Measuring Proxy Performance with the Wisconsin Proxy Benchmark

Jussara Almeida and Pei Cao
Department of Computer Sciences
University of Wisconsin-Madison
jussara,cao@cs.wisc.edu

April 13, 1998

Abstract

As Web proxies become increasingly widespread, there is a critical need to establish a benchmark that can compare the performance of different proxy servers and predict their performance in practice. In this paper, we describe the Wisconsin Proxy Benchmark (WPB) and the performance comparison of four proxy software using the Benchmark. Using the benchmark, we also study the effect of more disk arms on proxy performance, and the effect of low-bandwidth modem client connections. We find that proxy implementations differ in their performance characteristics significantly. In addition, though disk arms appear to be the bottlenecks in proxy throughput, adding extra disks does not result in performance improvement for all proxy implementations. Finally, we find that the latency advantage of caching proxies vanishes in front of modem connections.

1 Introduction

As the World Wide Web continues to grow, caching proxies become a critical component to handle Web traffic and reduce both network traffic and client latency. However, despite its importance, there has been little understanding of how different proxy servers perform compared with each other, and how they behave under different workloads. The design and implementation of a proxy benchmark is the first step to allow one easily to test and understand the performance characteristics of a proxy server. A benchmark allows customers not only to test the performance of a proxy running on different software and hardware platforms, but also to compare different proxy implementations and choose one that best matches the customer's requirements. The Wisconsin Proxy Benchmark (WPB) has been developed in an attempt to provide a tool to analyze and predict performance of different proxy products in real-life situations.

The main feature of WPB is that it tries to replicate the workload characteristics found in real-life Web proxy traces. WPB consists of Web client and Web server processes. First, it generate server responses who sizes follow the heavy tailed Pareto distribution described in [4]. In other words, it includes very large files with a non-negligible probability. This is important because heavy-tail distribution of file sizes does impact proxy behavior. as it must handle (and store in the local cache) files with a wide range of sizes. Second, the benchmark generate a request stream that has the same temporal locality as those found in real proxy traces. Studies have shown that the probability that a document is requested t requests after the last request to it is proportional to $1/t$ [8, 3]. The benchmark replicates the probability distribution and measures the hit ratio of the proxy cache. Third, the benchmark emulates Web server latency by letting the server process delay sending back responses to the proxy. This is because the benchmark is often run in a local area network, and there is no natural way to incur long latencies when fetching documents from the servers. However, Web server latencies affect the resource requirements at the proxy system, particular network descriptors, and must be modelled. Thus, the benchmark supports configurable server latencies in testing proxy systems.

The main performance data collected by the benchmark are latency, proxy hit ratio and byte hit ratio, and number of client errors. There is no single performance number since different environments weight the four performance metrics differently. Proxy throughput is estimated by dividing the request rate by the request latency.

Using the benchmark, we compare the performance of four popular proxy servers – Apache, Cern, Squid and a Cern-derived commercial proxy – running on the same hardware and software platforms. We find that caching incur significant overhead in terms of client latency in all proxy systems. We also find that the different implementation styles of the proxy software result in quite different performance characteristics, including hit ratio, client latency and client connection errors. In addition, the proxy softwares stress the CPU and disks differently.

We then use the benchmark to analyze the impact of adding one extra disk on the overall performance of proxies. We can only experiment with Squid and the Cern-derived proxy as neither Apache nor Cern allows one to spread the cache storage over multiple disks. These results show that disk is the main bottleneck during the operating of busy proxies. However, although this bottleneck is reduced when one extra disk is added to the system, the overall proxy performance does not improve as much as we expected. In fact, Squid's performance remains about the same.

Finally, using an client machine that can emulate multiple modem connections [2], we analyze the behavior of the four proxies when they must handle requests sent by clients connected through very low bandwidth connections. These results show that, in this case, transmission delays are the main component of latency and the low bandwidth effect clearly dominates the overall performance. As a consequence, client latency increases by more than a fact of two, and caching does not reduce client latency significantly.

This paper is organized as follows. Section 2 presents a detailed description of the design and implementation of WPB. Section 3 shows a performance comparison of four popular proxy servers. Section 5 and Section 6 give some insight on the effect of multiple disks and low bandwidth connections on proxy performance. Section 7 discusses related work. Finally, section 8 shows the conclusions and future work.

2 Wisconsin Proxy Benchmark

As the World-Wide Web continues to grow, many institutions are considering the installation of proxy servers for performance and security reasons. As the industry moves to meet customer demands and produces many Web proxy server products, there is a critical need for a benchmark that can compare the performance of various proxy products. This section describes our effort to build a proxy benchmark called the Wisconsin Proxy Benchmark (WPB).

Figure 1 shows an overview of the design of WPB and illustrates a typical benchmarking setup. The general setup of the benchmark is that a collection of Web *client* machines are connected to the proxy system under testing, which is in turn connected to a collection of Web *server* machines. There can be more than one client or server processes running on a client or server machine. The client and server processes run the client and server codes in the benchmark, instead of running a browser or a Web server. There is also a *master* process to coordinate the actions of client processes and generate an overall performance report. Some of the setup parameters are defined in a configuration file. The following sections describe each of the main component of WPB as well as the main performance metrics reported.

2.1 Master Process

A master process is needed to coordinate the actions of client processes. Once a client process starts, it sends an "ALIVE" message to the master process, and then waits for a "GO" message from the master. Once it receives "GO", it starts sending HTTP requests to the proxy as described below. The master process waits till all client processes send it the "ALIVE" message, then

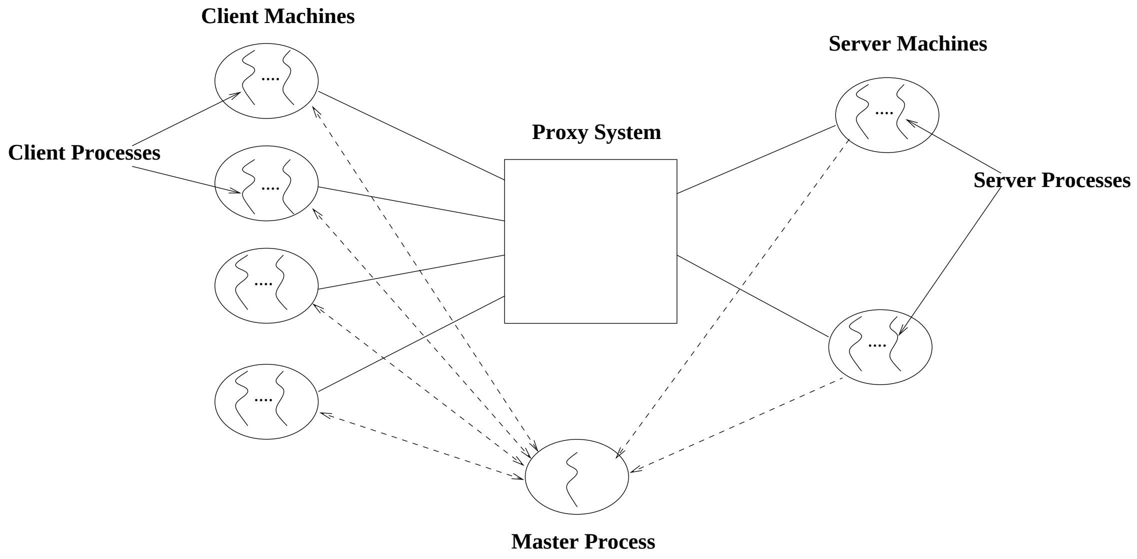

Figure 1: Design of WPB 1.0

sends "GO" to all clients. After sending all "GO" messages, the master waits for the end of the experiment when it receives statistics from all clients and servers. Based on this data, it generates an overall performance report. The number of client and server processes that the master has to wait for is given in a configuration file, whose name is passed as the command line argument to the master process. The master process may run on one of the client machines.

