

Server Capacity Planning for Web Traffic Workload

Krishna Kant, *Member, IEEE*, and Youjip Won, *Member, IEEE*

Abstract—The goal of this paper is to provide a methodology for determining bandwidth requirements for various hardware components of a World Wide Web server. The paper assumes a traditional symmetric multiprocessor (SMP) architecture for the web-server, although the same analysis applies to an SMP node in a cluster. The paper derives formulae for bandwidth demands for memory, processor data bus, network adapters, disk adapters, I/O-memory paths, and I/O buses. Since the web workload characteristics vary widely, three sample workloads are considered for illustrative purposes: 1) standard SPECweb96, 2) a SPECweb96-like workload that assumes dynamic data and retransmissions, and 3) WebProxy, which models a web proxy server that does not do much caching and, thus, has rather severe requirements. The results point to a few general conclusions regarding Web workloads. In particular, reduction in memory/data bus bandwidth by using the virtual interface architecture (VIA) is very desirable, and the connectivity needs may go well beyond the capabilities of traditional systems based on the traditional PCI-bus. Web workloads also demand a significantly higher memory bandwidth than data bus bandwidth and this disparity is expected to increase with the use of VIA. Also, the current efforts to offload TCP/IP processing may require a larger headroom in I/O subsystem bandwidth than in the processor-memory subsystem.

Index Terms—Web server, traffic characterization, self-similarity, symmetric multiprocessors, caching/proxy server, bandwidth requirements.

1 INTRODUCTION

THE World Wide Web (WWW) came into existence as a medium for delivering hypermedia documents over the network. It has exhibited exponential growth over the past several years and has become the dominant application in both public Internet and in corporate intranet environments. Recent studies suggest that more than 75 percent of the traffic on the Internet backbone is HTTP-related [4]. Therefore, planning and provisioning of adequate web-server resources is critical to getting acceptable user response times.

Under WWW service paradigm, services are provided to the end user based on the client-server paradigm. Client is an application which makes a request for uploading or downloading an *object*, and the server application processes this request. An *object* can be a text file, audio/video file, real-time audio/video stream, database query, etc. Web traffic characteristics and their resource requirements are quite different from the applications such as on-line transaction processing (OLTP) and decision support system (DSS), whose requirements are comparatively much better understood. For example, contrary to OLTP and DSS workloads, Web workloads tend to be extremely network intensive and not much disk I/O intensive. These workloads typically have smaller working sets and yet experience much worse processor cache hit ratios, as our measurements have repeatedly

shown. Also, the networking software stack (e.g., TCP/IP) may become the most severe bottleneck to good performance unless it is implemented carefully.

The objective of this paper is to provide some insight into the requirements for Web server by examining the bandwidth demands for the key hardware resources in a Web server. This requires a choice of some sample Web workloads that accurately characterize real web servers. Unfortunately, as with the rest of the Internet, there is no such thing as a “typical” web-server. Web servers may vary widely even in terms of coarse grained characteristics such as: 1) *native* vs. *proxy* server, 2) electronic commerce vs. search engine host vs. a rather passive display site, 3) image intensive (e.g., shopping catalog) vs. text-intensive (general documents), 4) highly dynamic vs. primarily static content etc. Because of the paucity of available measurement data, we do not attempt to explore the entire spectrum and, instead, concentrate on three types of web-sites as explained next. However, recognizing the vast ranges that almost any web-server related parameter might cover, we shall try to emphasize the various bandwidth formulae and the parameters that go in them, rather than the numerical values. Section 6 does present some sample results, but their purpose is only illustrative. The three types of web-servers considered are as follows:

1. **SPECweb96:** This is a popular, but rather simplistic, benchmark for a native web-server. By *native*, we mean that the requested content is supplied locally, rather than being obtained from elsewhere.¹

1. A native server may redirect requests for certain content types to other physical servers of the site. This aspect is not addressed in this paper.

• K. Kant is with the Server Architecture Laboratory, Intel Corporation, Beaverton, OR 97006. E-mail: krishna.kant@intel.com.

• Y. Won is with the Division of Electrical and Computer Engineering, Hanyang University, Seoul, Korea. E-mail: yjwon@hanyang.ac.kr.

Manuscript received 15 May 1998; revised 1 Oct. 1998.

For information on obtaining reprints of this article, please send e-mail to: tkdc@computer.org, and reference IEEECS Log Number 109085.

SPECweb96 is simplistic since it assumes that all requests simply involve downloading a specific file from a static set of files. This benchmark also does not model detailed user behavior (burstiness, abandonments, retries) or network behavior (delays, packet loss, retransmissions, etc.). Because of its extreme simplicity, the bandwidth requirements of SPECweb96 are best regarded as lower bounds on those for native web servers. The primary motivation for considering SPECweb96 is the availability of detailed low-level measurements in a controlled environment, which are crucial for estimating the usage of various resources within a web-server.

2. **WebProxy:** This workload is intended to model a *firewall proxy server*, i.e., a server in an intranet environment that requires all accesses to native servers to go through it for security/protection reasons. Such a server needs to interact with both the client and the native servers. A proxy server may also provide caching of the contents retrieved from the native servers in order to reduce delays for frequently accessed contents.

Characteristics of this workload were obtained by examining HTTP logs for several internal proxy servers. Although these servers do not perform any caching, the workload is defined with caching as an important component. The workload is also defined to include dynamic content and the impact of user and wide-area network (WAN) characteristics.

3. **DP-server:** This stands for dynamic page server. Here, the workload is the same as for WebProxy, except that we consider it in the context of a native Web server instead of the proxy server. An important difference between this workload and SPECweb96 is the assumption of significant amount of dynamic content and impact of user/network characteristics.

Given the workload characteristics, we develop formulae for the bandwidth demands of the following system components: network interface cards (NICs), disk adapters, I/O-memory paths,² PCI buses, memory, and the processor interconnect, often known as front side bus (FSB) in bus-based systems. We also derive the number of resources (e.g., NICs, buses, channels) needed to satisfy the bandwidth demands and examine the associated connectivity issues. Because of the difficulty in analytically characterizing such aspects as cache miss ratios or coherency traffic on the processor bus, some of the numbers are necessarily derived from SPECweb96 measurements on a baseline system. This immediately raises the question of how realistic the given numbers are for a different hardware platform or O/S. Because of this very valid concern, the numerical values should be considered only for their

2. This refers to a direct path between the I/O controller and main memory, which avoids the use of processor interconnect for I/O.

illustrative value; the real use of the results in this paper should perhaps be the equations, along with measurements on the desired baseline system.

The key hardware components in a server are necessarily architecture dependent. Instead of considering specific architectures, here we concentrate on a symmetric multi-processing (SMP) based Web server, which are by far the most common. In a clustered environment, the server may include several SMP nodes, and the overall architecture may range from a CC-NUMA/COMA (cache-coherent nonuniform memory access/cache-only memory access) [12] to a loosely coupled network of nodes; however, we do not address cross-node traffic and, hence, do not need to make any assumptions regarding the overall architecture. We also assume a traditional bus-based node architecture since such an architecture involves the most coherency related traffic on the processor interconnect (snoop cycles, invalidations, implicit writebacks, and explicit writebacks). This should provide an upper bound on the most recent architectures using point-to-point links.

The main contribution of the paper is in the Web proxy traffic characterization and in examining the server bandwidth issues for various Web traffic environments. While the precise numerical values strongly depend on the assumed traffic characteristics and the platform under consideration, the general conclusions drawn here should be valuable in designing high-performance server for Web environments. Several large studies exist in open literature that analyze Web server logs for request/response size distributions and caching characteristics [1], [6], [3], [7], [8], [11]. Some of these studies have used the logs to drive a simulation of proxy server in order to better understand the caching behavior. Park et al. [17] have examined the impact of self-similarity in a TCP/IP based network performance. However, the use of traffic characteristics in engineering various resources of a web server is believed to be new.

The outline of the paper is as follows: Sections 2 and 3 characterize the workload for the SPECweb96 benchmark and the measured proxy servers. Section 4 discusses how to compute the used bandwidth of various resources for a given target throughput level. Section 5 discusses setting of rated utilizations, which is needed for estimating bandwidth requirements. Section 6 shows some sample numerical results. Finally, Section 7 makes some general observations about web server bandwidth demands and the follow-up work.

2 CHARACTERIZATION OF SPECWEB96 BENCHMARK WORKLOAD

In this section, we briefly characterize SPECweb96 benchmark [2], both to allow bandwidth computations and to contrast it with DP-server/WebProxy workloads discussed later. SPECweb96 models only the dominant transaction of a web server: serving requests from clients to download specific web pages or "files." Each client process repeats the following cycle: sleeps for some time, issues an HTTP GET request for a specific "file," waits for the complete file to be received, and then repeats the cycle. The server throughput is measured as the number of cycles (operations or

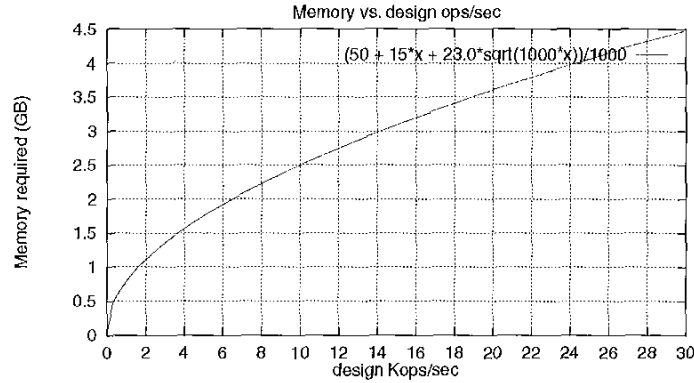


Fig. 1. Memory required vs. design trans/sec for SPECweb96.

transactions) per second. The response time experienced by a client from the server is also an important metric that must be reported, but no upper limit is placed on it.

SPECweb96 defines a static set of “files” served by the web server with a very simple structure: The web server works with a certain number, say D_{reqd} , of identical “directories,” each of which contains 36 files. The 36 files are divided into four classes, with nine files per class. Files within a class are referred to by a *file number* ranging from 0 to 8. A class refers to the order of magnitude of the file size: class 0 files are a few hundred bytes in size, class 1 files are a few KB in size, etc. Within each class, the nine files have equally spaced file sizes. That is, file no. i in class j has the size $S(i, j)$ given by $S(i, j) = 1.024(i + 1)10^{j+1}$ bytes, $i \in 1..9, j \in 1..4$. The total size of a directory works out to be about 5.12 MB ($M = 10^6$).

