memory effects).

The memory usage for Squid 1.1. with a cache, shown in Figure 5, matches the predicted usage: 23+5M Bytes for the kernel and daemon processes, 35M Bytes for core proxy, 5M Bytes for DNS and Ftp servers, 128M Bytes for the memory cache, and 60M Bytes for the proxy cache meta-data, totaling 256M Bytes. Without a cache, Squid should use about 196M Bytes. Figure 4 shows both Squid versions use only 125-150M Bytes of memory. Since there are no cacheable objects Squid probably does not allocate the entire 128M Bytes for the memory cache; it only requires space for in-transit objects.

The two distinct memory usage bands seen in Figure 4 for Squid 1.1 are due to a memory pool extension during load peak time: The lower level points (125 M Bytes) represent measurements taken before the load peak, and the higher level points (150 M Bytes) are taken after the load peak. With a cache, Squid 1.0 uses only slightly more memory than the cacheless configuration. This is probably due to a bug in its memory management [21].

The CERN proxy memory usage is entirely load dependent. Other system components, such as the operating system still require independent memory. Each CERN process uses 280K Bytes. Higher loads result in a larger number of simultaneous processes, which require more memory. Slower Internet response increases the average life time of a process which increases the number of simultaneous processes and the corresponding memory requirements. For the cacheless CERN configuration, there were about 145 processes for a load of 10 requests per second, and for the cache configuration about 125 processes. Faster service times for cache hits translate into shorter process lifetimes and fewer simultaneous memory consuming processes for the same load. For this reason, a caching CERN uses less memory than a cacheless CERN for the same load (comparing Figures 4 and 5).

Although we report load as the number of requests per second, memory usage is actually related to the number of simultaneous connections the proxy must support. By carefully selecting workloads from days with few Internet service problems, the requests/second metric is relatively proportional to the amount of state required for our site. A site with less connectivity will see higher request service times, which requires more simultaneous processes/threads/state to support the same load. For Squid the process state requirements are minimal in comparison with the meta-data and memory cache usage. For CERN the process state directly determines its memory usage.

At measured load levels CERN requires less memory than Squid but its memory utilization is linearly load dependent. At 20 requests/second all three cacheless proxies should require about the same amount of memory. For the cache case CERN and Squid 1.1 probably will not converge until about 30 requests/second. Although they will use identical amounts of memory, Squid has a single process space which reduces replication, and will allow it to use a larger portion of its memory for meta-data and in-memory caching to speed up request processing. CERN is also more susceptible to external network/Internet problems. Its memory usage is directly related to Internet performance; poor Internet performance could increase the memory requirements and cause paging.

Disk I/O utilization Figure 6 shows the disk utilization for the three caching proxy configurations. For the cacheless configurations, disk use is negligible. Disk utilization for the three caching proxies is remarkably similar. The fact that CERN's simple architecture of directly accessing the file system on every request works

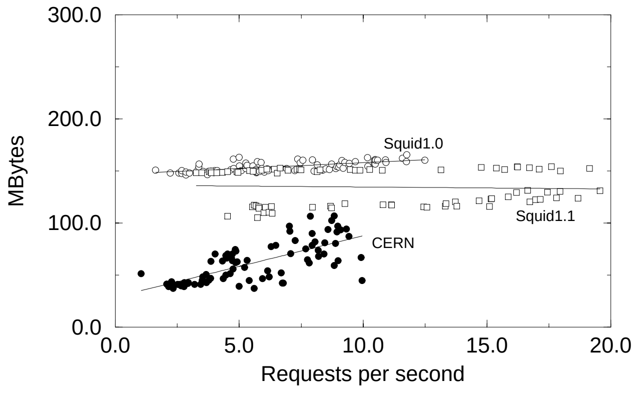

Figure 4: Overall memory utilization - Cacheless configurations. Squid's memory management pools memory while CERN allocates and releases memory with each request processing. Squid occasionally extends its memory pool at peak load without releasing it again.