2.2 Client Processes

The client process runs on the client machine and issues HTTP requests one after another with no thinking time in between. This means that clients send requests as fast as the proxy can handle them. The client process takes the following parameters as command line arguments: URL address of the proxy server (e.g. cowb05.cs.wisc.edu:3128/), number of HTTP requests to issue, seed for the random number generator, and name of the configuration file specifying the Web servers to which the client should send requests. Currently, the clients send HTTP requests in the format of "GET http://server_name:port_number/dummy[filenum].html HTTP/1.0", for example, "GET http://cowb06.cs.wisc.edu:8005/dummy356.html". The server_name, port_number and filenum vary from requests to requests. Clearly, our client code does not yet include other types of HTTP requests or HTTP 1.1 persistent connection. We plan to fix it soon, once we learn more about the typical mix of HTTP requests from the clients and the characteristics of most persistent connections.

The client process varies the server_name, port_number and filenum of each request so that the request stream has a particular inherent hit ratio and follows the temporal locality pattern observed in most proxy traces. The client process sends requests in two stages. During the first stage, the client sends N requests, where N is the command line argument specifying the number of requests need to be sent. For each request, the client picks a random server, picks a random port at the server, and sends an HTTP request with the filenum increasing from 1 to N. Thus, during the first stage there is no cache hit in the request stream, since the file number increases from 1 to N. These requests serve to populate the cache, and also stress the cache replacement mechanisms in the proxy. The requests are all recorded in an array that is used in the second stage.

During the second stage, the client also sends N requests, but for each request, it picks a random number and takes different actions depending on the random number. If the number is higher than a certain constant, a new request is issued. If the number is lower than the constant, the client re-issues a request that it has issued before. Thus, the constant is the inherent hit ratio in the request stream. If the client needs to re-issue an old request, it chooses the request it issue t requests ago with probability proportional to $\frac{1}{t}$. More specifically, the client program maintains the sum of $\frac{1}{t}$ for t from 1 to the number of requests issued (call it S). Everytime, it has to issue an old request, it picks a random number from 0 to 1 (call it r), calculates $r * S$, and chooses t where $\sum_{i=1}^{t-1} \frac{1}{i} \; < \; r * S \; < \; \sum_{i=1}^{t} \frac{1}{i}$. In essence, t is chosen with probability $\frac{1}{S*t}$.

The above temporal locality pattern is chosen based on a number of studies on the locality in Web access streams seen by the proxy. (We have inspected the locality curves of the requests generated by our code and found it to be similar to those obtained from traces.) Note here that we only capture temporal locality, and do not model spatial locality at all. We plan to include spatial locality models when we have more information.

Finally, the inherent hit ratio in the second stage of requests can be specified in the configuration file. The default value is 50%.

2.3 Server Processes

The server process listens on a particular port on the server machine. When the server receives an HTTP request, it parses the URL and finds the filenum. It then chooses a random number (according to a particular distribution function) as the size of the HTML document, and forks a child process. The child process sleeps for a specified number of seconds, then constructs an array of the specified size, fills the array with string "aaa[filenum]", replies to the HTTP request with the array attached to a pre-filled response header, and then exits. The response header specifies that the "document" is last modified at a fixed date, and expires in three days. The server process also makes sure that if it is servicing a request it has serviced before, the file size and array contents stay the same.

The sleeping time in the child process models the delay in the Internet and the server seen by the proxy. We feel that it is important to model this delay because in practice, latency in serving HTTP requests affects the resource requirement at the proxy. Originally, we set the sleeping time to be a random variable from 0 to 10 seconds, to reflect the varying latency seen by the proxy. In our preliminary testing, in order to reduce the variability between experiments, we have change the latency to be a constant number of seconds that can be set through a command line argument. We are still investigating whether variable latencies would expose different problems in proxy performance from constant latency. For now, we recommend using a constant latency (see below, benchmarking rules).

Currently, the server process does not support other types of GET requests, such as conditional GET, which we will fix soon. The server process also gives the fixed last-modified-date and time-to-live for every response, which would be changed as we learn more about the distribution of TTL in practice.

The server program uses two different file size distributions. The default distribution is very primitive. It is basically a uniform distribution from 0 to 40K bytes for 99% of the requests, and 1MB for 1% of the requests. It is also possible to use a more realistic file size distribution, such as the heavy tail Pareto distribution. In this case, the two parameters of the distribtuion, α and k, must be specified in the configuration file. The parameter k represents the minimum file size and α is such that the average file size av is given by $\alpha = \frac{average}{average-k}$. Typical values of α and k are 1.1 and 3.0KB, as shown in [4].

2.4 Configuration File

The configuration file specifies the following: the machine where the master process runs, the port number that the master process is listening at, the total number of client processes, the total number of server machines, and the specification of the server processes for each server machine. The specification of the server processes includes the name of the server machine, the base port number, and the number of ports at which there are server processes listening. For example, if the specification is "cowb06.cs.wisc.edu 8000 10", it means that there is a server process listening at each of the ports 8000, 8001, 8002, ..., 8009 on cowb06.cs.wisc.edu. This way one server machine can host more than one server processes. The inherent hit ratio is an optional parameter as well as α and k, the parameters that define the heavy tail file size distribution. It is also possible to specify the average file size instead of α. In this case, α can be calculated by the formula 2.3. The default minimum file size is 3.0KB.

2.5 Main Performance Metrics

At the end of the experiment, the master process receives from clients the number of requests sent and the total number of bytes requested and from servers the total number of requests and bytes that they serviced. Clients also send the average latency and the total number of errors observed during the two phases of the experiment. An error is reported everytime a connection between client and proxy could not be established, because the pending connection queue was full. Then, the master process generates an overall performance report that contains the following data: average latency in the first and second phase, total number of errors observed during both phases, hit ratio and byte hit ratio. Proxy throughput, i.e., the number of requests that are handled in each time unit, can be estimated based on the average latencies and on the number of requests handled.

2.6 Recommended Benchmarking Steps and Rules

The setup of the benchmarking experiments should follow the rule that the ratio between the number of client machines and the number of server machines is always 2 to 1, and the ratio between the total number of client processes to the total number of server processes is also kept at 4 to 1. The server latency should be at least 3 seconds. Until we have a better understanding of the spatial locality in user access streams, these ratios seem to be a reasonable choice.