In SPECweb96, the server is sized for a target throughput (or design transactions/sec), henceforth denoted as λ_d . The load generation mechanism attempts to maintain this throughput so long as the server is not saturated. Consequently, for good benchmarking results, the achieved throughput λ_a is very close to the design throughput λ_d and there is no need to distinguish between the two. λ_d determines the number of directories that the server must service. In order to model the expectation that bigger web servers will perhaps handle a larger number of files, SPECweb96 requires the number of directories to increase as square root of the design throughput. In particular:

$$D_{reqd} = (\text{int})(0.5 + 10\sqrt{\lambda_d/5}) \quad (1)$$

Thus, the memory needed to fully cache all directories is given by $51\sqrt{\lambda_d/5}$ MB. The descriptors of the files stored in the file-cache may themselves be cached in a separate cache for quick access. Additional memory requirements include space for TCP control blocks (about 1 KB per active connection), SPECweb96 application, O/S, and required utilities. Considering these, the minimum memory required as a function of design throughput, denoted $\text{Mem}(\lambda_d)$, can be approximated as:

$$\text{Mem}(\lambda_d) = 50 + 0.015\lambda_d + 23.0\sqrt{\lambda_d} \quad (2)$$

Fig. 1 plots this function. It is seen that 4 GB memory is adequate to fully cache all files for the level of performance

expected in the near future. Consequently, most published SPECweb96 results are based on runs that fully cache all directories and, thus, completely hide any inefficiencies associated with disk I/O and file caching algorithms. (The only other disk I/O in SPECweb96 is logging; each client request results in writing a log entry of 128 bytes. Given buffered writes, the impact of logging is miniscule.)

In SPECweb96, requests are very simple and typically require about 70 bytes without the headers. Each request results in a file access, which is specified at three levels: *directory*, *class*, and *file* level. Directories are accessed with uniform distribution across all directories. The relative access probabilities for classes 0-3 are 35 percent, 50 percent, 14 percent, and 1 percent, respectively. That is, there is a strong preference for small files. The file number in each class follows a truncated Poisson distribution with mean of 4. That is, with

$$\sigma \triangleq 4,$$

the probability of accessing file no. i , denoted $P(i)$, is given by:

$$P(i) = \begin{cases} \frac{\sigma^i e^{-\sigma}}{i!} & \text{if } i < 8 \\ 1 - \sum_{j=0}^7 \frac{\sigma^j e^{-\sigma}}{j!} & \text{if } i = 8. \end{cases} \quad (3)$$

As expected from the properties of Poisson distribution, files 3 and 4 are accessed most often and the access frequency decreases for file numbers away from the mean value.

From this description, it follows that the average access size is 14.93 KB, but the median access size is only 3 KB. This indicates that maximum packet size of more than 3 KB is perhaps not worthwhile. The maximum file size of 912 KB is, however, much smaller than the hundreds of MBs observed in our measurements. Each request fits in one Ethernet packet and involves just one interrupt to the CPU. However, because of multiple outgoing packets per request, nonpiggybacked TCP acknowledgment packets will occur on the inbound direction (and only in the inbound direction). Thus, inbound traffic may be significant in terms of packet count, but negligible in terms of byte count. Also, each ack results in a processor interrupt.

SPECweb96 requests can be generated using one or more client machines. Typically, each client machine runs a

certain number, say N_p , client processes. If N_c is the number of client machines, the total number of client processes is $N_c N_p$. The *sleep time* of each of these client processes, say T_{sleep} , is computed as follows:

$$T_{\text{sleep}} = \frac{N_c N_p}{2\lambda_d} + 0.0005 \text{ secs.} \quad (4)$$

By Little's Law, the first term is one-half the cycle time of a client. That is, the intent is to divide the cycle time equally between the client sleep time and active time (which includes client side and server side response times). The second term merely provides a nonzero lower bound on the sleep delay.

The major shortcoming of SPECweb96 is a static model where files are predefined and never change, quite unlike reality where, in addition to frequent modifications to stored documents, a significant percentage of the responses may have data generated on the fly by CGI scripts, which doesn't even fit the stored-file concept. Another problem is a very limited set of file sizes and access frequencies (only 36 distinct values) and a very regular arrangement (e.g., same number of files of each of the 36 sizes, and the same set of access probabilities for them). Uniform access across directories also does not properly model the tremendous access skew seen in real web-servers. Yet another problem is that the workload says nothing of issues like user abandonments/timeouts and retries.

In this paper, we partially address these issues by considering a SPECweb96-like workload where each file is modified with certain probability between successive accesses to it, thereby forcing the file to be read from the disk with some probability over and above the disk reads required due to inadequate memory. We also assume a certain probability of abandonment and subsequent retry that results in wasted resources. As stated before, the primary reason to build on SPECweb96 as a base is the availability of detailed low-level measurements for that benchmark.

3 CHARACTERIZATION OF PROXY WEB TRAFFIC

To characterize this traffic, we examined data from a live proxy Web server recorded in the *extended log format*. The analysis presented here is constrained by the kind of information provided by the extended log format and its limitations. The most relevant information consists of 1) name-server entry or IP address of both client and server, 2) time of request, 3) request and response sizes, and 4) total transfer time in seconds (remote server to proxy). Unfortunately, the time resolution is only 1 second, which masks out all small time-scale behavior. The following sections present analysis of the collected data.

3.1 General Observations

The collected data showed that two operations, GET and POST, accounted for almost all of the accesses. The POST operation characterizes a query to a web-server to locate the desired information. We found that almost 97 percent of the operations are GETs, and POST sizes do not seem to be very different from the sizes of GET requests. Therefore, both in the data analysis and in the modeling, POSTs are

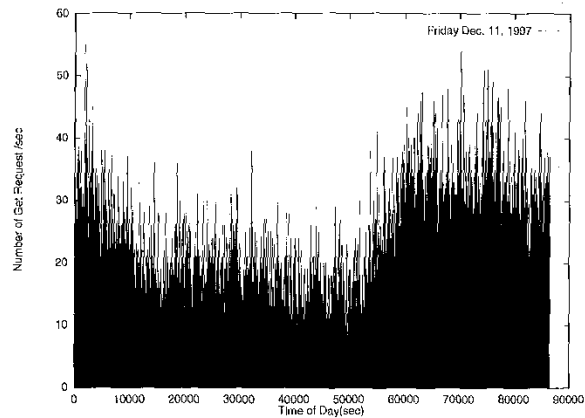


Fig. 2. Typical weekday traffic profile.

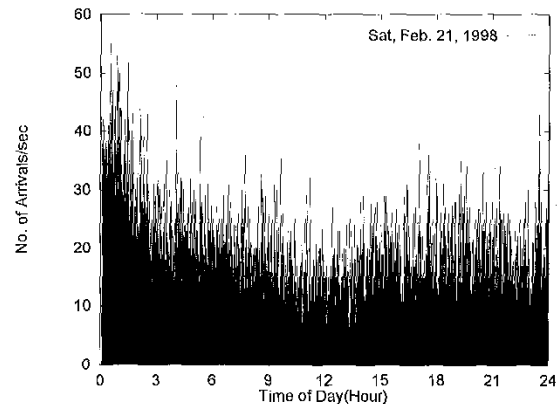


Fig. 3. Typical Saturday traffic profile.

not explicitly distinguished from other requests. We also examined the "network load," defined as the total traffic on the wire (both in terms of bytes and packets). It is found that the network load and arrival processes are very similar and, thus, it suffices to concentrate on the latter.

3.2 Request Arrival Process

This process shows a characteristic daily variation, as can be seen from Fig. 2 and Fig. 3, which show the number of arrivals per second for a weekday and weekend as a function of wall clock time (in seconds). During weekday, the traffic is highest from 4:30 p.m. to 10:10 p.m. and can be assumed to be reasonably stationary during this period.³ The data collected over several weekdays shows that the daily traffic profile is remarkably stable. (The weekend traffic is much lower and was ignored).

A high traffic profile occurring at the tail end of a typical workday may be surprising and is perhaps a result of the fact that the concerned web server is a *proxy* server. The *proxy* server acts as a gateway between intranet and internet and, thus, must handle all traffic that crosses the firewall boundary. The outgoing traffic would be expected

3. The traditional busy Internet period, from a Telco perspective, is typically between 9–11 p.m.; however, there the measure used is the trunk load in erlangs, rather than the arrival rate.

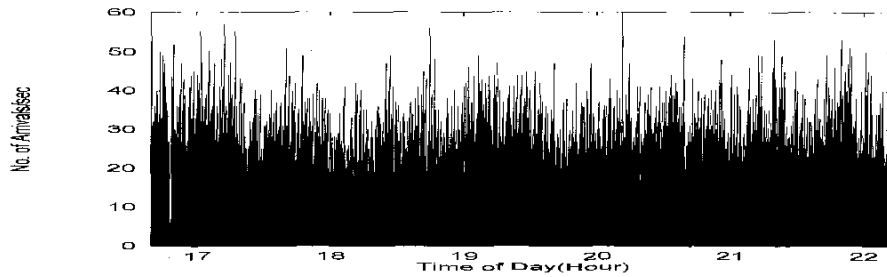


Fig. 4. Busy period arrival process: Friday, Feb. 20, 1998. Time unit = 1 sec.

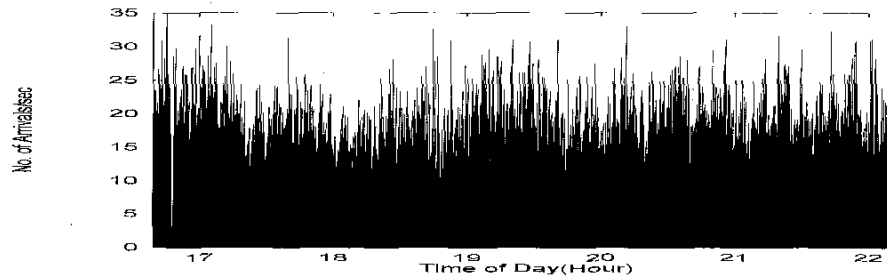


Fig. 5. Busy period arrival process: Friday, Feb. 20, 1998. Time unit = 10 sec.

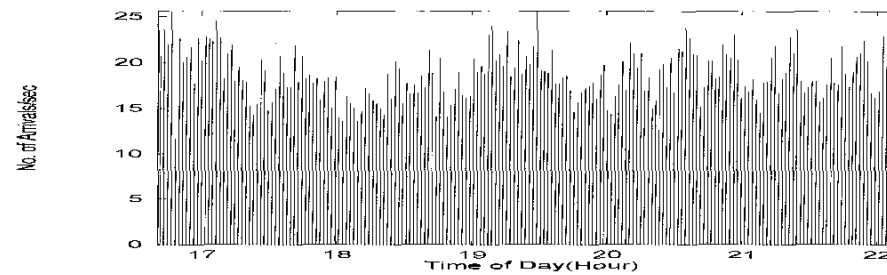


Fig. 6. Busy period arrival process: Friday, Feb. 20, 1998. Time unit = 100 sec.

to increase significantly at the end of the day and, hence, the observed behavior. This conclusion is corroborated by examining the weekend traffic, which does not show any “camel back” features, as shown in Fig. 3. This traffic shows heavy arrival traffic during early morning hours, but this is essentially a Friday phenomenon.