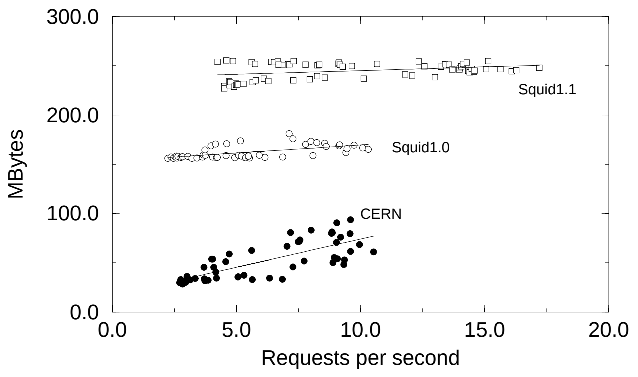

Figure 5: Overall memory utilization - Cache configurations. The memory utilization of a caching CERN is slightly lower than of a cacheless CERN because it translates hit rate into fewer memory consuming processes. Squid's memory utilization is much higher because of its meta-data approach and therefore depends on the cache size and less on the load.

almost as well as the Squid meta-data approach could indicate that the file system's caching of directory path name translations works well even on large working sets: as described in Section 2 CERN accesses the file system to see whether a requested object is in the proxy cache. In the case of a proxy cache miss, the corresponding path name translation is in the file system cache when CERN subsequently writes the requested object to the proxy cache. Squid does not access the file system to find out whether a requested object is in the proxy cache. But it does access the file system to retrieve a proxy-cached object in case of a proxy cache hit or to write a new object into the proxy cache in case of a proxy cache miss. In either case the corresponding path name translation is not reused. Furthermore, CERN stores *adjacent objects*, i.e., objects with URLs that only differ in their last path component, in the same leaf directory. One access to such a leaf directory would bring all adjacent objects into the file system's data cache. Assuming that objects with adjacent URLs are likely to be co-referenced, CERN makes better use of the file system data cache. Squid on the other hand uses fingerprinting which does not preserve object adjacency in the proxy cache structure. We are currently investigating file system performance under various proxy cache structures.

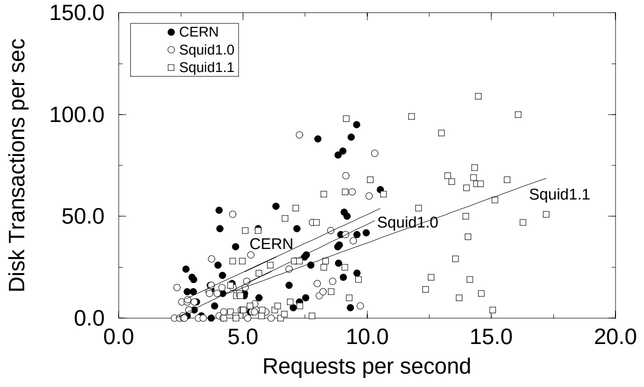

Figure 6: Disk I/O utilization for caching proxy configurations.

4.3 Quality of Service

We measure the quality of service of a proxy by its service time. The service time of a proxy is the time it takes a proxy to successfully complete a request from a client. Shorter service times indicate a higher quality of service. Figure 9 shows the 25th-, median, and 75th-percentile service times for all the configurations. Percentiles provide a meaningful way to evaluate the aggregate service time and the response provided to the client. Percentiles show the typical response and filter out anomalous cases, timeout errors, and long file transfers. All CERN and Squid 1.1 configurations deliver similar service. Half of all requests are served to the client in under 0.5 seconds. The CERN cacheless configuration lags the other three a bit for the 75th-percentile service, but still provides adequate service. The newer version of Squid clearly performs better than its predecessor, Squid 1.0. The remainder of this section will only evaluate Squid 1.1 and CERN.

It is also important the service time not vary with load. If the service time increases with load, the proxy is incapable of handling the load, or scaling to a higher load. We already saw that there were adequate system resources for an increased load. Figure 10 shows the 75th-percentile service time correlated with the requests/second load. There is no degradation in serve time as the load increases. This is true for all the percentile measures.

For a caching proxy configuration, the service times include both hits and misses. The distribution for hit service times is considerably shorter than that of the miss service times. To evaluate the impact of of hit rate on service, we plotted the hit rate with the requests/second load. The hit rate was constant across load. The CERN cache hit rate was 35% Squid 1.0 was 24%, and the Squid 1.1 hit rate was 28%. By comparing the service time distributions of caching and cacheless configurations we can quantify the contribution of caching to the quality of service. In a cacheless proxy configuration, service times consist only of miss times.

