Web Content Adaptation to Improve Server Overload Behavior

Tarek F. Abdelzaher
Real-Time Computing Laboratory
EECS Department, University of Michigan
Ann Arbor, Michigan 48109–2122
zaher@eecs.umich.edu

Nina Bhatti
Hewlett Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
nina@hpl.hp.com

Abstract

This paper presents a study of web content adaptation to improve server overload performance, as well as an implementation of a web content adaptation software prototype. When the request rate on a web server increases beyond server capacity, the server becomes overloaded and unresponsive. The TCP listen queue of the server's socket overflows exhibiting a drop-tail behavior. As a result, clients experience service outages. Since clients typically issue multiple requests over the duration of a session with the server, and since requests are dropped indiscriminately, all clients connecting to the server at overload are likely to experience connection failures, even though there may be enough capacity on the server to deliver all responses properly for a subset of clients. In this paper, we propose to resolve the overload problem by adapting delivered content to load conditions to alleviate overload. The premise is that successful delivery of a less resource intensive content under overload is more desirable to clients than connection rejection or failures.

The paper suggests the feasibility of content adaptation from three different viewpoints; (i) potential for automating content adaptation with minimal involvement of the content provider, (ii) ability to achieve sufficient savings in resource requirements by adapting present-day web content while preserving adequate information, and (iii) feasibility to apply content adaptation technology on the web with no modification to existing web servers, browsers or the HTTP protocol.

Keywords: Web server performance, adaptive content, overload protection

1 Introduction

Web servers today offer poor performance under overload. As the request rate increases beyond server capacity, server response-time and connection error rate deteriorate dramatically, potentially causing client-perceived service outage. Server overload may occur either due to saturation of CPU bandwidth or due to saturation of the communication link capacity connecting the server to the network. This paper introduces content adaptation as a technique to alleviate both types of overload. We note that our technique is designed for alleviating peak load conditions, rather than cope with permanent sustained overload. The latter case simply calls for upgrading the server platform.

Several techniques have been proposed to alleviate server overload such as methods for distributing the load across a number of geographically separated servers are presented in [4, 5, 7]. Redirection servers were proposed in [13] to transparently redistribute users' requests. In [2] a model is presented for dynamically scheduling HTTP requests across clusters of servers to optimize the use of resources. Rent-a-server; a technique for server replication on demand is presented in [15] to replicate servers on overload.

While the above techniques essentially propose solutions based on load balancing among multiple servers, we concern ourselves with the problem of overload management of an individual server. Generally, to cope with overload, servers either are over-provisioned or use admission control. When over-provisioning is used, administrators often allocate to a web server twice the normal capacity as a rule of thumb [14]. The approach does not always prevent overload conditions. Techniques that rely on client admission control are explored in [10]. Admission control improves the average latency of admitted client requests by rejecting a subset of clients. The premise is that consistent rejection of all requests from a subset of clients may be better than indiscriminate connection failures affecting all clients alike in the absence of admission control. Operating system research has also addressed the problem of resource allocation on server platforms, rethinking fundamental OS abstractions [3] and introducing new mechanisms [6, 11]. In this paper, however, we explore server resource management schemes implementable on top of mainstream operating systems such as UNIX.

As an alternative to connection rejection or failures, clients may be willing to receive a degraded, less resource intensive version of requested content. This makes a case for replacing (or augmenting) admission control schemes with mechanisms to adapt delivered web content to load conditions. Under heavier load, less resource intensive content can be served. In addition to alleviating overload, content adaptation will reduce the amount of server resources wasted on eventually unsuccessful or rejected connections. In our measurements, for example, when server load (request rate) is about 3 times the maximum server capacity we observed that about 50% of the end-system utilization is wasted on rejected connectionions. In other words, the end-system spends half of its time processing eventually rejected requests (e.g., protocol stack processing, queuing, socket call processing, etc). This problem is akin to the receive livelock, which has been addressed at the operating system level [12]. In the absence of an operating system solution, one might as well deliver an inexpensive version of content to the requesting clients instead of rejecting them after consuming a large amount of resources.