In analyzing this data, we examined both the distribution of number of arrivals per second, as well as correlations between arrival counts in successive seconds. The interarrival time distribution appears pretty tame—it is reasonably approximated by a branching Erlang distribution with coefficient of variation of about 0.5. However, the autocorrelation function decreases rather slowly with the lag, thereby suggesting the presence of long-term correlations or self-similarity. This is illustrated by plotting the arrival rate process at increasing granularity. In Fig. 4, the arrival rate is the average computed over successive 1 sec intervals. In Fig. 5, the arrival rate is the average over successive 10 second intervals and, in Fig. 6, it is over successive 100 second intervals. It can be seen that the three plots look similar, i.e., an aggregation over 3 orders of magnitude of time scale does not significantly

smooth out the time series. This is a clear indication of self-similarity property [19].

A self-similar process can be formally defined as follows: Let $X_t, t=1,2,3,\dots$, denote a covariance stationary stochastic process with mean μ , variance σ^2 , and autocorrelation function:

$$r_k \triangleq E[(X_t - \mu)(X_{t-k} - \mu)]$$

for $k=1,2,\dots$. For the purposes of this paper, X_t can be considered as the arrival counting process that gives the number of arrivals during the last 1 second interval. Also assume that $r(k)$ is a slowly decreasing function of k for $k \rightarrow \infty$. In particular, assume that:

$$r(k) = O(k^{-\beta}) \quad \text{for } k \rightarrow \infty, 0 < \beta < 1. \quad (5)$$

That is, $r(k)$ decreases slower than hyperbolically. Now, divide X into nonoverlapping blocks of size m and construct the aggregated process $X_j^{(m)}, j=1,2,\dots$ where $X_j^{(m)}$ is the arithmetic average of the j th block. Then, X is called exactly self-similar if X and $X^{(m)}$ have identical autocorrelation structure, i.e., $r^{(m)}(k) = r(k)$ for all m and all

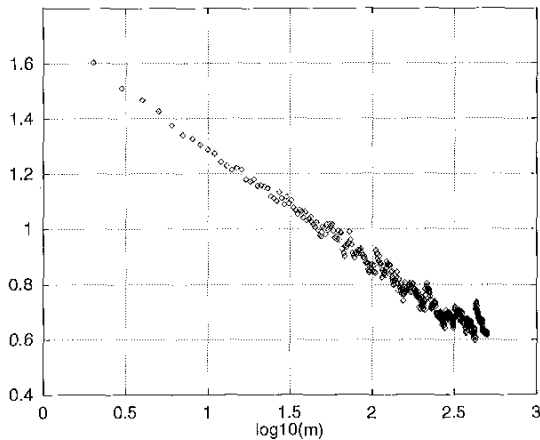


Fig. 7. Variance time plot: request arrival process.

k . That is, X and $X^{(m)}$ are indistinguishable in terms of their second order properties. For such a process, the *Hurst parameter* H is defined as:

$$H \triangleq 1 - \beta/2.$$

Note that $0.5 < H < 1$. A larger H means a slower decay for $r(k)$ and, hence, longer persistence of correlations. Traditional (i.e., short-range dependent) processes have $H = 0.5$ and their autocorrelation functions decay exponentially fast. This fundamental difference makes self-similar processes qualitatively different from the traditional Markovian arrival processes.

There are several techniques to ascertain self-similarity of a process [19]; here, we only use the *variance time plot* technique, which is based on equation (5). That is, if the process is self-similar, a plot of $\log(\text{Var}(X^{(m)}))$ against $\log(m)$ (time scale) will be approximately linear with slope β . Fig. 7 shows this for the Feb. 20 data. According to this plot, the arrival process is highly self-similar with $H = 0.8$. The estimate of H was found to be remarkably stable over the seven weekdays worth of data that we analyzed. This result is not surprising since self-similarity seems pervasive in virtually all of high speed network traffic including world wide web traffic (e.g., see [19], [6], [18]).

For the purposes of this paper, the main impact of self-similarity is on queuing delays and, hence, on the maximum utilization levels at which resources should be operated. This issue is discussed in Section 5. Self-similarity also makes caching more difficult (both in the context of proxy web cache and file-cache on the server), but it is difficult to relate the H parameter directly to the caching behavior.

It may be noted here that the SPECweb96 benchmark does not attempt to generate self-similar traffic; however, it is possible to introduce self-similarity in that workload by making the sleep time and active times (duration for which requests are made without any sleep) heavy-tailed [20].

TABLE 1
Some Percentile Values of File Size Distribution (in KBytes)

quantile	Data for days 1..7						
95.0%	21.2	21.5	22.1	22.0	23.9	23.6	20.5
99.0%	53.0	50.3	56.4	60.3	64.0	64.1	53.0
99.9%	1255	501	1024	1297	834	894	591

3.3 Request and Response Size Processes

Average request size in the proxy server workload was found to be about three times that for SPECweb96 (330 bytes vs. about 110 bytes for SPECweb96). Also, request sizes cover a wide range from about 60 bytes to 16 KB, although less than 0.1 percent requests need more than one Ethernet packet. Consequently, the standard deviation of request sizes is small at about 116 bytes. In contrast, SPECweb96 requests are almost constant in size. The request size process is not self-similar, but does show small short-term correlations.

Since the predominant HTTP operation is GET, the server response typically refers to the web page or "file" that the client wants to download. Table 1 shows some quantiles of file-size distribution over seven typical weekdays. It is found that 95 percent of the responses are less than 24 KB, and 99 percent of the responses are less than 64 KB. Also, the response size distribution during the busy period of 4:30 p.m. to 10:10 p.m. is approximately the same as for the entire 24-hour period. Note that, unlike SPECweb96, about 0.1 percent of the files exceed 1 MB, with maximum file size in excess of 200 MB. Fig. 9 compares response size distribution of our data against Pareto distribution with $\alpha = 0.3, 0.5, 1.0$ given by,

$$P(X > x) = \left(\frac{k}{x}\right)^\alpha$$

where k is a distributional parameter. Evidently, Pareto distribution does not properly capture the response size distribution of web traffic for smaller file size range (less than 10 KByte). A lognormal distribution can more precisely explain the observed distribution.

We note here that the response size distribution could heavily depend upon the type of content served by the server or the environment in which it is operating (e.g., academia, industry, e-commerce, etc.). In this study, we do not cover file access pattern over various systems. A number of articles, including [6], have investigated the characteristics of file size distribution in a web environment. Consistent with these studies, we find that the file size distribution is fairly heavy-tailed, however, it doesn't appear to follow the Pareto distribution. However, a lognormal distribution can accurately explain the observed distribution.

For the purposes of bandwidth calculation, we simply assume average request size of 330 bytes for WebProxy, with the request always contained in one Ethernet packet. Even though our data shows that WebProxy file sizes are smaller than those assumed in SPECweb96 (11,000 bytes vs. 14,926 bytes), we decided to retain the same average file size as SPECweb96 in order to 1) better highlight the impact of other characteristics, such as longer tails in the distribution,

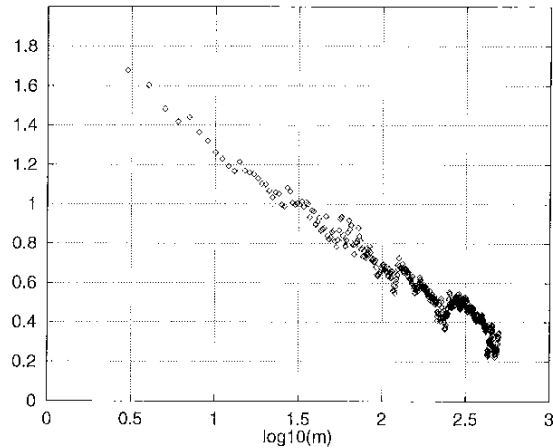


Fig. 8. Variance time plot: transfer time.

and 2) to allow for some “headroom” since file-sizes tend to differ widely across web servers. For caching and latency considerations, we extend the tail of the distribution in SPECweb96 framework by modeling them as seven file classes (instead of four), which extend from 10 bytes to 100 MB. The class probabilities were chosen as (0.10, 0.25, 0.514, 0.132, $3.7e-3$, $0.25e-3$, $0.05e-3$). The tail extension with the same mean results in more mass at smaller sizes and thereby results in more partially full network packets and disk blocks.

In addition to the response size, we also examined the transfer time. It was found that transfer time covers a very wide range, from 0 secs to 5,000 secs, although 45 percent of the requests registered 0 seconds transfer time (which really means a transfer time < 0.5 secs). The remaining 55 percent requests have a mean transfer time of 6.2 secs and standard deviation of 50 secs. This metric is perhaps not reliable since, in case of streaming audio or video, the transfer time would indicate the duration of the entire transfer, even though this is perhaps not a useful delay measure. Transfer time process is strongly correlated with file size process and has daily variation similar to that for the arrival process. It is found to be self-similar with $H = 0.717$, as seen from Fig. 8. However, unlike the arrival process, the impact of self-similarity in transfer time on the engineering of the web server is not clear and has been ignored in this paper.

3.4 Remote Server Access

One major function of a Web proxy is to cache the requested “files” and, thereby, minimize going out to a remote server for repeated requests to the same content. Not all files are considered cacheable however; files that represent dynamically constructed contents via CGI scripts or those that change very frequently would mostly result in caching overhead and, thus, may be marked by the origin server as noncacheable. In addition, certain web sites may expressly prohibit caching of all or parts of their contents by marking it as uncacheable. It is likely that the fraction of noncacheable information increases over time as content providers become more sensitive about the proxies storing or massaging their contents. We, henceforth, denote the fraction of uncacheable bytes as α_{nch} , and the uncacheable

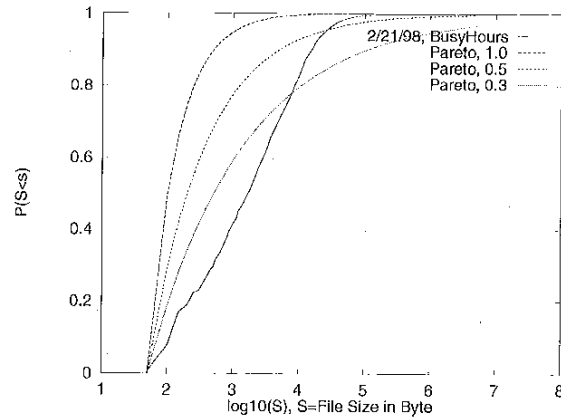


Fig. 9. Response size distribution and Pareto distribution.

file fraction as α_{nch} . Since CGI generated data is usually rather small in size, $\alpha_{nch} < \alpha_{nch}$ typically.