Figure 7 shows that in almost 90% of all cases the service time difference between a caching and a cacheless configuration is less than a second. Thus, service times are not much improved by caching. This is especially true for Squid 1.1. CERN's service time is more sensitive to caching than other proxies. We conclude from this that caching is not as important to service times as other aspects of proxy architectures. Figure 9 illustrates this more clearly: the various percentile service times for CERN and Squid 1.1 in a cacheless configuration do not differ significantly from the corresponding caching configurations. However, across different architectures, the service times are in some cases very different.

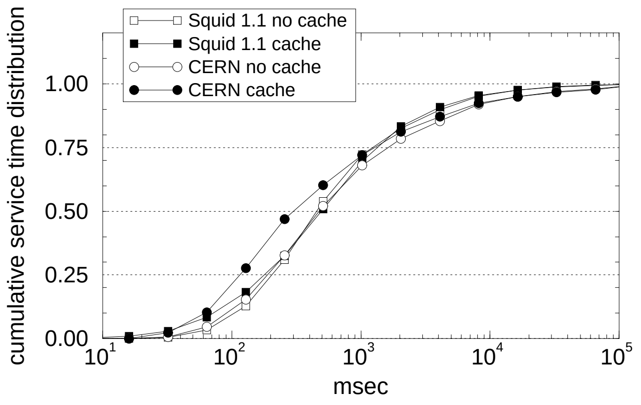

Figure 7: The cumulative service time distribution - the distributions for caching and cacheless configurations are similar: caching proxies do not much improve service times over cacheless proxies. In spite of their very different architectures, CERN and Squid deliver comparable service times in both caching and cacheless configurations.

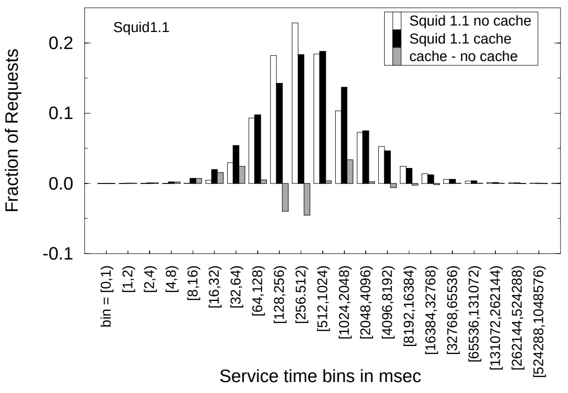

Figure 8: The service time distributions of caching and cacheless Squid 1.1 - The caching configuration "flattens" the service time distribution but does not significantly improve it. The greater number of 1 to 2 seconds service times could represent improvements over cache misses that take 0.5 to 1 minute. But they could also represent slow-downs of services which take only 100 - 500 ms with the cacheless Squid.

5 Discussion

In the light of Squid's sophisticated architecture we found the above results surprising. Squid and its predecessor, the Harvest Object Cache are perceived as at least an order of magnitude faster than CERN [6, 23]. We found that the service times of CERN and Squid are about the same. The load used in the performance analysis of the Harvest Object Cache [6] is very small compared to our load. The Harvest Object Cache's architecture addresses performance issues that are visible at low load, such as the elimination of context switches and the introduction of the DNS cache. These features should also significantly improve performance under high load. However, our results do not confirm this. Squid implements a number of features that are supposed to enhance performance. Some of these features might not increase performance as expected or they might even cancel performance gains of other features. The result is a combined performance that is not much different from CERN's performance.

With heavy real workload external network factors such as network latency become more important. It is therefore important to isolate a proxy from the network as much as possible. DNS caching is a good start: on a day with good DNS performance, DNS caching saves 250 ms on the amount of time each request is open. While a user will probably not notice savings of 250 ms per request, it reduces the number of processes or threads needed for a workload. However, the DNS server occasionally has sever performance failures if its cache table exceeds the physical memory size. In this case, a DNS lookup might take several seconds; Squid's DNS cache services many of the translations, reducing the impact on overall performance.

Although hit rate is typically seen as an important factor for network latency and bandwidth savings our results show it has a much more profound effect on reducing the resource utilization.