The paper proposes an architecture for content adaptation on the web as a means to control overload. We discuss content adaptation in terms of the suitability of today's web content to degradation, the required content provider involvement, the software mechanisms needed for adaptive web servers, and the resulting expected impact on performance. In addition to alleviating server overload, content adaptation technology has other important benefits. For example, it may be used to adapt server output to client-side resource limitations, or to provide better content to more important clients. The processing power, connection bandwidth and display resolution may vary significantly from one client to another. Content adaptation can provide the most appropriate version of content to each client in accordance with their resource constraints. Currently, content providers are faced with fine-tuning a compromise version of content that hopefully will not encumber slower clients, yet remain satisfactory to higher-end clients. The existence of adaptation technology, such as that described in this paper, will allow content providers to deliver higher quality content whenever possible thus enhancing their clients' browsing experience. Service adaptation to client-side limitations and network variability has been proposed, for example, in [9]. While acknowledging this benefit of adaptation technology, in this paper we focus on adaptation as a means to control server overload.

The rest of this paper is organized as follows. Section 2 examines different ways of adapting content on the web. Section 3 evaluates the resulting performance impact drawn from an analysis of representative web sites. Section 4 describes and evaluates software architectures for content adaptation from the web server's perspective. We focus on software solutions that require no modifications to the servers, browsers, or the HTTP protocol, and we demonstrate, using test results from an implemented prototype, the potential of our approach in solving the overload problem. The paper concludes in Section 5 with a summary of contributions and suggestions for future work.

2 Techniques for Content Adaptation

Web content must be adapted in a way that preserves essential information yet reduces the resource requirements of content delivery. We have verified in a companion paper [1] that with a 100Mb network connection, the overload bottleneck resource for typical web server workload is usually the end-system rather than the network. When the end-system is overloaded, introducing an extra stage of computation, such as data filtering and compression, will only further increase the load on the server. Therefore, in order to eliminate extra overhead at overload, content must be pre-processed *a priori* and stored in multiple copies that differ in quality and processing requirements. At overload, the server should be allowed to switch to a pre-existing lighter version of content.

Intervention of the content provider may be required to authorize or fine tune certain types of adaptation during the off-line pre-processing stage. For adaptation technology to be cost effective, this intervention has to be minimal and should not change the ways content providers have traditionally created the art. The cost of adapting content should hopefully be less, for example, than the cost of alleviating overload by upgrading the server's machine and/or its network connection. One may imagine appropriate authoring tools that allow web content developers to annotate parts of the content with specific *adaptation tags*, (e.g., expendable, degradable, or important). These tags are preprocessed by content management tools to create separate standard-HTML versions of the site. The appropriate version will be served at run-time depending on load conditions. Since the created content versions contain only standard HTML and image formats, no modification is required to the browsers. Default adaptation actions may be used by the preprocessing tools on those parts of the content that have not been tagged. Defaults will reduce the need for explicit adaptation tags thus substantially reducing the effort of utilizing adaptation technology by content providers. While a complete description of such tools is beyond the scope of this paper, in the rest of this section we present some techniques that might be used by such tools. Viability of these techniques is considered from two standpoints; their potential for partial automation, and their expected resource savings.

- *Image degradation by lossy compression:* In a survey we conducted in July 1998 of 80 shopping web sites we found that GIF and JPG images alone constitute, on average, more than 65% of the total bytes of a site. In many cases, these images can be significantly compressed without an appreciable decrease in quality. To help appreciate the potential bandwidth savings, Figure 1 compares a 74KB GIF to a 8.4KB JPG image of the same object. Although the difference in byte size is roughly an order of magnitude, the difference in quality is insignificant on most clients' displays. This demonstrates a potential to conserve resources by degrading image quality (e.g., via automated JPG lossy compression).
- *Reduction of embedded objects per page:* From the server-side perspective, document size is not as important as the number of embedded objects per page. Upon retrieving a URL the client application sends independent requests to fetch its embedded objects. Each request to the server consumes a relatively large fixed overhead in addition to a variable document-size-dependent overhead. On an Apache server [8] running on HP-UX we found the end-system processing time per request to be 1.6 ms independently of the retrieved URL size,

plus an additional 65 μs per each KB of delivered data.[1] Thus, retrieving a 1 KB file, for example, will consume more than 50% of the end-system's processing resources needed to retrieve a 25 KB file. Therefore, at overload, end-system processing savings arising from eliminating small cosmetic items (such as little icons, bullets, bars, separators, and backgrounds) can be significant. Many web developers today use an "alt" clause to define alternate text for such items. Adaptation tools may make use of this clause, when available, to replace cosmetic items in less resource intensive versions of content. Content providers may tag objects that should not be removed by default treatment.