The hit ratio for the cacheable information is obviously a function of the available storage; however, even with a fairly large available storage at the proxy server, high hit ratios are difficult to achieve because of a number of reasons:

1. *Highly skewed access pattern:* It is well-known that file access pattern often follows a Zipf-like pattern where a small fraction (say 5 percent) of the entire set of accessed files may account for 80 percent of the accesses. Although this facilitates caching in that a small cache can hold most of the frequently accessed files, it also implies that the unpopular files are very unpopular and will appear to be accessed one time only (during the time window relevant for caching). This also implies that the marginal utility of increasing the cache size (which could be very large even at 5 percent caching level) may be very small.
2. *Highly bursty accesses:* The observed self-similarity of the arrival process indicates that the accesses may experience large bursts followed by long lull periods which makes caching difficult.
3. *Frequent modification of popular content:* Recent studies [8] have confirmed the expected behavior that more popular files change more frequently. This correlation results in much poorer caching behavior than for random modifications.
4. *Suboptimal replacement policy:* Most caches currently use LRU (least-recently used) replacement policy, which is not particularly good in the web context because of the wide range of file sizes involved and rather irregular access pattern. Several other policies have been devised and shown to work much better [1], but are currently not implemented.
5. *Coherency Issues:* For scalability, it is the proxy, rather than the origin server, that is responsible for keeping the copy current. This is done via expiration or refresh interval along with conditional GET requests. The coherency protocol may itself lead to some unnecessary reload or “false misses.”

In view of these factors, it is not surprising that the maximum achievable hit ratios are found to be rather low. For example, the study in [1] suggests maximum cacheability at 65 percent, and the study in [11] indicates that cookies are sent with 30 percent of the responses, which indicates that at least 30 percent of responses are uncacheable unless delta encoding is used [15]. The study in [8] finds that 50 percent of the references are "one-timers" and, hence, may not be worth caching. The combined impact of these factors is that even achieving a 50 percent cache hit ratio may be very difficult in practice. The detailed study in [7] indicates that the achievable cache ratio is typically between 24 percent and 45 percent, with false miss ratios in 2-7 percent range.

The proxy servers that we examined completely disabled any caching because caching was not found to reduce bandwidth or latency significantly. This is not all that surprising because of rather low cache hit ratios, additional delays introduced by proxy, and user abort/retry not being propagated to the native server directly [3]. Nevertheless, we shall consider the more general situation where the proxy does perform caching.

Let q_{hb} denote the proxy cache hit ratio, defined as the fraction of *cacheable* bytes that are delivered out of the proxy cache. Let q_{hf} denote the corresponding file-count measure, i.e., the fraction of files found in the proxy cache. (The discussion above referred to q_{hf} .) Both measures are important for a proxy server: q_{hf} is important from the perspective of transaction rate that the proxy server can handle, whereas q_{hb} is important from the network bandwidth perspective. Let S_{ca} and S_a denote, respectively, the average cached access size and average overall access size. Then, $q_{hb} = q_{hf}S_{ca}/S_a$.

Typically, small files are more frequently accessed than large ones, which would suggest that $S_{ca} < S_a$ and, hence, $q_{hb} < q_{hf}$. On the other side, however, more popular files are more frequently modified which makes them less likely to be obtained from the cache—in fact, the uncacheable dynamic files are typically very small in size. We believe that analytic expressions for S_a and S_{ca} are unlikely to be useful in practice; therefore, we work with q_{hb} and q_{hf} directly, assuming that their ratio is available from measurements.

3.5 Disk Access Characteristics

At a proxy server, the available disk space (rather than the main memory space) defines how much data can be made available locally; therefore, the proxy cache size refers to the available disk space only. That is, all cached files must reside on the local disk subsystem of the proxy server, although performance considerations often dictate that frequently accessed files be cached in the main memory. We assume that this is done via the normal file-caching mechanism provided by the O/S. The main consequence of this scheme is that all remotely accessed cacheable files must be eventually written to the disk unless they are either modified or evicted from the proxy cache before the disk write actually happens. Let T_w denote the amount of time a remotely retrieved file is allowed to sit in the file-cache before being written out to the disk. Since a large T_w (exceeding a few minutes or so) is undesirable from a

reliability perspective, it is safe to assume that no eviction from proxy cache will occur during the time T_w . (The file may, however, get evicted from the file-cache before time T_w due to lack of references, but such an eviction will only force an immediate write to the disk.) Thus, the only situation where the file never makes it to the disk is that it gets modified before it gets a chance to be written. We denote this probability as α_{mod} .

Disk reads are driven by the file-cache miss ratio. Because of typical user preference for smaller files, the average access size for disk resident files is expected to be larger than that for memory resident files. We avoid the complexity of estimating these sizes by working with accesses in terms of bytes. We denote by q_{fc_mr} , the fraction of locally resident data (in bytes) that is delivered from the disk.

4 COMPUTING BANDWIDTH USAGE

The bandwidths considered in the calculation are *Network Bandwidth, Disk I/O bandwidth, PCI bandwidth, I/O-memory path bandwidth, memory bandwidth, and processor data bus or FSB bandwidth*. Let B_{ntwk} , B_{disk} , B_{pci} , B_{iom} , B_{mem} , and B_{fsh} denote, respectively, the per-transaction-used bandwidth for network, disk I/O, PCI bus, I/O-memory path, memory, and FSB resources. In the following, we develop equations for these bandwidths for the WebProxy workload only since the corresponding SPECweb96/DP-server equations are mere special cases of these. We start with send, receive, and total network bandwidth in bytes/sec, denoted B_{send} , B_{recv} , and B_{ntwk} , respectively.⁴ These can be obtained by simply adding the required bandwidths over proxy to client and proxy to remote server connections. Noting that a client request may occasionally require forwarding the request to a remote server and obtaining the response before data is sent to the client, we have:

$$B_{send} = [I_{resp} + q_{rrf}(L_{req} + A_{recv}) + (M_{send} + q_{rrf}M_{recv})\theta_{ka}](1 + n_{rex}) \quad (6)$$

$$B_{recv} = [(L_{req} + A_{recv} + q_{rrf}L_{resp}) + (q_{rrf}M_{send} + M_{recv})\theta_{ka}](1 + n_{rex}), \quad (7)$$

and

$B_{ntwk} = B_{send} + B_{recv}$. The notations used in these equations have the following interpretation:

1. L_{req} : Average size of a client request in bytes. For SPECweb96, $L_{req} = 110$, and, for WebProxy, $L_{req} = 330$. The number of packets per request, denoted N_{recv} , is 1 for SPECweb96, and is assumed to be the same for the other two workloads.
2. L_{resp} : Average size of a server response in bytes. $L_{resp} = I_{send}N_{send}$ where I_{send} is the average send packet size (including headers) and N_{send} is the average number of send packets per file. Assuming Gigabit Ethernet medium with a maximum of 1,500

4. Throughout this paper, "send" and "recv" are taken from Proxy's perspective.

byte packets, TCP/IP header of 44 bytes, 4 byte LLC header, and a maximum LAN header of 20 bytes, we get $N_{\text{send}} = 11.0$ for SPECweb96, and $N_{\text{send}} = 11.03$ for the other two workloads. In both cases, $L_{\text{send}} = 1,432$.

3. A_{recv} : Explicit acknowledgment overhead, apportioned on a per transaction basis for network receives. In TCP, every packet needs to be acknowledged. TCP allows block acknowledgment whereby the last w_{recv} packets can be acknowledged by a single "ack." By default, $w_{\text{recv}} = 2$. Typically, "acks" are piggybacked onto regular messages in the other direction; however, when the traffic is asymmetric, explicit acknowledgment messages become necessary. The average number of explicit acks in the incoming direction, denoted $N_{\text{exp_ack}}$, is given by $N_{\text{exp_ack}} = (N_{\text{send}} - N_{\text{recv}})/w_{\text{recv}}$.

Let $L_{\text{min}} = 44$ bytes denote the smallest message size at TCP/IP level. Then, $A_{\text{recv}} = L_{\text{min}}N_{\text{exp_ack}}$. The explicit ack overhead on the send side, i.e., A_{send} is negligible and assumed to be zero.

4. M_{recv} : Management overhead, apportioned on a per transaction basis for network receives. This includes dynamic connection set-up and teardown, and TCP flow control. We assume four recv packets per connection (SYN, SYN-ACK, FIN, and FIN-ACK), each L_{min} bytes long (TCP/IP header, LLC header, and MAC header).
5. M_{send} : Management overhead, apportioned on a per transaction basis for network sends. This includes dynamic connection setup/teardown and TCP flow control. We assume two send packets per connection (send SYN, send FIN), each L_{min} bytes long.
6. q_{rrb} : Remote receive fraction in bytes, i.e., fraction of response data per request, in bytes, that is received from a remote (native) web server. From the definition of α_{ncb} and q_{hb} in Section 3.4, it follows:

$$q_{\text{rrb}} = \alpha_{\text{ncb}} + (1 - \alpha_{\text{ncb}})(1 - q_{\text{hb}}) \quad (8)$$

7. q_{rrf} : Remote receive fraction in terms of file-counts. From (8), we have:

$$q_{\text{rrf}} = \alpha_{\text{ncf}} + (1 - \alpha_{\text{ncf}})(1 - q_{\text{hf}}) \quad (9)$$

Note that we use q_{rrf} for both the data and acks. The use of q_{rrf} for acks is only an approximation; the true value depends on the size distribution of remotely received files, which could be different from the overall file-size distribution. The true value should lie between q_{rrf} and q_{rrb} .

8. n_{rex} : Average number of retransmissions that a message (connection setup, client request, or server response) may suffer due to transmission errors or large delays in acknowledgments.⁵
9. θ_{ka} : Probability of not keeping the connection alive. With HTTP1.1, the same TCP connection can be reused for successive interactions between the same peers.

5. Packet loss may substantially increase the user response time due to long end-to-end delays and TCP flow-control mechanism, but that aspect is not relevant here.