Each run should last at least 10 minutes. Thus, the number of operations should be chosen so that this minimum duration is achieved. The total size of HTTP documents fetched by the clients should be at least four times the cache size of the proxy.

The current version of the benchmark does not model DNS lookups, HTTP 1.1 persistent connections, conditional Get and other forms of HTTP requests, realistic path name for URL's and spatial locality in Web accesses (i.e., once a user accesses a document from one Web server, it tends to access other documents at the same Web server). We are continueing the development of the benchmark and hope to eliminate these limitations in future versions of the benchmark.

2.7 Measuring the Behavior of Overloaded Proxy Servers

We have also incorporated new features to WPB to generate bursty traffic, with peak loads that exceed the capacity of the proxy, based on [1]. We have changed the client process code to have a pair of process – *sender* and *handler* – generating HTTP traffic to the proxy using non-blocking sockets. The *sender* process is responsible for establishing a connection with the proxy. After the connection is established, it passes the socket descriptor to the *handler* and sends a new connection request. The *handler* is responsible for keeping track of all pending connections and receiving the requested files from the proxy. If the connection could not be established in a reasonable amount of time (500 ms in our implementation), the *sender* issues a new connection request immediately. Each pair *sender-handler* manages a pre-defined number of sockets descriptors so that the number of sockets (simulated clients) that are trying to establish new connections is kept constant.

We have done some experiments with this new version of WPB (version 1.1) using different proxies. The main results show that there are a lot of connection time-outs that are reported as errors. The timeout period used in our implementation (500 ms) is much larger than the maximal round-trip time between client and proxy in our testbed (described in section 3). Therefore, the proxies were actually overloaded and could not handle many of the requests. For some proxies, we observed a very big number of errors so that the effective proxy throughput was very low. As a consequence, latency dropped very significantly. We are still working on this new version of WPB and trying to interpret the results. Therefore, all the results presented in this paper were collected using version 1.0 of WPB.

3 Performance of Example Systems

We used WPB to measure the performance of four proxy systems:

- **Apache version 1.3b2** - it is a multiprocess proxy that forks, at startup, a predefined number of processes to handle incoming requests. This number is dynamically adjusted as a response of the current load in the proxy. Apache stores cached documents in a four-level directory tree. However, the number of entries in each subdirectory as well as the size of directories in the same level are not fixed. Apache also copies documents to a temporary file in the root cache directory before copying it to a final location. Currently, we have not been able to find any information about how Apache finds this final location.

- **Cern version 3.0A** - it is a multiprocess proxy that forks a new process to handle each request. After handling a single request, the new process terminates. It uses the file system to cache data (web documents) as well as proxy meta-data, such as expiration times, content type, etc. Web pages are stored in separate files and the directory structure is derived from the structure of the URL: each URL component is translated into a directory name. The directory path for a specific URL is called URL directory. Metadata for each file, used to find out whether it is stored in the cache, is kept in the corresponding URL directory.

- **A Cern-derived comercial proxy** - This is a multiprocess proxy that at startup forks a predefined number of processes that are responsible for handling all the incoming requests. After a predefined number of requests serviced, a process terminates and gets respawned by a master process. Therefore, the total number of processes remains constant. For our experiments, we set the number of processes to 32, the default value. It uses the file system to cache data and proxy meta-data. The proxy cache is separated into one or more three-level cache sections. Each cache section contains 64 subdirectories and can hold 100 to 250 MB of data. It uses an algorithm to determine the directory where a document should be stored that tries to ensure equal dispersion of documents in the sections. The proxy uses the RSA MD5 algorithm to reduce a URL to 8 characters, which it uses for the file name of the document and to determine the storage directory. This proxy server will be identified as **Proxy N**.

- **Squid version 1.1.14** - it uses non-blocking network I/O abstractions in order to avoid forking new processes. Only one process handles all incoming requests. It manages its own resources and keeps meta-data about the cache contents in main memory. Therefore, it is not necessary to access disk to determine if a file is stored in the cache. Squid uses main memory to cache in-transit objects, to cache the most recently used objects and to cache error responses. The cache storage in disk is structured as a three-level cache: there are 16 sections, each one containing 256 subdirectories where files are stored. Squid maps URLs to cache object names using "fingerprinting". It implements its own DNS cache and uses a predefine number of "DNS servers" to which it sends non-blocking DNS requests

We have run a set of experiments using WPB (version 1.0) to analyse how the above systems perform under different loads. We varied the number of client processes and collected statistics for caching and no caching configurations. Before presenting the mains results obtained, we describe the hardware platform where the experiments were run.

We ran our experiments in a COW – *Cluster of Workstatsions*, that consists of forty Sun Sparcstations 20, each one with 2 66 Mhz CPUs, 64 MB of main memory and two 1 GB disk. The COW nodes are interconnected through different network interfaces, including a 100 Base-T

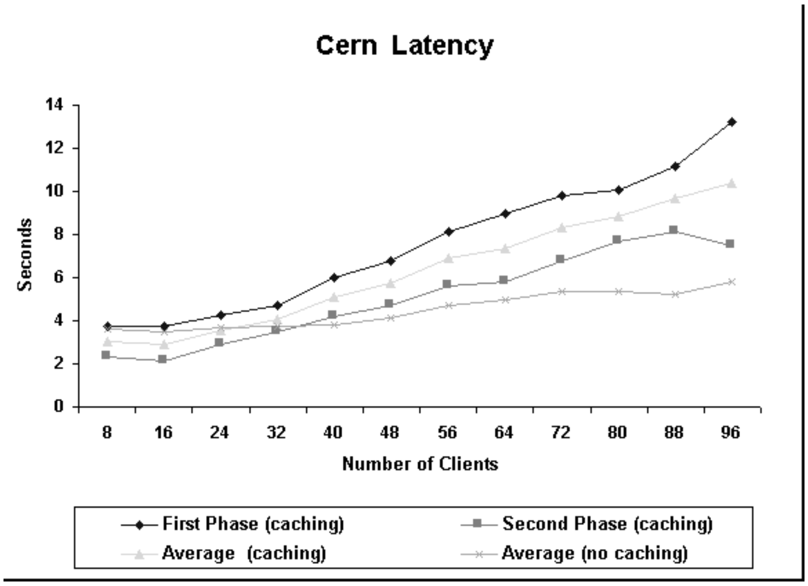

Figure 2: Client Latency for CERN 3.0A.

interface, which we used during the experiments. We ran experiments varying the number of clients from 8 to 96, using four client machines and two server machines. We kept the ratio 4:1 between number of client processes and number of server processes. The cache size was set to 75 MB. **The server latency is set to 3 seconds in all the experiments**. The inherent hit ratio in the client request stream is the default value 50%. Thus, the maximum hit ratio that can be observed in the experiments (which is an average of the two phases) is around 25%.