The following anecdote during our measurements also illustrates the importance of real workload as opposed to artificial load: as we mentioned earlier we observed an extreme two hour peak of 30K requests per 15 minutes during a day at which we happened to monitor Squid. During these two hours the average net fetch time and the average error time dropped considerably and Squid used less resources than we expected. After studying the logs we found that the load had been generated by a local web robot repeatedly accessing a local web server through the proxy. All of these connections were fast and therefore did not consume many resources. This short period made Squid 1.0 appear as though it could easily scale up to 30K per 15 minutes, whereas our results indicate that this is not the case. A benchmark such as the SPECweb96 would test a proxy under conditions that are similar to this incident.

6 Conclusions

Implementation is at least as an important factor in performance as architecture. Squid's sophisticated architecture should significantly improve performance under high load. However, our results do not confirm this and some of Squid's features are often costly to implement. For instance, Squid uses the CPU cycles it saved by not forking processes to implement memory management and non-blocking network communication. CERN's architecture is inherently inefficient, but manages to efficiently use underlying operating systems constructs. As a result CERN has comparable performance.

With cache hit rates of around 30% we were unable to see a significant difference in the service time profiles between caching and cacheless proxy configurations. Caching might have a larger impact on service times at sites with less available bandwidth than our test site, Although hit rate is typically seen as an important factor for network latency and bandwidth savings our results show it has a much more profound effect on reducing the resource utilization.

CERN uses memory for process state such that its memory utilization grows linearly with the number of processes that are required to support a given request load. Squid keeps global cache and process state in main memory. This state is largely independent of load. The indifference point of memory utilization of CERN and

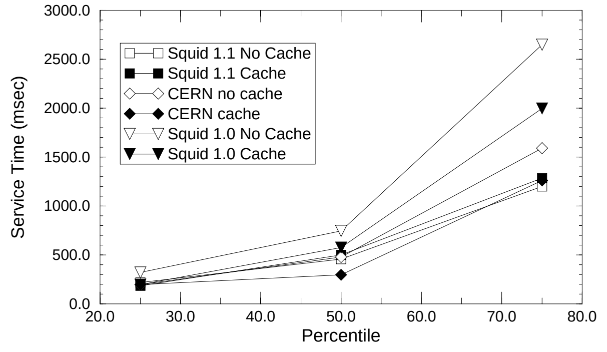

Figure 9: 25th-, median, and 75th-percentile service times in ms for caching and cacheless proxies - Squid 1.1 has the shortest service times among the cacheless proxies, while caching CERN's service times win over the other caching proxies.

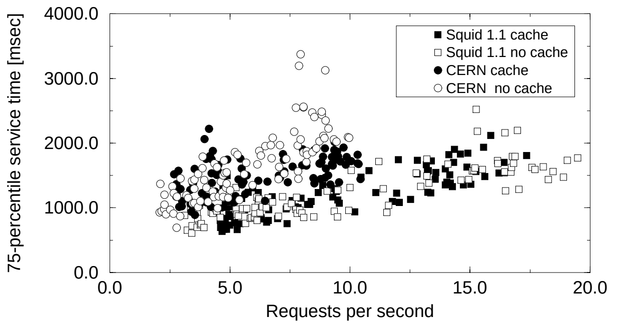

Figure 10: Load against the 75-percentile service time (each point represents a 15 minute time period) - All proxy configurations have a stable service time over the measured load spectrum. Squid 1.1's 75-percentile service times are slightly better than CERN's.

Squid is around 20 to 25 requests per second. CERN requires less memory than Squid at a load below 20 requests per second while Squid is likely to require less memory than CERN at a load above 25 requests per second. Poor network connectivity is likely to lower the indifference point.

Although Squid has many features designed to reduce disk traffic, our measurements did not show any discernable difference between the two architectures. It is likely that the CERN access patterns map very well to the file system caching strategy, and the operating system effectively eliminates many of the potential CERN disk accesses.

We were quite disappointed by the overall robustness of the proxies we tested, especially in the caching configuration. We were unable to run any cache configuration for more than three days without a catastrophic failure that interupted service and required a full manual proxy restart.

We are in the process of investigating various ways to insulate a proxy from network performance. We are also researching the interactions of various proxies with the underlying operating system.

7 Acknowledgments

We would like to thank Jennifer Anderson, Zulah Eckert, and Kirk Johnson for giving a lot of very useful advice and for reviewing early drafts of this paper. Many thanks also to the anonymous reviewers who provided very constructive critique. A special thank you goes out to all the Digital Employees forced to endure yet another unstable proxy configuration experiment.