In addition to the raw byte bandwidth, we also need bandwidth in transmission units or packets, since many processing costs (interrupt handling, header formatting, segmentation/reassembly, control overhead at the NIC, etc.) depend on the number of packets, rather than on the actual number of bytes. We, henceforth, use the symbol T (instead of B) for bandwidth in transmission units. Thus, (6) and (7) can be written as:

$$T_{\text{send}} = [N_{\text{send}} + q_{\text{rrf}}(N_{\text{recv}} + N_{\text{exp_ack}}) + (2 + 4q_{\text{rrf}})\theta_{\text{ka}}](1 + n_{\text{rex}}) \quad (10)$$

$$T_{\text{recv}} = [N_{\text{recv}} + N_{\text{exp_ack}} + q_{\text{rrb}}N_{\text{send}} + (2q_{\text{rrf}} + 4)\theta_{\text{ka}}](1 + n_{\text{rex}}), \quad (11)$$

and $T_{\text{itwk}} = T_{\text{send}} + T_{\text{recv}}$. Again, the use of q_{rrf} in these equations is an approximation. The real fraction would lie between q_{rrf} and q_{rrb} .

User abandonments and retries involve wasted work at almost all server resources. We denote the user retry probability as q_{retry} and account for its impact by increasing the effective load by the factor $1/(1 - q_{\text{retry}})$. (This is somewhat pessimistic since not all operations may need to be performed for abandoned requests.)

For the disk bandwidth in bytes/sec, we let:

$$B_{\text{disk}} = B_{\text{disk,r}} + B_{\text{disk,w}},$$

where $B_{\text{disk,r}}$ and $B_{\text{disk,w}}$ are disk read and disk write bandwidths per transaction. Assuming a sequential disk transfer size of L_{xfr} bytes, the average number of disk blocks per file, denoted N_{blks} , can be computed for the assumed file-size distribution. Typically, L_{xfr} will be some integer multiple of physical disk block size (e.g., 16 KB *logical block* or maximum sequential transfer size and 4 KB physical disk block size) and the file storage scheme would ensure that the entire transfer size can be accessed without significant rotational/seek delay. Then, the average disk read bandwidth per transaction is given by:

$$B_{\text{disk,r}} = L_{\text{xfr}}N_{\text{blks}}[(1 - \alpha_{\text{ncb}})q_{\text{hb}}]q_{\text{tc_mr}}, \quad (12)$$

where the expression within brackets gives the fraction of cacheable data that is available locally at the proxy server and $q_{\text{tc_mr}}$ gives the probability that this data resides on the disk.

Note that this bandwidth applies only to the steady state operation. The initial loading of the file-cache may require much higher bandwidth. In particular, for native web servers with a fairly large file-cache, the steady state disk bandwidth may be very small (usually a result of highly skewed access pattern), but adequate bandwidth should still be available for initial loading and O/S initiated cache flushes, if any. To handle this, we define a parameter $q_{\text{min_atk}}$ which sets a minimum per transaction disk read probability. Accordingly, for computing disk bandwidth requirements, we let:

$$B'_{\text{disk,r}} = L_{\text{xfr}}N_{\text{blks}} \max(q_{\text{tc_mr}}(1 - \alpha_{\text{ncb}})q_{\text{hb}}, q_{\text{min_atk}}). \quad (13)$$

In situations where $B'_{\text{disk,r}} > B_{\text{disk,r}}$, the additional bandwidth is needed only for initial or occasional file loading purposes. Therefore, we use $B'_{\text{disk,r}}$ only for appropriately inflating I/O bandwidth, and continue to use $B_{\text{disk,r}}$ for all other purposes,

including the computation of memory and FSB bandwidths. The disk write bandwidth per transaction is given by:

$$B_{\text{disk,w}} = L_{\text{log}} + L_{\text{xfr}} N_{\text{blks}} [(1 - \alpha_{\text{reb}})(1 - q_{\text{hb}})] (1 - \alpha_{\text{mod}}), \quad (14)$$

where L_{log} is the number of bytes of log per transaction. The expression in square bracket gives the fraction of cacheable data that obtained from the remote server, and $1 - \alpha_{\text{mod}}$ is the probability that the data is actually gets a chance to be written out to the disk before being modified.

As with network I/O, we also need to consider disk I/O bandwidth in terms of I/O operations per second. Since log writes are also done lazily in complete transfer sizes, the bandwidth in transfers/sec (again denoted by substituting the symbol T for B) can be obtained as $T_j = B_j/L_{\text{xfr}}$ for $j = (\text{disk,r}), (\text{disk,w}), \text{disk}$.

Let Δ_{nic} and Δ_{da} denote the average PCI burst sizes in bytes for NICs and disk adapters, respectively. Typically, burst sizes range from 32 bytes to 256 bytes. We assume six cycles for the address part of PCI access.⁶ In addition, depending on the details of the adapter and how it is set up, a number of PCI cycles are needed for setting up and completing the transfer. When multiple adapters sit on the same PCI bus, one or more unsuccessful retries may also occur. We assume a total of $N_{\text{ctrl}} = 26$ cycles for this overhead, which again could be optimistic or pessimistic depending on the adapter and configuration details. Altogether, the overhead per transfer is 32 cycles per burst. In the data phase, a 32-bit PCI bus transfers $n_t = 4$ bytes/cycle, and 64-bit PCI bus transfers $n_t = 8$ bytes/cycle. The resulting bandwidth inflation factors for NICs and disk adapters, denoted α_{nic} and α_{da} , respectively, are given by $\alpha_{\text{nic}} = 1 + 32n_t/\Delta_{\text{nic}}$ and $\alpha_{\text{da}} = 1 + 32n_t/\Delta_{\text{da}}$. Therefore, the overall PCI bandwidth is given by:

$$B_{\text{pci}} = \alpha_{\text{nic}}(B_{\text{recv}} + B_{\text{send}}) + \alpha_{\text{da}}(B'_{\text{disk,r}} + B_{\text{disk,w}}) \quad (15)$$

The data traffic on I/O-memory path and PCI buses is merely the sum of the network and disk I/O traffic (including control traffic involved in DMA setup). However, we thought it more reasonable to use the bandwidth B' (instead of B) for disk reads in this calculation. The I/O-memory path has its own transfer protocol, which results in some overhead due to header/trailer for each transfer, henceforth denoted as $\alpha_{\text{iom,r}}$ ($\alpha_{\text{iom,w}}$) for inbound reads and writes, respectively. The data parts of the bandwidth are easily seen as $\alpha_{\text{iom,r}}(B_{\text{send}} + B_{\text{disk,w}})$ for inbound reads and $\alpha_{\text{iom,w}}(B_{\text{recv}} + B'_{\text{disk,r}})$ for inbound writes, respectively. The control part can be computed by estimating the number of PCI bursts per transaction and converting N_{ctrl} from cycles to number of bytes transferred. For brevity, we omit detailed equations here [14].

The required memory and data bus bandwidths depend on the processor cache size and the amount of memory-to-memory copying involved in various I/O operations. With respect to cache size, we consider the following two cases:

1. **Large cache:** Results for this case are obtained assuming a 2 MB cache for a system providing about 4,275 trans/sec throughput. This case is intended to cover systems with large on-chip L2 caches or large on-board (L3) caches but only small L2 caches.
2. **Small cache:** The results here are obtained assuming a much smaller (512 KB) cache for approximately the same performance level (3,900 trans/sec). Small caches may be attractive for cost reasons or because their impact on CPU stalls is compensated by significantly lower latencies.

An important point to note here is that, as the performance level increases, so does the amount of data (files to be handled). For SPECweb96, this increase is roughly proportional to the square root of the throughput. The increase in amount of data means that there is less cache space available per unit of data. That is, a CPU with 2 MB cache operating at 17,100 trans/sec would be about equal to a CPU with only 1 MB of cache at 4,275 trans/sec. Put another way, given a baseline system with 2 MB cache at 4,275 trans/sec, the effective cache size to consider at 17,100 trans/sec is only 1 MB. Thus, if the cache sizes on future (higher performance) systems do not scale, the performance will suffer because of smaller effective cache size. This is yet another reason for considering the small cache case.

Current operating systems (e.g., Solaris2.6) support 0-copy sends, i.e., to send out user data over the network, no copying is necessary from user space to Kernel space; instead, the data is directly placed from user space on to the I/O bus for delivery to the NIC. (NT4.0 O/S uses 0-copy sends only for static files; dynamic file sends are usual 1-copy sends.) The virtual interface architecture (VIA) [9] completely bypasses O/S kernel for all I/O and, thus, all I/O becomes 0-copy. Accordingly, we consider the following two cases:

1. **0-copy send:** Network sends are 0-copy for both static and dynamic files. Other I/O operations involve a single copy between user and kernel address spaces. This represents the current mode of operation for most operating systems.
2. **0-copy I/O:** All I/O (network sends, network receives, disk reads, and disk writes) are 0-copy. Note that 0-copy receives imply TCP checksums in the NIC itself.

For memory and data bus bandwidth, the parameter of interest is the number of memory references per transaction. We divide these into cache miss ratio dependent (cmr-dep) and cache miss ratio independent (cmr-ind) references. As a simple approximation, the latter class includes the number of data bytes involved in PCI-side data transfer and host-side memory to memory copies.⁷ All other references are

6. The overhead of an initial address cycle is only three cycles; however, inbound reads are always deferred and require a retry, which costs an additional three cycles. In all the workloads considered here, most PCI operations are inbound reads and, thus, an average of six cycles of address overhead is reasonable.

7. Although memory-to-memory copies typically do go through the cache (as opposed to being done by a separate copy engine) and, thus, affect the cache miss ratio, that's not relevant here. The point is that the bus/memory traffic resulting from this copying itself is not affected by the cache miss ratio.

lumped as cmr-dep type. The cmr-dep references for 0-copy send case can be estimated from measurements for a baseline case.

Cache miss ratio independent memory references can be related to the bandwidth parameters already computed. A basic 0-copy operation involves only PCI-side memory reads (for network sends and disk writes) or memory writes (for network receives and disk reads), whereas a 1-copy operation also involves host-side memory read followed by a memory write. It follows that, for 0-copy sends (i.e., 0-copy used only for sends):

$$B_{\text{nd},r} = B_{\text{recv}} + B_{\text{disk},r} + B_{\text{send}} + 2B_{\text{disk},w} \quad (16)$$

$$B_{\text{nd},w} = 2B_{\text{recv}} + 2B_{\text{disk},r} + B_{\text{disk},w}. \quad (17)$$

Let $B_{\text{dep_read},j}$ and $B_{\text{dep_write},j}$ denote, respectively, the cache miss ratio dependent memory read and write references for the modeled system. Here, j refers to whether we are talking about the large cache case ($j = lc$) or small cache case ($j = sc$). For the baseline system, these parameters were obtained from measurements, and are denoted as $B_{\text{dep_read},j}^{(b)}$ and $B_{\text{dep_write},j}^{(b)}$ respectively.