Figure 2 shows, for CERN 3.0A, how the average latency varies as a function of the number of clients in both caching and no caching configurations. For caching configuration, it also shows how the latency in the first and second phases of the experiment varies. For no caching, latency increases very slowly; for caching, however, it does increase with the number of clients. These curves show that, for our experiments, the cost of a hit may offset the cost of accessing the remote server if a great number of clients are concurrently trying to connect to the proxy. In other words, network transmission time is shorter than the time spent retrieving the file from disk. It is interesting to note that this is true despite the fact that we try to model transmission overhead by imposing a delay of 3 seconds in the server. For Cern, this limit is 32 clients; after this point, the second phase latency is bigger than the no caching average latency. Figure 3 shows the hit and byte hit ratios as a function of the number of clients. Both curves are very similar and, roughly, they show a decrease in the hit ratio. As the number of clients increases, the total number of unique files retrieved from the proxy also increases (since each client uses a different seed for the random number generator). As a consequence, hit ratio decreases. Figure 4 shows how CERN is effective in handling the incoming requests. As can be seen, after 40 concurrent clients, CERN is unable to handle all the connection requests that it receives. The number of errors increases as the number of clients increases. However, during the second phase of the caching experiment, when hits occur, no errors are observed

Figure 5 shows how latency varies for Apache. Although the curves are quite unstable, it is clear that latency for the caching experiments increases as a function of the number of clients. However, latency remains roughly constant when caching is disabled. Latency for the second phase of the caching experiments remains shorter than the no caching latency up to 72 clients. After this point, it increases significantly. The degradation of Apache performance as the number of clients increases is also noteworthy. The latency increases up to almost 30 seconds for 88 clients when no hit is observed. This probably is a consequence of the two-phase write that may involve several memory operations. Figure 6 shows the hit ratio curves for Apache. It seems like hit ratio in

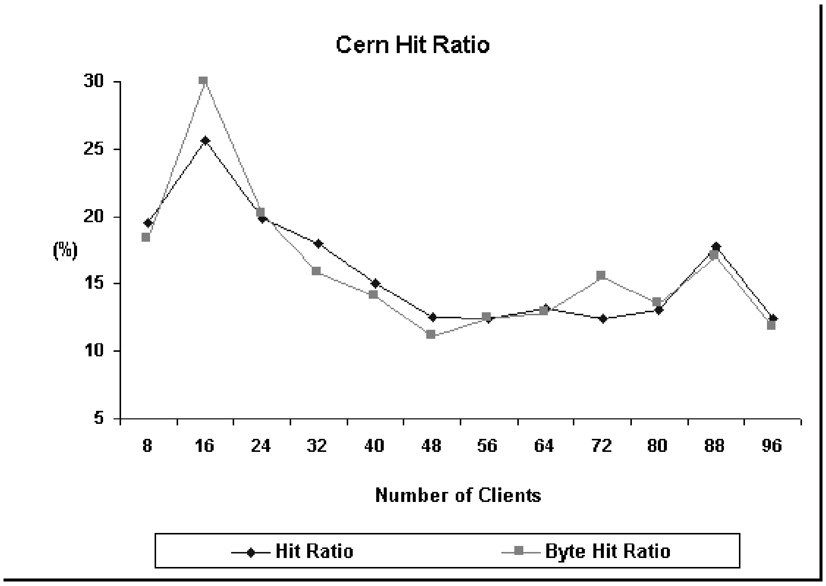

Figure 3: Hit Ratio for CERN 3.0A.

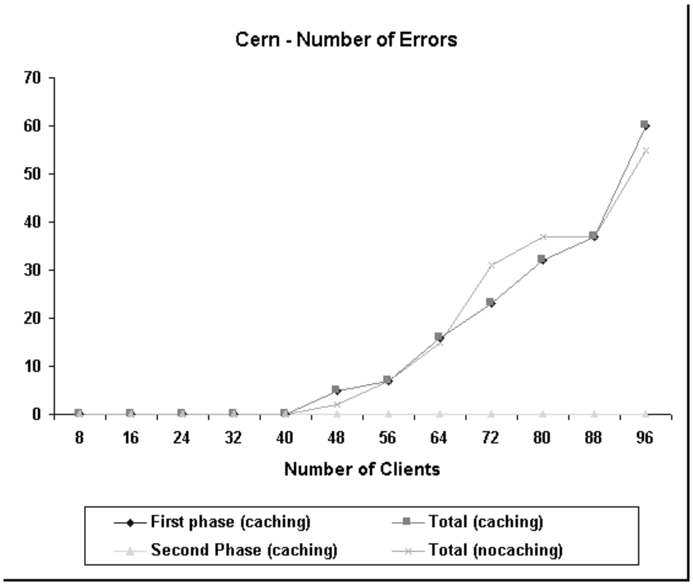

Figure 4: Client Errors for CERN 3.0A.

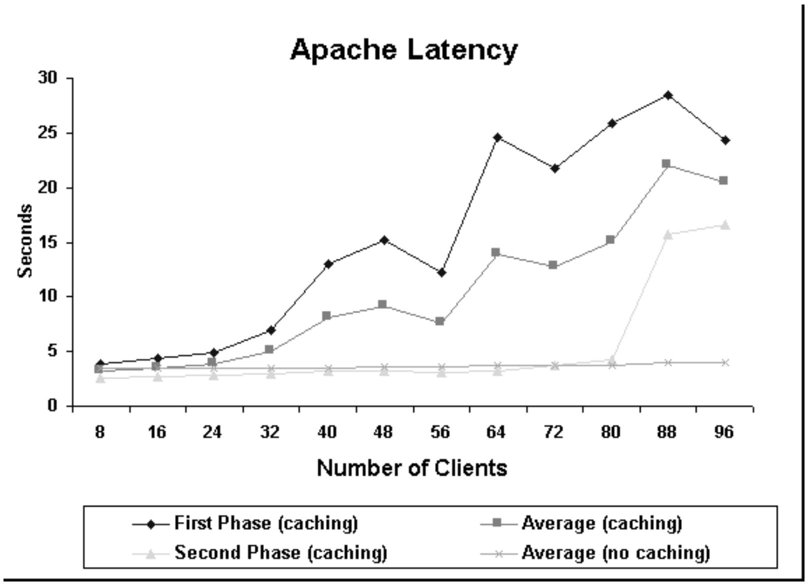

Figure 5: Hit Ratio for Apache 1.3b2.