References

[1] Jussara Almeida, Virgilio Almeida, and David Yates. Measuring the Behavior of a World-Wide Web Server. Technical Report CS 96-025, Boston University, October 29 1996.

[2] Virgilio Almeida, Azer Bestavros, Mark Crovella, , and Adriana de Oliveira. Characterizing Reference Locality in the WWW. In *IEEE PDIS'96: The International Conference in Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996. IEEE.

[3] Martin F. Arlitt and Carey L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *ACM Sigmetrics '96*, pages 126–137, Philadelphia, PA, May 23-26 1996. ACM Sigmetrics, ACM.

[4] Lance Berc, Sanjay Ghemawat, Monika Henzinger, Shun-Tak Leung, Mitch Lichtenberg, Dick Sites, Mark Vandevoorde, Carl Waldspurger, and Bill Weihl. DIGITAL Continuous Profiling Infrastructure. Available on the World Wide Web at *http://www.research.digital.com/SRC/dcpi/papers/osdi96-wip.html*, October 1996.

[5] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Man ber, Michael F. Schwartz, and Duane P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder , CO, August 1994 (revised March 1995) 1994.

[6] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. In *1996 USENIX Technical Conference*, San Diego, CA, January 1996. USENIX.

[7] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Duane Wessels, Mike F. Schwartz, and Erhyuan Tsai. The Harvest Cache and Httpd-Accelerator. Available on the World Wide Web at *http://excalibur.usc.edu/*, July 1995.

[8] Dan Connolly and Tim Berners-Lee. Names and Addresses, URIs, URLs, URNs, URCs. Available on the World Wide Web at *http://www.w3.org/pub/WWW/Addressing/*, December 1990.

[9] M. Crovella and A. Bestavros. Self-similarity in World-Wide Web Traffic: Evidence and Possible Causes. In *Proc. of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1996, May 1996. ACM.

[10] Peter Druschel and Gaurav Banga. Lazy Reiceiver Processing (LRP): A Network Subsystem Architecture for Server Systems. In *Second Symposium on Operating System Design and Implementation (OSDI 96)*, Seattle, WA, October 1996.

[11] Thomas Kroeger and Jeffrey Mogul. Digital's Web Proxy Traces. Available on the World Wide Web at *ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html*, October 1996.

[12] Ari Luotonen, Henrik Frystyk Nielsen, and Tim Berners-Lee. CERN httpd 3.0A. Available on the World Wide Web at *http://www.w3.org/pub/WWW/Daemon/*, July 15 1996.

[13] P. Mockapetris. Domain Names - Concepts and Facilities. Available on the World Wide Web at *ftp://ftp.internic.net/rfc/rfc1034.txt*, November 1987.

[14] P. Mockapetris. Domain Names - Implementation and Specification. Available on the World Wide Web at *ftp://ftp.internic.net/rfc/rfc1035.txt*, November 1987.

[15] Jeffrey C. Mogul. Network behavior of a busy web server and its clients. Technical Report 95/5, DEC Western Research Laboratory, Palo Alto, CA, October 1995.

[16] Jeffrey C. Mogul. Operating Systems Support for Busy Internet Servers. In *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, page addendum, Orcas Island, Washington, May 1995.

[17] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *1996 Usenix Technical Conference*, pages 99–111, San Diego, CA, January 1996.

[18] Standard Performance Evaluation Corp. (SPEC). SPECweb96 Benchmark. Available on the World Wide Web at *http://www.specbench.org/osg/web96/*, July 1996.

[19] Gene Trent and Mark Sake. WebSTONE: The First Generation in HTTP Server Benchmarking. Available on the World Wide Web at *http://www.sgi.com/Products/WebFORCE/WebStone/*, February 1995.

[20] Neal R. Wagner. Fingerprinting. In *Symposium on Security and Privacy*, pages 18–22, Oakland, CA, 1983. IEEE.

[21] Duane Wessels. Personal communication. during which became clear that Squid 1.0.beta17 has a memory management bug, August 1996.

[22] Duane Wessels. Squid Internet Object Cache. Available on the World Wide Web at *http://squid.nlanr.net/Squid/*, May 1996.

[23] Duane Wessels. SQUID Frequently Asked Questions. Available on the World Wide Web at *http://squid.nlanr.net/Squid/FAQ.html*, January 1997.