An accurate scaling of these parameters for a nonbaseline system (i.e., $B_{\text{dep_read},j}$ and $B_{\text{dep_write},j}$) is very difficult even for the same workload (SPECweb96) and nearly impossible for a very different workload such as WebProxy. In particular, memory references are determined by miss ratios in the entire cache hierarchy and by the writeback traffic on the bus. These, in turn, depend on the fine-grain details of the workload, network stack, O/S, locking mechanisms, processor architecture, and chip-set architecture. Based on detailed SPECweb96 measurements, we have developed a scheme that allows scaling with respect to number of CPUs, design op-count, cache-size, and architectural features and appears to work well (provided that we are still dealing with SPECweb96). For simplicity, in this paper, we continue to keep the number of CPUs and architectural features the same as in the baseline system (described in Section 6) and discuss scaling with respect to only the cache-size, design op-count, and additional I/O in DP-server and WebProxy workloads.

An increase in cache-size lowers the miss ratio. For a range of cache-sizes, the effect is approximately linear in a log-log plane, i.e., doubling the cache size reduces the miss ratio by some constant multiplicative factor δ . However, for small caches (< 256 KB) or very large caches (> 1.5 MB), the effect is nonlinear. In the linear region, the miss ratio for cache size L , denoted $MR(L)$ can be related to the baseline value $MR_0(L_0)$ as:

$$MR(L) = MR(L_0)\delta^{L_0/L} = \left(\frac{L_0}{L}\right)^\beta \text{ where } \beta = -\log_2(\delta). \quad (18)$$

As discussed in the next paragraph, it is the *effective*, rather than the *actual*, cache size that is relevant here; therefore, a more elaborate scaling that considers the nonlinear region as well is needed, even with actual cache size in the 0.25-1.50 MB range. For the purposes of this paper, however, we assume that $\beta = \log_2(0.87) = 0.201$ for large cache, and $\beta = \log_2(0.82) = 0.286$ for small cache.

As discussed earlier, an increase in design trans/sec decreases the effective cache size roughly according to the required data space. By simply using the total required memory space instead of the data space only, we can estimate the impact of higher trans/sec on cache-dependent memory references as follows:

$$B_{\text{mem,dep},j} = B_{\text{mem,dep},j}^{(b)} \left[\text{Mem}(\lambda_d) / \text{Mem}(\lambda_{d,j}^{(b)}) \right]^\beta, j = lc, sc \quad (19)$$

where $\lambda_{d,j}^{(b)}$, $j = lc, sc$ are design throughputs for the baseline system with large and small caches, respectively.

The next issue is how to deal with workloads other than SPECweb96. For this, we concentrate only on the impact of additional I/O in DP-server and WebProxy workloads. The most direct impact of additional I/O is an increase in "path length," defined as number of instructions per transaction. Let ξ_{ntwk} denote the path length of sending/receiving a single TCP/IP packet. Similarly, let ξ_{disk} denote the path length of a single disk I/O operation (with certain maximum sequential transfer size). Then, the new path length per transaction (PL) can be related to the base (i.e., SPECweb96) path length PL_0 as follows:

$$PL = PL_0 + (T_{\text{ntwk}} - T_{\text{ntwk}}^{(b)})\xi_{\text{ntwk}} + (T_{\text{disk}} - T_{\text{disk}}^{(b)})\xi_{\text{disk}}, \quad (20)$$

where the superscript (b) again indicates that the parameter refers to the baseline system. An increase in path-length would result in a corresponding increase in memory references involved in executing the extra instructions. Here, we assume a proportional increase in computational memory references as well. That is, $B_{\text{mem,dep},j}$ is further scaled by the factor PL/PL_0 . Note that this is optimistic since the I/O related path-length typically has higher misses per instruction (MPI) for the processor cache.

An additional consequence of increased path length is that, with the same CPU, the achievable throughput would go down. Thus, everything being the same, the DP-server/WebProxy workloads may achieve only a fraction of the throughput of SPECweb96. Thus, one way to exhibit differences between the three workloads is to actually scale the throughput by the path length and compute the bandwidth requirements. While this is perfectly legitimate, we do not do so in the sample results shown in Section 6 since we want to display the bandwidth requirements for a given throughput level. In this point of view, *it is assumed that we have a CPU as capable as need be, and the emphasis is on various bandwidths.*

Note that additional I/O would result in worse cache miss ratios because of additional snoops from the PCI side. However, computing the additional memory references via this route is very difficult and, therefore, we resorted to a more direct path-length-based approach. Also note that the additional I/O would also alter the read/write ratio for memory references; but, fortunately, we do not need to estimate read and write traffic separately.

5 SETTING RATED UTILIZATION

The equations developed in the last section provide the *used bandwidth* for various resources for a given target op-count

value. In order to compute the corresponding *required bandwidth*, we need to supply an appropriate maximum resource utilization level, henceforth termed as *rated utilization*. For example, if the rated utilization is 0.75, the used bandwidth will be only 75 percent of the required bandwidth. In particular, let B_j denote the required bandwidth for resource j on a per transaction basis and U_j its "rated utilization." Then,

$$B_j = B_j U_j / \lambda_d \text{ where } j = \text{ntwk, disk, pci, exp, mem, fsb.} \quad (21)$$

The desired rated utilization depends on the nature of the traffic. Generally, as the traffic becomes more bursty, more headroom is needed to better handle short-term traffic peaks and, hence, a lower rated utilization must be used. Traffic characteristics depend strongly on the level at which it is observed: Generally, resources closer to the CPU core will experience less bursty traffic than at outer levels. For example, the traffic on the front side bus will be less bursty than that on the PCI bus, which, in turn, will be less bursty than the traffic experienced by NICs. This difference is a result of *filtering* or *traffic shaping* that occurs because of smaller transaction sizes, non-FIFO scheduling, and various queue-depth limitations at lower levels. For example, memory access from the host or PCI side occurs in small units (single or at most a few cache-lines) and is subject to limitations on the number of pending PCI transactions and the amount of undelivered data. Also, all memory accesses concerned with a single high-level operation, e.g., sending out a given packet, do not occur as a single burst, but are mixed with accesses for other operations. This effectively "smooths" the traffic seen by the memory controller. An important point to note in this regard is that if all the data required by the high level operation was streamed without interruption, no such smoothing with take place and the outer level burstiness will sneak at the lower levels. Also, even if the traffic closer to the CPU core is smooth, the rated utilization of resources at that level should be kept relatively low in order to avoid long delays that would starve the CPU core. The rated utilizations suggested here are guided by these considerations.

Another important aspect in deciding rated utilizations is request and response size distributions. For SPECweb96, request sizes are almost constant, but response sizes have significant variability. High NIC utilization in this case causes problems if several clients want to concurrently receive large files.⁸ For WebProxy traffic, both request and response sizes vary over a large range. In particular, occasional transfers of files exceeding 10 MB would require transmitting $> 6,900$ packets and > 100 ms time, even using a GB Ethernet. Such a transfer may seriously impact many other requests that are queued up behind this one. A lower rated utilization would help by reducing the probability of running out of bandwidth and by allowing better mixing of various requests.

8. In isolated cases, a client may not be fast enough to absorb large files and, thus, result in TCP window flow control coming into effect.

The values assumed for rated utilization factors for both SPECweb96 and WebProxy cases were selected as follows (WebProxy utilization values are used for DP-server workload as well because of the assumption of similar traffic processes):

1. *Network utilization* U_{ntwk} : We use 0.8 for SPECweb96 and 0.5 for WebProxy. The latter value is chosen by using queuing results for a deterministic queue fed by a FBM (fractional Brownian Motion) arrival process [16], which indicates that $U_{\text{ntwk}} = 0.5$ for WebProxy should lead to about the same level of delays as $U_{\text{ntwk}} = 0.8$ in the Markovian arrival case.
2. *Disk adapter utilization* U_{disk} : We use 0.60 for SPECweb96 and 0.45 for WebProxy. For SPECweb96, we retain disk utilization small simply to allow fast initial loading of files. For other workloads, the arrival process to the disk adapter can be considered to be a thinned NIC arrival process. Therefore, this process should stay self-similar with the same Hurst parameter. In view of this and the need to keep the impact of reading very large files minimal, we choose $U_{\text{disk}} = 0.45$.
3. *I/O-memory path utilization* U_{iom} : We use 0.55 for SPECweb96 and 0.45 for WebProxy. A low utilization is necessary here to keep PCI-memory transfer delays low; however, because of transfers in small bursts, the high burstiness at the higher level should not significantly increase the burstiness at this level. Thus, we use only a marginally lower $U_{\text{iom}} = 0.45$ for WebProxy traffic.
4. *PCI bus utilization* U_{pci} : We use 0.60 for SPECweb96 and 0.50 for WebProxy. Assuming that there is no significant intermediate buffering, PCI bus and I/O-memory path traffic characteristics should be similar. A high PCI utilization may result in a lot of retries for inbound reads. PCI bus transfers have a significantly higher overhead than I/O-memory path transfers, but much of the overhead is due to small transfers and retries. Thus, it is reasonable to choose PCI bus utilization that results in about the same bandwidth requirement as the I/O-memory path (for 32 bit PCI).
5. *Memory utilization* U_{mem} : We use 0.60 for SPECweb96 and 0.50 for WebProxy. We model memory as a four-stage pipeline corresponding to various access stages of a DRAM [12], but only one of these needs to be modeled as a queuing station (one server per memory channel), and its utilization is of concern here. Burstiness is perhaps not that much of an issue here; however, memory channels may experience a significant amount of overhead (refresh, channel synchronization, etc.), which makes the sustainable memory bandwidth significantly lower than the peak bandwidth. Thus, the assumed 60 percent utilization for SPECweb96 may itself be quite aggressive.
6. *Processor address/data bus utilization* U_{fsb} : We use 0.6 for SPECweb96 and 0.50 for WebProxy. The issues here are very similar to those for memory channels. In particular, the occurrence of dead

clock cycles may result in much higher bus occupancy that the address/data related utilization. Thus, 60 percent data bus utilization may also be quite aggressive.

6 SAMPLE RESULTS

In this section, we list some sample results computed from the above equations for the three workloads considered here. To avoid confusion, it is important to note that the numerical values are applicable only to the Web server configurations similar to the baseline. In particular, *the baseline system discussed here did not use any web accelerator hardware or software and, thus, achieved SPECweb96 throughput much below the numbers achievable by comparable platforms.* A system including four Pentium II Xeon processors at 500 MHz each and Intel 82450NX chip set, running Microsoft Windows NT4.0 and IIS5.0 was used as a baseline. The achieved throughputs were

$$\lambda_{d,lc}^{(b)} = 4,275 \text{ trans/sec}$$

with 2 MB cache and

$$\lambda_{d,sc}^{(b)} = 3,900 \text{ trans/sec}$$

with 512 KB cache. At the time of this writing, numbers as high as 3X have been claimed using what amounts to a “benchmark special” configuration (although much of the performance boost results from the fact that SPECweb96 file-set is entirely static and predetermined). Even without going to such lengths, continuing improvements in O/S, TCP/IP stack, HTTP implementation, etc., would perhaps squeeze out much better performance than this. The numerical example here should be considered against this backdrop.