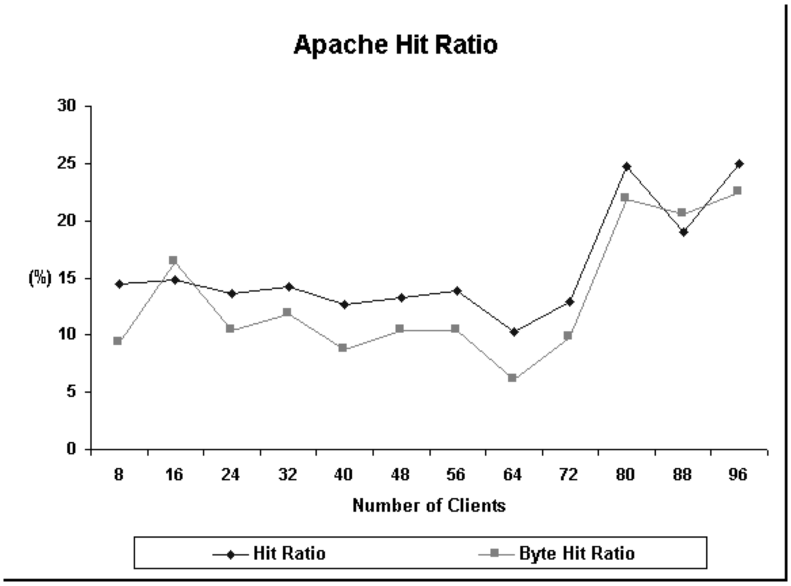

Figure 6: Hit Ratio for Apache 1.3b2.

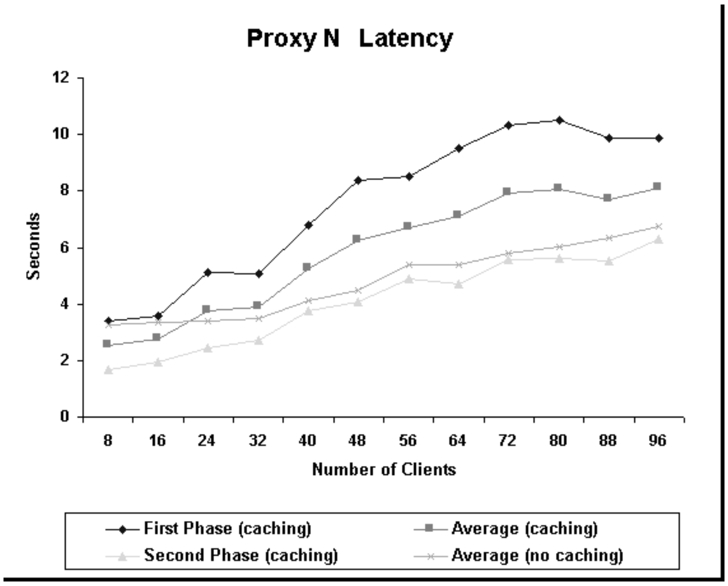

Figure 7: Hit Ratio for Proxy N.

Apache is not significantly affected by the number of clients. We are currently unable to explain Apache's behavior. No errors were observed during the experiments.

Figure 7 shows the latency curves for Proxy N. The average latency for no caching experiments increases linearly with the number of clients for this proxy. Also, the second phase latency for the caching experiments is always lower than the no caching latency. This may be due either to a better usage of disk resources or a more expensive implementation. Since latency increases linearly with the number of clients even when caching is disabled, we conjecture that overhead of proxy implementation is responsible for this behavior. One explanation is that because the number of proxy processes is always the same, as the number of client increases, more requests must be delayed waiting for available processes to handle them. This is true for both caching and no caching experiments. Figure 8 shows hit ratio curves for Proxy N. Hit ratio degrades very slowly. The only proxy that has a better hit ratio is squid, as will be shown next. This may be due to different algorithms for cleaning the cache. Figure 9 shows that the fixed number of process implementation results in a high number of errors during the experiments, especially when there is no hit in the cache. With 32 processes, the proxy is unable to handle all the requests if there are more than 48 clients. If the number of proxy processes is inceased to 64, the overall behavior is the same, but the saturation point is shifted to 88 clients. So, this parameter must be carefully tuned in order to minimize the number of errors. As a consequence, proxy N may have problems in handling bursty traffic since this parameter must be statically chosen.

Figure 10 shows the latency curves for Squid. These curves have a behavior similar to those for Apache and Cern. However, Cern has a slightly better average latency for caching experiments. These results are consistent to those presented in [9]. When caching is disabled, Squid performs better. Figure 11 shows the hit ratio curves. Byte hit ratio is very unstable but it can be seen that Squid can sustain a fairly constant hit ratio, independent of the number of clients. We are currently trying to investigate this behavior with more details.

4 Effect of Adding Disk Arms

Several studies have claimed that disks are a main bottleneck in the performance of busy proxies. We wanted to analyse the impact of spreading the cached files over multiple disks on proxy performance. We expected that by increasing the number of disks, queueing overheads would be

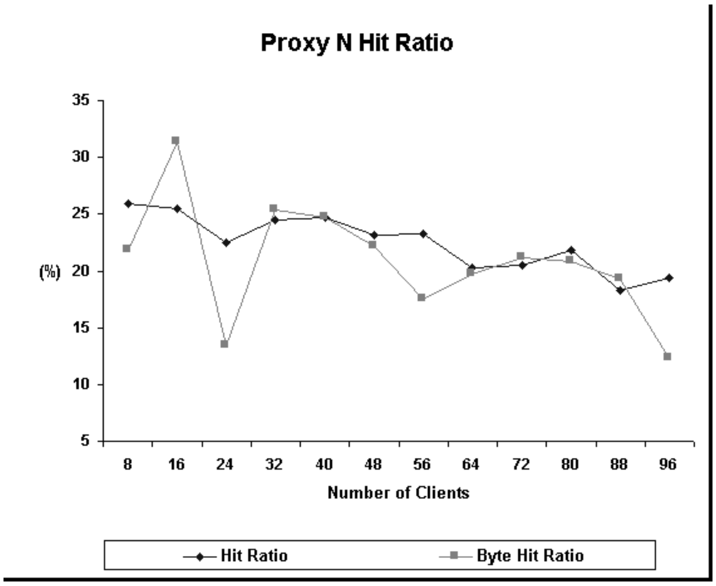

Figure 8: Hit Ratio for Proxy N.

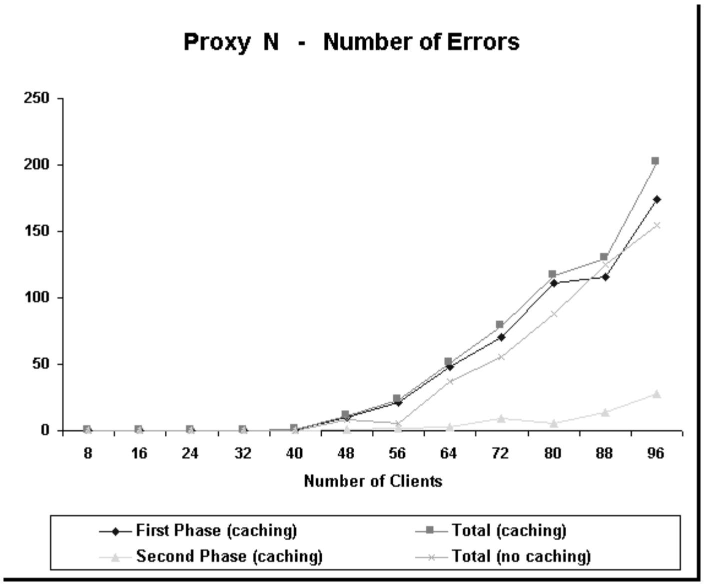

Figure 9: Client Error for Proxy N.