- *Reduction in local links:* Another way of adapting content is to reduce local links. This reduction will affect user browsing behavior in a way that tends to decrease the load on the server as users access less content. Reduction of local links may be automated, e.g., by limiting the web site's content tree to a specific depth from the top page. Content providers may indicate, using special tags, subtrees that should be preserved beyond the default depth during the reduction process.

Figure 1: Comparing a 74KB GIF and an 8.4KB JPG

To support multiple versions of content the path to a particular URL in a given content tree is the concatenation of the content tree name and the URL name, prefixed by the name of the root service directory of the web server. For example, in the root service directory "/root" one may create two content trees, "/full_content" and "/degraded_content". A URL "/my_picture.jpg" will be served from the directory "/root/full_content/my_picture.jpg" if server load permits. Otherwise the URL will be served from "/root/degraded_content/my_picture.jpg" thus supplying a more economic version.

The scheme applies to dynamic content as well, e.g., that generated by CGI scripts. Multiple content trees may contain different versions of the named CGI script (e.g., "/cgi-bin/my_script.cgi"). The script URL is prepended by the right tree name (e.g., "/full_content" or "/degraded_content") to determine which version of the script to execute under given load conditions. For example, the web server can invoke a less resource-intensive search script that looks for only the first 5 matches under overload, instead of one that looks for 25 matches under normal load conditions.

Our experience indicates that serving dynamic content is much more resource consuming than serving static content. Our adaptation mechanism allows some dynamic content to be replaced by static content by switching to a different content tree. For example, an on-line vendor can use a dynamically generated version of their product catalog that interacts with a stock database to display the items currently in stock. Content adaptation mechanisms allow the server to switch automatically, without system administrator assistance, to a statically

[1]These are HTTP 1.0 measurements taken for Apache 1.3.0 on a single-processor K460 (PA-8200 CPU) running HP-UX 10.20, with 512 MB main memory, and GSC 100-BaseT network connection.

pre-stored catalog version whenever bandwidth becomes scarce thereby saving resources. The burden of creating the static version lies on the content provider and authoring tools. For example, a given parent URL may have a link to a dynamic CGI script in one version and to static content in another.

In the following section we argue for the efficacy of content adaptation on the web by surveying selected web sites and verifying the expected performance benefits of adapting their content.

3 Adaptation Impact Study

We investigate the expected impact of the aforementioned content adaptation techniques in terms of achieved performance gains. Before the gains are described, note that, from a performance standpoint, adaptation by switching to a different content version may result in decreased cache performance due to initial cache misses. However, since content switching will occur much less often than the time it takes to cache the popular pages, the performance benefits of adaptation are achieved. In general, as we shall describe in Section 4.2, the server may serve different versions of the same URL to different clients concurrently, in which case one may require a slighly larger file cache on the server to hold both the adapted and non-adapted versions of popular pages at the same time. If the file cache is too small, adaptation may involve some performance penalty since it tends to require caching more files. In the experiments conducted in this paper, however, the server's file cache could hold the entire working set of popular pages. Thus, no cache performance penalty was observed. The effect of server file cache size will therefore not be discussed any further.

This section presents results of a study of the nature of content of 80 selected web-sites, and the expected performance impact derived by applying the adaptation techniques described in Section 2 to the content of surveyed sites and estimating the resulting savings in server resource consumption. Due to the increasing interest in e-commerce on the web, in this study we focus on shopping web sites. Today, a significantly increasing number of businesses advertise their products on-line. Shopping sites, in addition to their growing popularity, have several other favorable attributes that make them a good candidate for our analysis. These attributes are outlined below.

- As more users find it easier to shop online, a shopping site may get a large number of hits, and thus (unlike personal web pages, for example) is more susceptible to overload. Shopping sites are therefore potential customers of adaptation techniques designed to alleviate overload conditions.
- Businesses that advertise products online are often not web experts. They outsource site management to professional web-hosting service providers who might be interested, for economical considerations, in cohosting several sites per machine which further increases overload potential and the need for overload management mechanisms.
- Shopping sites are visually intensive. The site should attract e-shoppers attention, and depict products in a way that encourages a purchase. These sites can greatly benefit from adaptation technology that allows providing a richer content whenever possible, yet avoids overload by switching to a less resource intensive version when needed.