For the baseline system, the cache miss ratio dependent memory references were found to be

$$\begin{aligned} B_{\text{dep_read,lc}}^{(b)} &= 320, & B_{\text{dep_write,lc}}^{(b)} &= 1,565, \\ & & B_{\text{dep_read,sc}}^{(b)} &= 736, \text{ and } \\ & & B_{\text{dep_write,sc}}^{(b)} &= 1,656. \end{aligned}$$

All these are in units of 32 byte cachelines. Considering a time horizon of the next two or three years, a reasonable high-water mark for a 4P SMP system (or a 4P SMP node in a cluster) may be about 4 times the current performance. Consequently, we chose a target value of 17,000 trans/sec in the numerical example.

Other parameters used in the example are as follows: We assume that one-third of the files and one-fifth of the bytes received by the proxy server are noncacheable, i.e., $\alpha_{ncf} = 0.333$ and $\alpha_{ncb} = 0.20$. We also assume that the cacheable data gets a file-hit ratio of 60 percent and byte hit ratio of 50 percent, i.e., $q_{hf} = 0.60$ and $q_{hb} = 0.50$. This implies that the overall file and byte hit ratios of the proxy cache are both 40 percent. We assume that there is 10 percent probability that the data will be modified before it is written out to the disk (i.e., $\alpha_{mod} = 0.1$). We assume that the file-cache byte miss

TABLE 2
Overhead Factors for 32 Bit and 64 Bit PCI Buses

PCI width (in bits)	ntwk adapter overhead (α_{nic})	disk adapter overhead (α_{da})
32-bits	2.0	1.5
64-bits	3.0	2.0

ratio (i.e., q_{lc_mv}) is 15 percent. Finally, we assume the same keep-alive probability $(1 - \theta_{ka}) = 0.75$ for both client-proxy and proxy-server connections. For SPECweb96, we set $n_{rex} = 0$ and, for the other two workloads, we assume $n_{rex} = 0.1$, which is not unreasonable for transmissions over wide area networks during busy periods. We assume $q_{retry} = 0.0$ for SPECweb96 and 0.05 for other workloads. Note that 5 percent retry probability may be a severe underestimate for popular web servers during busy times, but capacity planning should not be driven by extreme situations. We assume a sequential disk read/write size of 8 KB, which is conservative. Finally, we assume an average PCI burst size of $\Delta_{nic} = 128$ bytes for NICs and $\Delta_{da} = 256$ bytes for disk adapters. By current standards, these are actually optimistic, but should be reasonable in the longer run. Table 2 lists the corresponding values of bandwidth inflation factors α_{nic} and α_{da} .

For estimating the path length impact, we assume $PL_0 = 100,000$, which is the typical path length for SPECweb96 transaction in a 4P SMP system. In SPECweb96, it is found that about 65 percent of processor time is spent in TCP/IP stack, of which about 75 percent utilization concerns packet sending and reception. Accordingly, we use $\xi_{ntwk} = 2,500$. Based on some measurements of disk I/O path length, we let $\xi_{disk} = 12,500$.

Table 3 lists the *used bandwidth* (i.e., assuming $U_j = 1$ for all j) with default TCP/IP settings and 32 bit PCI buses. The last subtable in this table is concerned with memory and data bus bandwidths for large/small caches, and 0-copy sends/0-copy I/O. In the first two subtables, the bandwidths are listed in both directions of transfer in order to highlight the degree of load imbalance. Note that, for SPECweb96 and DP-server, the receive bandwidth in bytes/sec is rather small but not so in terms of packets/sec. For disks, “read” and “write” columns refer to disk reads and disk writes (or memory writes and memory reads). However, for I/O-memory path and PCI bus, the “read” and “write” columns refer to memory reads and memory writes, respectively. For total I/O transfers, we simply add up the number of packets handled by NICs and I/Os by disk adapters in each direction.

Table 4 lists the *required bandwidth* at utilization levels discussed in Section 5. The assumptions and rationale for the number of units listed in Table 4 is as follows:

1. **Number of NICs:** We assume Ethernet based network using 1 Gb/sec full-duplex links. The column “no.” under “Network BW” in Table 4 lists the number of Gigabit Ethernet NICs. Note that the full-duplex operation means that the number of NICs is governed by the direction with higher bandwidth demands, which happens to be network

TABLE 3
Used Bandwidth for 17,000 Trans/Sec Throughput: Normal Case

Workload type	Network BW			Network pkts			Disk IO BW			Disk xfrs K/s
	send	rcv	tot.	send	rcv	tot.	read	write	tot.	
	(MB/sec)			(K-pkts/sec)			(MB/sec)			
SWeb-96	262	3	265	195	119	314	61	2	63	8
DPserver	304	7	311	227	138	365	64	2	67	8
WebProxy	308	190	498	310	274	583	64	118	182	22

Workload type	IO-mem path BW			PCI bus BW			IO Transfers		
	read	write	total	read	write	total	read	write	total
	(MB/sec)			(MB/sec)			(K-IOs/sec)		
SWeb-96	356	105	461	528	97	625	196	126	322
DPserver	412	119	531	611	111	722	227	146	373
WebProxy	572	423	995	794	476	1269	324	282	606

Workload type	Total memory BW (GB/s)				Total dbus BW (GB/s)			
	0-copy sends		0-copy I/O		0-copy sends		0-copy I/O	
	Large cache	Small cache	Large cache	Small cache	Large cache	Small cache	Large cache	Small cache
SWeb-96	1.46	1.89	1.45	1.88	1.19	1.63	1.18	1.62
DPserver	1.82	2.31	1.70	2.20	1.46	1.95	1.34	1.84
WebProxy	2.95	3.57	2.30	2.91	2.32	2.93	1.66	2.28

sends for all three workloads (see "Network BW" columns in Table 3).

- Number of disk adapters:** We assume the use of *Fiber Channel Arbitrated Loop*, FC-AL, adapters, each with peak rate of 4,250 I/Os per sec. (This rate is at 100 percent loop utilization; each adapter will actually be running at the rated utilization U_{disk} .) Assuming 2,000 I/Os per sec for SCSI adapters, each FC-AL adapter equals 2.13 SCSI adapters.
- Number of I/O-memory paths:** Table 4 lists the number of I/O-memory paths (each with transfer rate of 400 MB/sec) under the column "no." in "I/O-memory path BW." The two numbers separated by a slash are, respectively, the number of half-duplex and full-duplex I/O-memory paths. The half-duplex number considers the direction in which the traffic is higher, which happens to be memory reads for all three workloads.
- Number of PCI buses:** Table 4 lists the number of 32/66 (32-bit, 66 MHz) and 64/66 (64 bit, 66 MHz) PCI buses separated by a slash under the column "no." in "PCI bus BW." These are computed using the overhead data in Table 2. It is seen that, because of much higher overhead of 64 bit PCI bus, the required number of buses goes down only modestly.

The substantially higher bandwidth demands for a proxy server compared to a native server should be obvious from these tables. The much higher demands

result primarily from a high probability of having to obtain the web page from a remote server. In particular, an available memory and FSB peak bandwidth of 3.2 GB/sec are adequate to support the 17,000 trans/sec in case of SPECweb96 (given a reasonably large cache). In contrast, for the WebProxy case, even 6.4 GB/sec bandwidth is inadequate unless all I/O is 0-copy type or the system has a large cache.

It is interesting to compare the benefit of 0-copy I/O (as supported by VIA) vs. 0-copy sends only (as supported by current operating systems already). As can be seen from memory and the FSB bandwidth subtables in Table 3 and Table 4, for SPECweb96, there is almost no benefit (small receive packets and very little disk I/O). For DP-server, the benefit is small but significant. For Web-Proxy, however, 0-copy I/O results in about 20 percent decrease in memory bandwidth requirements.

For the network and disk I/O, the maximum number of I/Os per second is usually a much more significant metric than the raw bandwidth (in bytes/sec) since each I/O involves a DMA setup, interrupt handling, packet header manipulation, transfer overheads, etc. Table 5 shows the impact of making certain optimizations to reduce the I/O rate. In particular, the packet size is doubled to 3,000 bytes, packets per ack, increased from 2 to 5, sequential transfer size increased from 8 KB to 16 KB, and inbound read size over the I/O-memory path is increased from two cachelines to four. Note that the decrease in I/O rate also reduces the path length by about 10 percent and, thus, frees up that much CPU time for useful work. In addition,

TABLE 4
Bandwidth Demands for 17,000 Trans/Sec Throughput: Normal Case

Workload type	Network BW			Disk IO BW			IO-mem path BW			PCI bus BW		
	GB/s	no.	util	GB/s	no.	util	GB/s	no.	util	GB/s	no.	util
SWeb-96	0.33	3	0.80	0.11	4	0.60	0.84	3/2	0.55	1.04	4/3	0.60
DPserver	0.62	5	0.50	0.15	5	0.45	1.18	3/3	0.45	1.44	6/5	0.50
WebProxy	1.00	5	0.50	0.41	12	0.45	2.21	6/4	0.45	2.54	10/7	0.50

Workload type	rated mem util (%)	Total memory BW (GB/s)				total dbus util	Total dbus BW (GB/s)			
		0-copy sends		0-copy I/O			0-copy sends		0-copy I/O	
		Large cache	Small cache	Large cache	Small cache		Large cache	Small cache	Large cache	Small cache
SWeb-96	0.60	2.43	3.16	2.42	3.14	0.60	1.99	2.71	1.97	2.70
DPserver	0.50	3.63	4.63	3.40	4.39	0.50	2.91	3.90	2.68	3.67
WebProxy	0.50	5.90	7.13	4.59	5.82	0.50	4.63	5.86	3.32	4.55

TABLE 5
Used Bandwidth for 17,000 Trans/Sec Throughput: Optimized Case

Workload type	Network BW			Network pkts			Disk IOBW			Disk xfls K/s
	send	recv	tot.	send	recv	tot.	read	write	tot.	
SWeb-96	258	3	261	106	51	157	61	2	63	4
DPserver	299	7	307	123	59	182	64	2	67	4
WebProxy	304	187	491	158	133	291	64	118	182	11

Workload type	IO-mem path BW			PCI bus BW			IO Transfers		
	read	write	total	read	write	total	read	write	total
SWeb-96	309	105	414	520	97	617	106	55	161
DPserver	357	118	476	602	111	713	123	63	186
WebProxy	498	418	916	784	470	1255	166	137	302

Workload type	Total memory BW (GB/s)				Total dbus BW (GB/s)			
	0-copy sends		0-copy I/O		0-copy sends		0-copy I/O	
	Large cache	Small cache	Large cache	Small cache	Large cache	Small cache	Large cache	Small cache
SWeb-96	1.46	1.89	1.45	1.88	1.19	1.63	1.18	1.62
DPserver	1.77	2.25	1.65	2.13	1.41	1.89	1.30	1.78
WebProxy	2.73	3.27	2.09	2.62	2.11	2.64	1.46	2.00

the optimized system requires fewer disk adapters and disks (disk adapter and disk capacity is limited primarily by I/Os per sec rather than by the raw bandwidth). Finally, the reduced I/O rate correspondingly reduces the processor interrupt rate, which may result in additional offloading of CPU in the 1-2 percent range.