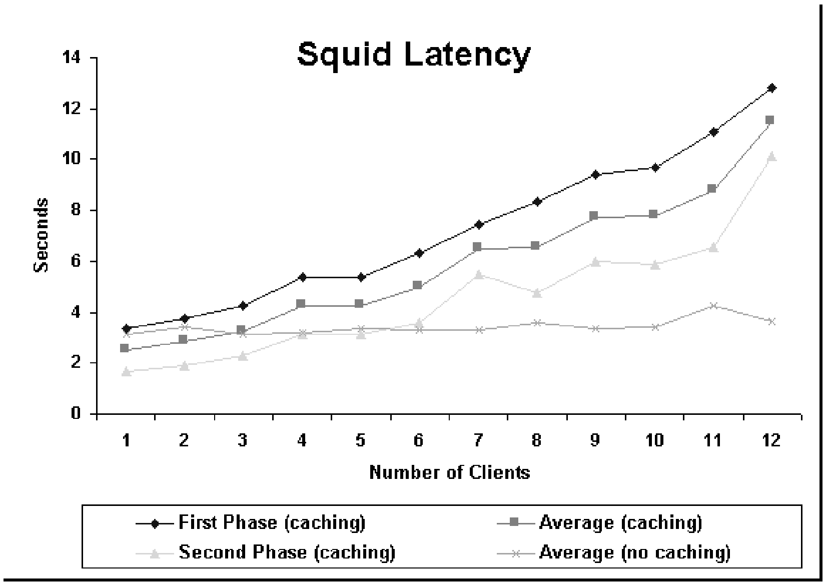

Figure 10: Hit Ratio for Squid 1.1.14.

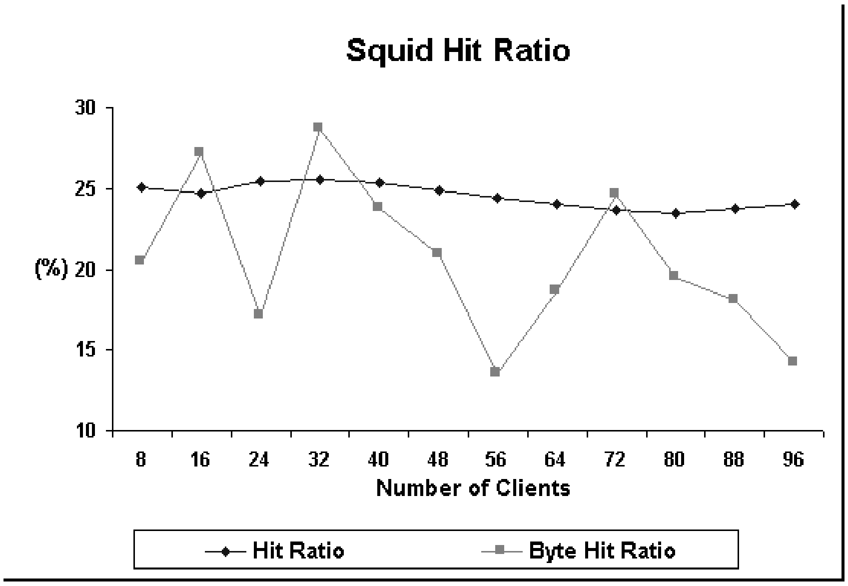

Figure 11: Hit Ratio for Squid 1.1.14.

reduced, the time spent servicing each disk request would be shortened and, ultimately this would reflect on the overall performance of the proxy.

Metrics	one disk	two disks	nocaching
First phase latency (s)	9.00 (2%)	9.88 (1%)	3.65 (0.6%)
Second phase latency (s)	5.44 (1.1%)	5.89 (5%)	3.42 (2%)
First phase errors	0	0	0
Second phase errors	0	0	0
Hit Ratio	20.29 (1%)	22.50 (1%)	0
Byte Hit Ratio	5.67 (22%)	11.52 (9%)	0

Table 1: The effect of multiple disks on Squid performance.

Metrics	one disk	two disks	nocaching
First phase latency (s)	9.82 (5%)	8.78 (2%)	6.71 (5%)
Second phase latency (s)	5.42 (4%)	4.91 (5%)	5.34 (6%)
First phase errors	86.67 (4%)	68.67 (7%)	77.67 (17%)
Second phase errors	12.67 (32%)	14.33 (20%)	16.33 (52%)
Hit Ratio	21.61 (2%)	22.46 (0.7%)	0
Byte Hit Ratio	12.18 (56%)	21.70 (31%)	0

Table 2: The effect of multiple disks on Proxy N performance.

Only two of the four proxy servers in our testbed allowed us to specify multiple directories for cache storage: Squid and Proxy N. In this section, we present results for both Squid and Proxy N when the cache storage is spread over one and two disks. We also compare these results with those collected when caching was disabled. For these experiments, we used the same cache size (75 MB), since we are interested only in undertanding the impact of one extra disk in proxy performance. Table 1 shows these results for Squid. Surprisingly, there is no improvement in the overall performance when we add an extra disk. Actually, there is a small slowdown in latency of both phases, but hit ratio remains around the same. Table 2 shows results for Proxy N. For Proxy N, the extra disk guaranteed an improvement of 10% in client latency. Hit ratio remained around the same. However, the number of errors occurred in the first phase decreased by 20%. This is probably a side effect of the improvement in latency: because requests are handled faster, new requests can be taken out from the pending connections queue faster and the probability of finding this queue full decreases.

Comparing Proxy N's latency for two disk and no caching experiments, it can be seen that adding the extra disk indeed alleviates the disk bottleneck for this proxy. In the no-caching experiment, there is minimal load on the disk. In the caching experiment, when the proxy has only one disk arm to use, the client latency in the first phase is increased by 46%, mostly contributed by the disk bottleneck. However, when the proxy has two disk arms to use, the latency increase is limited to only 30%. Furthermore, when the proxy has two disk arms to use, the processing of cache hits is sped up and the second phase latency is significantly improved.

However, despite the improvements, we were very surprised with these results, since we expected a more drastic improvement in latency. During those experiments, we also collected disk, processor and paging activities using *vmstat* and *iostat* in order to have a better picture of how the system resources are consumed by the proxies.