In our study, we downloaded site content, degraded its quality, and estimated the expected performance improvement. Performance improvement was estimated in terms of the reduction, upon content degradation, in consumed server utilization arising from client access. In turn, this reduction was inferred from the average decrease in delivered bandwidth and request rate that clients are likely to impose on the server after content has been degraded. Performance improvement depends on the particular way content was degraded, which may be arbitrary and subjective. We therefore found it informative to first determine an upper bound on such performance improvement achievable by any degradation technique.

Intuitively, the best performance improvement is achieved if all content is reduced to mere text. Being too severe a degradation, it serves as an upper bound on the actual performance improvement one may reasonably expect from using a more realistic degradation approach.

Performance improvement due to adaptation further depends on the size of a client's cache and the length of the client's session.

- In the absence of client caching or for very short sessions, server access rate and bandwidth delivered to the client will depend on the total number of embedded objects per page, as each HTML page access will entail downloading all its embedded objects.
- If the client has a sufficiently large cache each object will be downloaded only once into the client's cache, regardless of how many accessed URLs it is embedded in.

In the surveyed sites the difference between these two cases was quite noticeable. We therefore computed two estimates, an optimistic one that corresponds to no caching or short sessions, and a pessimistic one that corresponds to an infinite cache size and long sessions. Note that in the optimistic estimate performance improvement stems from both reducing the number of embedded URLs per page and reducing average object size. In the pessimistic estimate, on the other hand, performance improvement stems mostly from reducing object size since the number of embedded objects per page does not matter in the presence of adequate caching as mentioned in the bullets above. As we shall see, performance improvement resulting from reducing object size alone is much less than that resulting from reducing embedded images.

3.1 The Optimistic Estimate

In this case we assume that the client accesses at random one page of the site. This is an approximation since some pages are more popular than others. Ignoring document popularity in our analysis can be partially justified by noting that more popular pages will tend to be cached and served by proxies. This makes page accesses on the web server more uniformly distributed. A valid criticism of ignoring document popularity is that, presently, web site designers might manually optimize the most popular pages to avoid overload making them less rich than average. Such practice may reduce the benefits of adaptation but is not accounted for in our analysis. The authors note, however, that such optimization may be an artifact of not having adaptation technology to begin with. Site designers are forced to pre-degrade the most popular pages in anticipation of possible overload, which is rather unfortunate. Adaptation technology will allow popular pages to be no less visually attractive than the rest of the site, since content can be degraded "on-demand" when necessary. Taking this consideration into account, the use of average page statistics with no regard to document popularity might, in a sense, underestimate adaptation gains of future web sites.

Let the number of embedded objects per HTML page be Emb in the site's original content tree. Retrieving exactly one web page will therefore impose, on the average, $1 + Emb$ accesses on the server (1 access to retrieve the HTML file and Emb accesses to retrieve its embedded objects). If content is degraded to pure text, retrieval of the same page will impose only 1 access.

To compute the savings in bandwidth resulting from content degradation, let the average HTML file size be H bytes, and let the average embedded object size be I bytes. The bandwidth delivered per page is therefore $H + IEmb$ if content is not degraded, and H if content is degraded to text.

We have experimentally verified, as discussed later in the paper, that server utilization, U, consumed at a given request rate R and delivered bandwidth BW is well approximated by the linear function $U = aR + bBW$, where

a and b are measurable constants that depend on the server software and platform. The ratio of server utilization values consumed by retrieving an average web page before and after degradation is thus:

$$Ratio_{optimistic} = (a(1 + Emb) + b(H + IEmb))/(a + bH)$$

This ratio reflects the performance improvement achieved due to degradation.