7 DISCUSSION AND CONCLUSIONS

It is clear from the discussion in the previous section that memory bandwidth requirements significantly exceed processor data bus bandwidth requirements. *This is a fundamental difference between web workload and OLTP workload such as TPC-C.* Web workloads are very I/O intensive (primarily network I/O) and, thus, involve a considerable amount of PCI-to-memory transfers (which

do not use processor data bus). To support such workloads well, the provisioned memory bandwidth needs to be about 20 percent higher than the provisioned data bus bandwidth.

Currently, SPECweb96 performance on high end systems is limited by the latencies and resource locking within the TCP/IP stack. In view of this, efforts are currently underway to move much of the TCP/IP processing from the main processor down to either the NIC or to an I/O processor (I2O style) [5]. Such a change could easily give a 25 percent performance boost *if only the I/O-memory subsystem can provide the needed bandwidth.* This points to the need for a larger headroom in I/O subsystem design than in the design of the processor-bus subsystem. The large performance boost achievable for SPECweb96 using web accelerators points to a similar conclusion.

TABLE 6
Notation for All Resources and Operators

A_x	Explicit ack overhead per trans for $x = recv$ and $x = send$
B_x	Required bandwidth for resource-operation combination given by x
B_x	Required bandwidth per transaction in bytes for resource x
B_{min_dk}	Minimum required disk I/O bandwidth
$B'_{disk,r}$	Total disk read bandwidth
$B_{fsb,j}$	Required FSB bandwidth ($j = lc$ for large cache, $j = sc$ for small cache)
$B_{fsb,ind}$	Cache miss-ratio independent FSB refs
$B_{ind,o}$	Cache miss ratio independent memory operations $o = r, w$
$B_{iom,j}$	Required I/O to memory bandwidth for $j = lc, sc$
$B_{mem,j}$	Required memory bandwidth for $j = lc, sc$
$B_{mem,dep}$	Cache miss-ratio dependent memory refs
$B_{mem,ind}$	Cache miss-ratio independent memory refs
D_{req}	Number of directories
L_{log}	Number of bytes written to the log per HTTP operation
L_{req}	Average size of a client request in bytes
L_{resp}	Average size of a server response in bytes
L_{xfr}	Maximum sequential transfer size for disk
$Mem(\lambda_d)$	Minimum memory required for at (throughput λ_d)
M_x	Management overhead per trans for $x = send$ and $x = recv$
n_{rex}	Average number of transmissions for correct communication
N_c	Number of client machines
N_{blk}	Number of IOs needed to transfer one file
N_{exp_ack}	Avg number of explicit acks per trans, per file send
N_{ctrl}	Number of PCI cycles per burst for control traffic
N_p	Number of client processes per machine
N_x	Number of packets per request for operation $x = send$ or $x = recv$
q_{c_mr}	Prob that cacheable data at proxy resides in the disk
q_{min_dk}	Minimum disk read probability per transaction
q_{hb}, q_{hf}	Fraction of <i>cacheable</i> bytes (files) delivered by the proxy
q_{retry}	User retry probability
q_{rrb}, q_{rrf}	Remote receive fraction in terms of bytes (files)
R_x	Average number of PCI bursts for $x = send$, and $x = recv$
S_a	Average overall access size in bytes
S_{ca}	Average access size of files cached by the proxy in bytes
T_x	Required bandwidth per trans in disk-blocks or packets for resource designator x
T_{sleep}	Sleep time of each client process in seconds
w_{recv}	Average number of packets received for one acknowledgement
α_{da}	PCI bandwidth inflation factor for disk adapter traffic
$\alpha_{iom,o}$	IO-memory bandwidth inflation factor for memory operation $o = r, w$
α_{mod}	Prob that a file is modified before being written to disk
$\alpha_{ncb}, \alpha_{ncf}$	Fraction of uncacheable bytes (files) at the proxy server
α_{nic}	PCI bandwidth inflation factor for NIC traffic
$\Delta_{da}, \Delta_{nic}$	Average PCI burst size for disk adapter (NIC) in bytes
λ_d, λ_a	Achieved and design throughputs respectively
θ_{ka}	Probability of not keeping the connection alive in HTTP 1.1
ξ_{disk}	Pathlength per trans for reading/writing one disk I/O block
ξ_{ntwk}	Pathlength per trans for sending/receiving one TCP/IP packet

It is clear from the analysis here that Web workloads are very I/O intensive in terms of both byte-count and I/O-count. This may put a severe squeeze on available connectivity since the current 66 MHz PCI can support

only a few adapters. This may be an impetus for higher connectivity I/O solutions such as those based on System Area Network (SAN) [13].

APPENDIX A

NOTATION

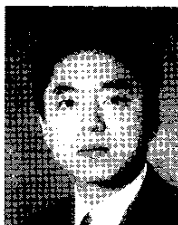
In Table 6, consistent notation is used for all resources and operations whenever possible. In particular, the subscript on many quantities, denoted by x , refers to the resource-operation combination (disk, r), (disk, w), disk (disk read, write, and total), fsb (front side bus), iom (I/O memory path), mem (memory), send, recv, ntwk (send, receive, and total), and pci (PCI bus).

REFERENCES

- [1] M. Arlitt, R. Friedrich, and T. Jin, "Performance Evaluation of Web Proxy Cache Replacement Policies," technical report, Hewlett Packard Laboratories, 1998.
- [2] G. Bhal, "SPECweb96—A Benchmark for Web Server Performance," Sun Microsystems, Oct. 1996.
- [3] R. Caceres et al., "Web Proxy Caching: The Devil Is in the Details," *Proc. Usenix Symp. Internetworking Technologies and Systems*, 1998.
- [4] K. Claffy, G. Miller, and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone," Cooperative Assoc. for Internet Data Analysis (CAIDA); URL: www.caida.org/Papers/Inet98.
- [5] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Comm.*, pp. 23-29, June 1989.
- [6] M.E. Crovella and A. Bestavros, "Explaining World-Wide Web Self-Similarity," technical report, Dept. of Computer Science, Boston Univ., Oct. 1995.
- [7] B.M. Duska, D. Marwood, and M.J. Feeley, "The Measured Access Characteristics of World-Wide Web Client Proxy Caches," *Proc. Usenix Symp Internetworking Technologies and Systems*, 1997.
- [8] F. Douglass, A. Feldmann, and B. Krishnamurthy, "Rate of Change and Other Metrics: A Live Study of World Wide Web," *Proc. Usenix Symp. Internetworking Technologies and Systems*, 1997.
- [9] D. Dunning et al., "The Virtual Interface Architecture: A Protected, Zero Copy User-Level Interface to Networks," *IEEE Micro*, pp. 66-76, Mar./Apr. 1998.
- [10] R. Felding et al., "Hypertext Transfer Protocol—HTTP 1. 1," IETF RFC 2068, Jan. 1997.
- [11] A. Feldmann et al., "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments," *Proc. IEEE INFOCOM '99*, Mar. 1999.
- [12] J. Hennesy and D. Patterson, *Computer Architecture: A Quantitative Approach*, second ed. Morgan Kaufmann, 1996.
- [13] R.W. Horst and D. Garcia, "ServerNet SAN I/O Architecture," *Proc. Hot Interconnects*, 1997.
- [14] K. Kant and Y. Won, "Server Capacity Planning for Web Traffic Workload," technical report, Server Architecture Laboratory, Intel Corp., Sept. 1998.
- [15] J. Mogul et al., "Potential Benefits of Delta Encoding and Data Compression for HTTP," *Proc. SIGCOMM '97*, pp. 181-194, Sept. 1997.
- [16] I. Norros, "A Buffer with Self-Similar Input," *Queueing Systems*, vol. 16, no. 2, pp. 382-396, Feb. 1994.
- [17] K. Park, G. Kim, and M. Crovella, "On the Effect of Traffic Self-Similarity on Network Performance," *Proc. SPIE Int'l Conf. Performance and Control of Network Systems*, Nov. 1997.
- [18] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226-244, June 1995.
- [19] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 1-15, Feb. 1994.
- [20] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson, "Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *Proc. SIGCOMM '95*, pp. 100-113, 1995.



Krishna Kant received his PhD degree in computer science from the University of Texas at Dallas in 1981. He was an assistant professor of computer science at Northwestern University, Evanston, Illinois, from 1981 to 1984; and an assistant professor from 1985 to 1989 at Pennsylvania State University, University Park, and an associate professor of computer science at Penn State from 1989 to 1991. In 1988, he also served with the Teletraffic Theory Division of AT&T Bell Laboratories, Holmdel, New Jersey; and, in 1991, he was with the Integrated Systems Architecture Division at BellCore, Piscataway, New Jersey. He was with the Network Services Performance and Control Group at Bellcore, Red Bank, New Jersey, from 1992 to 1997, working on a variety of narrowband and broadband signalling performance and congestion control issues. Since May 1997, he has been with the Server Architecture Laboratory at the Intel Corporation, Beaverton, Oregon, where he works on the impact of Web server architecture and input traffic characteristics on application-level performance. He is the author a book entitled *Introduction to Computer Systems Performance Evaluation* (McGraw-Hill, 1992). He is a member of the IEEE.



Youjip Won received his BS and MS degrees in computer science from the Department of Computer Science and Statistics at Seoul National University in Korea in 1990 and 1992, respectively; and the PhD degree in computer science from the University of Minnesota, Minneapolis, in 1997. After receiving his doctorate, he worked for the Server Architecture Laboratory of the Intel Corporation, Beaverton, Oregon, as a server performance analyst. Currently, he is a member of the faculty in the Division of Electrical and Computer Engineering, Hanyang University, Seoul, Korea. His current research interests include multimedia systems, Internet and web applications, and performance modeling and analysis. He is a member of the IEEE and the IEEE Computer Society.