Table 3 show the main statistics collected by these tools for Proxy N. Disk is clearly a bottleneck, since it was busy over 94% of time. With the extra disk, read and write requests were spread

Metrics	one disk	two disks	nocaching
Disk 1 - reads/s	–	10.60 (7%)	–
Disk 1 - writes/s	–	41.43 (10%)	–
Disk 1 - Kbytes read/s	–	43.53 (9%)	–
Disk 1 - Kbytes written/s	–	239.96 (5%)	–
Disk 1 - wait	–	0.068 (31%)	–
Disk 1 - svc_t	–	106.96 (8%)	–
Disk 1 - busy (%)	–	79.11 (9%)	–
Disk 2 - reads/s	7.06 (4%)	7.00 (10%)	0.0106 (153%)
Disk 2 - writes/s	57.47 (4%)	26.08 (12%)	0.53(4%)
Disk 2 - Kbytes read/s	18.09 (11%)	29.54 (8%)	0.0576 (158 %)
Disk 2 - Kbytes written/s	344.09 (1%)	156.53 (16%)	4.02(5%)
Disk 2 - wait	1.82 (11%)	0.00168 (67%)	0
Disk 2 - svc_t	174.46 (2%)	67.53 (8%)	24.18 (4%)
Disk 2 - busy (%)	94.36 (3%)	48.83 (12%)	1.14 (4%)
cpu idle (%)	76.17 (0.8%)	72.79 (4%)	80.11 (0.7%)
cpu user (%)	6.57 (3%)	7.36 (16%)	6.72 (1%)
cpu system (%)	17.25 (3%)	19.83 (11%)	13.17 (4%)
page-in/s	31.54 (2%)	39.30 (10%)	0.062 (173%)
page-out/s	180.57 (4%)	198.40 (10%)	0.539(173%)

Table 3: Proxy N resource consumption for one disk, two disk and no caching experiments.

over two disks and, as a consequence, service time (`svc_t`) which is the average time spenting servicing a request dropped significantly, due mainly to reduction of queueing delay. The average queue length (`wait`) drops to almost 0 for both disks. The total number of read and write operations is bigger for the two-disk experiment as a consquence of less contention to disk access. Disk throughput increases as well as disk bandwidth. Processor and paging activity remains almost the same, with slight increase in cpu utilization, both in terms of user and system modes. However, the statistics show that the load is not evenly distributed between the disks: disk one received a bigger number of requests. This may be one explanation for the less-than-expected improvements from the two disk arms, and we are still investigating the issues.

Table 4 shows similar results for Squid. It is clear that the disk bottleneck was reduced when one extra disk was added. The service time (`svc_t`) was reduced by almost 50% for both disks. The queue length (`actv`) and busy time were also reduced. As a consequence, disk throughtput and bandwidth increased. However, this improvements were not reflected in the client latency. Processor utilization remained about the same but paging activity increases when the extra disk is added. We are still in the process of finding out why Squid behaves sub-optimally, and why Squid and Proxy N behaves so differently.

5 Effect of Low Bandwidth Client-Connections

We were also interested in analysing the impact of low bandwidth connections on proxy performance. We wanted to analyse how proxies behave when they must handle requests sent by several clients, using slow connections, such as modems. We used a modem emulator [2] which introduces delays to each IP packet transmitted in order to achieve a certain effective bandwidth that is smaller than the one provided by the network connection. This modem emulator, implemented in the Linux operating system, assigns to each process a predefined maximum bandwidth that it can use. The default bandwidth used in our experiments is 28.8Kbps for each process. We used the WPB to measure the performance of the four proxy servers included in our testbed when requests

Metrics	one disk	two disks	nocaching
Disk 1 - reads/s	–	12.93 (5%)	–
Disk 1 - writes/s	–	17.30 (4%)	–
Disk 1 - Kbytes read/s	–	61.60 (7%)	–
Disk 1 - Kbytes written/s	–	134.48 (3%)	–
Disk 1 - wait	–	0.000778 (23%)	–
Disk 1 - svc_t	–	79.62 (2%)	–
Disk 1 - busy (%)	–	43.71 (3%)	–
Disk 2 - reads/s	14.71 (6%)	13.35 (5%)	0.0797 (12%)
Disk 2 - writes/s	28.98 (12%)	17.56 (3%)	0.663 (14%)
Disk 2 - Kbytes read/s	58.48 (9%)	69.17 (5%)	0.243 (45%)
Disk 2 - Kbytes written/s	207.07 (15%)	134.50 (3%)	6.67 (12%)
Disk 2 - wait	0.80 (18%)	0.0041 (37%)	0
Disk 2 - svc_t	153.23 (11%)	80.56 (2%)	25.69 (10%)
Disk 2 - busy (%)	58.62 (8%)	43.50 (2%)	1.297 (16%)
cpu idle (%)	78.59 (2%)	79.14 (0.5%)	67.88 (8%)
cpu user (%)	7.04 (7%)	6.73 (1%)	10.74 (18%)
cpu system (%)	14.32 (7%)	14.11 (2%)	21.35 (18%)
page-in/s	37.45 (9%)	47.96 (0.7%)	0.0433 (128%)
page-out/s	63.18 (34%)	100.97 (6%)	3.01 (86%)

Table 4: Squid resource consumption for one disk, two disk and no caching experiments.

were sent by clients running on top of this modified version of Linux. We spread 30 clients over three DEC Celebris XL590 90 MHz Pentium machines with 32 MB of main memory each and standard 10 Mbps Ethernet card. We spawn 20 WPB server processes on two COW nodes. We compare the results for these experiments with those collected when the same number of clients were spread over the same machines running an unmodified Linux. The network connecting clients and proxies is non-dedicated and includes one router in the path.

Tables 5 and 6 show the results obtained for all four proxies. No errors were observed during the experiments for any proxy server. It is clear that the low bandwidth effect dominates. The latency increases by more than a factor of 2 for all proxies. It is also interesting to notice that the difference between the latencies observed in the first and second phases is smaller for the low bandwidth connection. This reflects the fact that the time spent in the communication between client and proxy is the dominant factor in the overall performance and it is much more important than the delay introduced by misses in the cache, since the connections between servers and proxy is much faster in our setup. The decrease in the hit ratio is also noteworthy, especially for Apache and Proxy N. Observing the results collected by *iostat* and *vmstat*, it is clear that the bottleneck is the transmission in the network. Despite the shorter service time, disk throughput is lower for the experiments with low bandwidth. The utilization of both disk and cpu is much lower for the 28Kbps bandwidth experiment and, as consequence, the overall proxy throughput degrades.

6 Related Work

Benchmarking Web servers is an active research area. Several benchmarks for Web servers have been developed, including WebStone [11] and SPECWeb. There are also studies on the overload behavior of the benchmarks and improvement of the benchmarks [1]. However none of the benchmarks can be easily used to measure proxy performances. We borrowed several ideas from these benchmarks when designing WPB. However, WPB incorporates unique characteristics that make it appropriate to analyze the performance of proxy servers.