3.2 The Pessimistic Estimate

To compute the pessimistic estimate of performance improvement we assume the client has an infinite cache. The longer the client's session is, the more pages are cached and the less is the load imposed by the server. An infinite session will eventually cache all pages of the site, downloading each exactly once. Let h be the number of HTML files on the site, d be the number of unique embedded objects. The performance improvement achieved by content adaptation is the ratio of utilization values consumed to download the site before and after degradation. The ratio is given by:

$$Ratio_{pessimistic} = (a(h + d) + b(Hh + Id))/(ah + bhH)$$

Using simple algebraic manipulation, the above equation can be rewritten as:

$$Ratio_{pessimistic} = (a(1 + Emb_{min}) + b(H + IEmb_{min}))/(a + bH)$$

where $Emb_{min} = d/h$ is the ratio of the number of embedded objects to HTML files on the site. Note that the number of embedded objects per page, $Emb = Emb_{min}$ if every embedded object is referenced in exactly one HTML file. Since, some embedded objects (e.g., common icons, or backgrounds) may be referenced in more than one HTML file, in general $Emb \geq Emb_{min}$. Thus, $Ratio_{optimistic} \geq Ratio_{pessimistic}$. For an arbitrary user session length and cache size, we expect the performance improvement to lie generally between $Ratio_{optimistic}$ and $Ratio_{pessimistic}$ The values d, h, Emb, H, I above were computed for each surveyed site. The constants a and b were measured for our server (Apache 1.3 running on an HP-UX platform). The resulting estimates are shown in Figure 2. The figure depicts, for every achievable performance improvement factor, x, the percentage of sites, $P(x)$, whose performance will increase by *at least* x% upon adapting their content to pure text. Both the optimistic and pessimistic percentage estimates of are shown. Thus, for example, consider the point $x = 400$. The estimates indicate that 30% (pessimistic curve) to 90% (optimistic curve) of all sites will improve performance by at least 400% (value of x). Another way of interpreting the graph is to consider a particular percentage of sites $P(x)$, and observe the corresponding performance improvement range. For example, consider the point $P(x) = 60$. The estimates indicate that *at least* 60% of all sites will see a performance improvement of 200% (pessimistic) to 700% (optimistic).

3.3 A Content Degradation Heuristic

In order to access performance improvement for less severe degradation methods (as opposed to degrading to pure text) we repeated the derivation of optimistic and pessimistic performance improvement bounds when content is degraded using the following default policies:

- Remove all GIF images smaller than 1KB. Such images almost always constitute disposable cosmetic icons and clipart items.

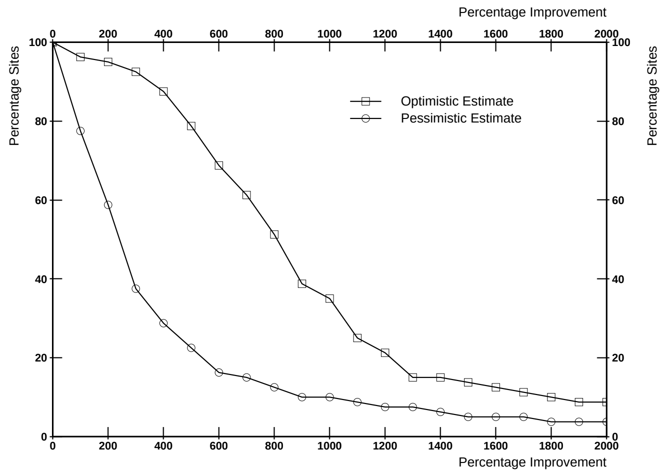

Figure 2: Performance Improvement Expected from Degrading to Text

- Degrade all images that are longer than 32KB by a factor of 8. As demonstrated in Figure 1 large images can generally be compressed by an order of magnitude with no significant effect on quality.
- Eliminate redundant items. Let each embedded object be referred to from exactly one HTML page (e.g., the most popular page in server logs among those where the object appears)

For lack of space we skip the derivation of performance improvement expressions used. Figure 3 compares the optimistic and pessimistic performance improvement bounds computed in this case. The performance improvement from the server's perspective comes mostly from reducing the number of hits on the server, e.g., reducing the embedded objects per page. Bandwidth (i.e., object size) reduction alone does not play a major role as seen from the pessimistic estimate in Figure 3 which relies mostly on bandwidth reduction. This is partly because most pictures on the web have already been degraded to a reasonably compressed size. Thus, there are very few images above 32KB, and the performance gain from degrading them is minimal. This situation is in part an artifact of the lack of adaptation technology. With adaptation technology in place, it will be possible to put higher quality images on more popular web pages.