Squid		
Metric	**28Kbps bandwidth**	**10 Mbps bandwidth**
First phase latency (s)	12.13 (7%)	4.74 (1%)
Second phase latency (s)	13.19 (42%)	2.45 (2%)
Hit Ratio	25.01 (0.5%)	25.11 (0.4%)
Byte Hit Ratio	21.36 (61%)	19.58 (19%)
reads/s	2.44 (51%)	6.68 (3%)
writes/s	6.10 (57%)	26.67 (8%)
svc_t (ms)	65.11 (45%)	182.76 (5%)
disk busy(%)	11.53 (54%)	45.3 (4%)
cpu idle(%)	97.98 (1%)	81.78 (2%)
cpu user (%)	0.51 (65%)	5.56 (6%)
cpu system (%)	1.46 (60%)	12.68 (8%)
Proxy N		
Metric	**28Kbps bandwidth**	**10 Mbps bandwidth**
First phase latency (s)	13.66 (15%)	5.02 (13%)
Second phase latency (s)	10.39 (9%)	4.29 (67%)
Hit Ratio	15.24 (2%)	20.86 (9%)
Byte Hit Ratio	11.30 (2%)	17.95 (8%)
reads/s	1.48 (56%)	6.06 (8%)
writes/s	17.22 (11%)	50.80 (23%)
svc_t (ms)	110.86 (6%)	155.78 (13%)
disk busy(%)	25.52 (14%)	83.19 (20%)
cpu idle(%)	95.06 (0.7%)	79.66 (7%)
cpu user (%)	1.07 (21%)	5.99 (25%)
cpu system (%)	3.79 (11%)	14.36 (27%)

Table 5: The effect of low bandwidth connections on Squid and Proxy N performance.

There has also been many studies measuring the performance of Squid proxies in real use [9, 10]. Our results confirm most of the findings in those studies. Different from those studies, WPB can be used as a tool to pin point the performance problems of a variety of proxy products.

Finally, there have been many studies on characteristics of proxy traces [3, 8, 5, 6, 7]. Our research borrows results from those studies in designing WPB.

7 Conclusion

In this paper, we have described the design of the Wisconsin Proxy Benchmark and the result of using the benchmark to compare four proxy implementations. We also use the benchmark to study the effect of extra disk arms and modem client connections.

Our main findings are the following:

- Disk is the main bottleneck during the operation of busy proxies: disk is busy up to 90% of time while CPU is idle for more 70% of time. Adding an extra disk reduces the bottleneck in the disk. However, for Squid, this reduction did not reflect in an improvement in the overall performance of the proxy. For proxy N, an improvement of 10% was achieved.

- When a proxy must handle requests sent throught very low bandwidth connections, the time spent in the network dominates. Both disk and cpu remains idle for more than 70% of time. As a consequence, proxy throughtput decreases and client latency increases by more than a factor of two.

Cern		
Metric	**28Kbps bandwidth**	**10 Mbps bandwidth**
First phase latency (s)	12.61 (4%)	4.98 (21%)
Second phase latency (s)	11.19 (21%)	2.92 (7%)
Hit Ratio	19.12 (24%)	22.98 (7%)
Byte Hit Ratio	15.59 (38%)	18.26 (9%)
reads/s	0.27 (21%)	1.18 (85%)
writes/s	13.64 (10%)	35.48 (85%)
svc_t (ms)	63.49 (9%)	78.36 (72%)
disk busy(%)	19.50 (13%)	53.99 (87%)
cpu idle(%)	87.12 (3%)	30.79 (18%)
cpu user (%)	2.68 (32%)	15.50 (13%)
cpu system (%)	10.16 (18%)	53.69 (7%)
Apache		
Metric	**28Kbps bandwidth**	**10 Mbps bandwidth**
First phase latency (s)	10.78 (4%)	5.51 (42%)
Second phase latency (s)	9.75 (7%)	3.00 (20%)
Hit Ratio	12.89 (4%)	26.06 (40%)
Byte Hit Ratio	8.84 (25%)	18.07 (51%)
reads/s	6.58 (3%)	9.48 (52%)
writes/s	11.31 (57%)	33.64 (76%)
svc_t (ms)	55.46 (43%)	115.64 (12%)
disk busy(%)	26.93 (35%)	65.62 (47%)
cpu idle(%)	94.05 (1%)	78.50 (6%)
cpu user (%)	0.64 (26%)	2.98 (32%)
cpu system (%)	5.27 (17%)	18.58 (21%)

Table 6: The effect of low bandwidth connection on Cern and Apache performance.

- The performances of Cern and Squid are comparable, despite their vast differences in implementations. Squid mainly suffers from not being able to use the extra processor in the multi-processor system. Cern, on the other hand, uses a process-based structure and utilize two processors. In addition, CERN takes advantage of the file buffer cache, which seems to perform reasonably well.

- Process-based proxy implementations must take care to avoid client connection errors. Both Cern and proxy N (Cern derived proxy) can not handle all requests and many errors occur. These errors reflect the fact that many requests were dropped due to overflow in the pending connection queue. For proxy N, this can be explained by the fact that the number of proxy processes handling all requests is fixed. For Cern, the overhead of forking a new process for each request may be an explanation. The master process (responsible for spawning new processes) can not handle all the requests. On the other hand, no error was observed for Apache. The dynamical adjustment in Apache on the number of processes seems to be successful. Squid does not suffer from this problem because of its event-driven architecture.

- In terms of latency, Apache has the worst performance, probably due to the two-phase store, that introduces extra overhead. Proxy N has a slightly better performance overall. However, this may be a consequence of the great number of errors. Since a smaller number of requests are effectively handled, delays due to contention are reduced.

- In terms of hit ratios, Squid and Apache maintains roughly constant hit ratios across the load. For both Cern and proxy N, hit ratio decreases significantly as the number of client increases.

Clearly, much more work remains to be done. First of all, the Wisconsin Proxy Benchmarks should model spatial locality and HTTP 1.1 persisten connections. We are currently working on this issue. Second, the performance of Squid is baffling and we are instrumenting the code to gain a better understanding. Third, we need to perform more experiments to better understand the impact of slow modem client connections and ways to improve proxy performance in those contexts. Lastly, we plan to investigate the effect of application-level main-memory caching for hot documents and its proper implementation (e.g., avoid double buffering with the operating system's file buffer cache).

References

[1] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server. In *Proceedings of 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[2] Kevin Beach and Pei Cao. Modem emulator: Faking multiple modem connections on a lan. *CS736 project report*, 1998. contact cao@cs.wisc.edu for more detail.

[3] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[4] M. Crovella and A. Bestavros. Self-similiarity in world wide web traffic: Evidence and possible causes. In *Proc of the 1996 Sigmetrics Conference on Measurment and Modeling of Computer systems Philadelphia*, May 1996.

[5] Bradley M. Duska, David Marwood, and Michael J. Feeley. The measured access characteristics of world-wide-web client proxy caches. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[6] Steve Gribble and Eric Brewer. System design issues for internet middleware service: Deduction from a large client trace. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[7] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[8] P. Lorenzetti, L. Rizzo, and L. Vicisano. Replacement policies for a proxy cache. Technical report, Universita di Pisa, Italy, October 1996. URL http://www.iet.unipi.it/ luigi/caching.ps.gz.

[9] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance issues of enterprise level web proxies. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, pages 13–23, June 1997.

[10] Valery Soloviev. Analyzing squid's performance. *Private Communication*, 1998.

[11] Gene Trent and Mark Sake. WebSTONE: The first generation in HTTP server benchmarking. Technical report, MTS, Silicon Graphics Inc., February 1995. available from http://www-europe.sgi.com/TEXT/ Products/WebFORCE/WebStone/paper.html.