The above feasibility study gives insights into how content should be degraded for maximum performance improvement, as well as estimates how much performance can be improved. It is shown that content adaptation can indeed lead to significant resource savings for a large category of sites if appropriate adaptation techniques are used.

4 Architecture for Content Adaptation

In this section we investigate content adaptation support from the web server's perspective. Having pre-processed the content *a priori* to create multiple versions we need on-line support for load monitoring and load-dependent content retrieval. We focus on approaches that do not require rewriting existing web server code. We describe the design of an add-on software component, called the *adaptation agent*, that executes as a separate process, independently from the web server, and provides the required online adaptation support. The main goal of this component is to determine which content tree needs to be used at a given time. To achieve its goal the adaptation agent performs the following functions:

- *Load Monitoring:* Web server load must be monitored in order to detect overload conditions.

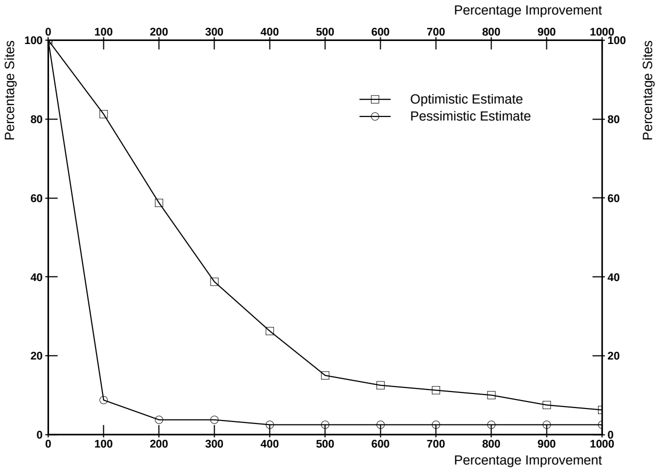

Figure 3: Performance Improvement Expected from Mild Degradation

- *Adaptation Triggering:* The adaptation trigger maps the monitored load value into a decision to invoke, undo, or change the extent of content degradation as appropriate.
- *Content Adaptation:* Once the trigger fires, indicating a state of server overload or underutilization, action is required to restore server load to desired conditions, if possible.

We describe different implementations of the above functions within the adaptation agent, their advantages and disadvantages, as well as their impact on achievable server functionality.

4.1 The Minimal Adaptation Agent

In the simplest case, the adaptation agent toggles, depending on load conditions, between two content trees; one for high quality content and one for degraded quality content. Its architecture is depicted in Figure 4. It implements an instance of a load monitor, adaptation trigger and content adaptor as follows:

- *Load Monitor:* A simple way of deciding whether or not the server is overloaded is to monitor server's response time. The adaptation agent periodically sends http requests and measures the corresponding response time. Measured response time is proportional to the length of the server's input request queue (e.g., the server's socket listen queue in a UNIX implementation). When the server is underloaded the queue tends to be short (or empty) resulting in small response times. At overload the request queue overflows making the response time grow an order of magnitude. This approximately bimodal behavior of the queue has been verified by our tests and can serve as a clear overload indicator. The advantage of estimating queue length by monitoring response time (rather than, say, counting queued requests) is that such a mechanism can be implemented outside the server requiring no modification to its code.
- *Adaptation Trigger:* Adaptation is triggered when measured server response time increases beyond a precomputed threshold, $Thresh$. This threshold can be set equal to (or slightly smaller than) the maximum server response time specified in a QoS agreement, if any, thereby causing adaptation when the agreement is about to be violated. In the absence of such a specification, the threshold can be derived from the preconfigured maximum input request queue length, Q (typically stored in a web server configuration file), and the average request service time, S. For example, if we consider a 90% full queue to be an overload indication, the trigger can be set to $Thresh = 0.9QS$. The parameter S can be found by benchmarking the server.
- *Content Adaptor:* Once the adaptation trigger is fired, the adaptation agent switches transparently from the high quality service tree to the degraded quality service tree in the root service directory. Content trees are

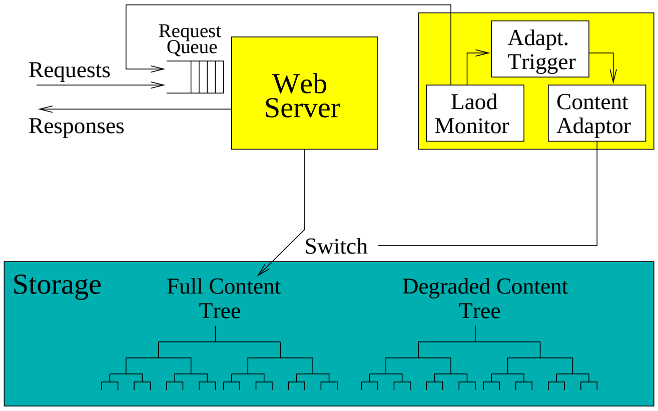

Figure 4: The Minimal Adaptive Server

switched by changing directory links. For example, let the root service directory be "/root", let the high quality tree name be "/full_content", and let the degraded quality tree name be "/degraded_content". The content adaptor creates a link from "/root" to "/root/full_content" to be used during normal operation. At overload, it changes that link to point to "/root/degraded_content". Thus, when the unmodified server accesses the same file name, such as "/root/my_url.html", it retrieves either the file "/root/degraded_content/my_url.html" or the file "/root/full_content/my_url.html" depending on load conditions.

Figure 5 compares the performance of an adaptive and non-adaptive servers by graphing the connection error probability versus request rate. In this experiment we generated requests for 64K images at an increasing rate. An adapted 8K version of the images was available in the degraded content tree. As shown in the figure, the traditional server suffers an increasing error rate when offered load exceeds capacity at about 160 requests/s. In contrast, our adaptive server switches to less resource-intensive content thus exhibiting almost no errors up to about 3 times the above rate. In general, the extent of performance improvement will depend on workload and the type of adaptation techniques employed during content pre-processing.

4.2 An Enhanced Adaptive Server

While increasing maximum throughput, as shown in Figure 5, the above described adaptation solution has some limitations. For example, it is not obvious when to switch back from degraded content to high quality content, and the server may become underutilized. Our measurements indicate that server response time is not particularly indicative of the degree of underutilization. The input request queue tends to be identically small as long as the server is operating below capacity. It is therefore difficult to tell whether or not reverting to high quality content at a particular time will result in overload. This limitation can be circumvented by measuring the load on the server instead of server response time. The measurement should give an idea of how underutilized the server or its communication link is. When server/link utilization decreases below a configurable value, the server may revert to non-degraded content.

CPU utilization alone is an insufficient load metric, since the bottleneck resource may be the communication bandwidth and not the CPU. Depending on the ratio of the platform's CPU bandwidth to the communication bandwidth a server may become overloaded due to communication bandwidth saturation at low CPU usage. Furthermore, the bottleneck resource can fluctuate between the CPU and the network depending on the load mix. We observed on our test platform that a large number of requests for small objects tends to saturate the CPU while a smaller number of requests for larger objects tends to saturate the network.

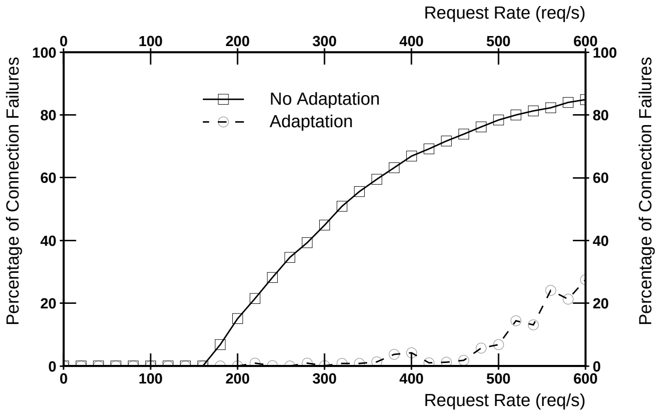

Figure 5: Adaptation and Connection Failure Probability

We therefore developed two different mechanisms for measuring the load on the bottleneck resource. The methods can be implemented at any software layer where server requests and responses are visible. For example, they can be implemented in a process (possibly residing on a different machine) that listens to the web server's subnet in promiscuous mode and observes all traffic to and from the web server. Alternatively, it can be implemented in a middleware layer such as a socket library that transparently performs server request rate and outgoing bandwidth statistics in the *read()* and *write()* socket library calls. These server utilization measurement mechanisms are described in the following section.

4.2.1 Bottleneck Resource Utilization Measurement

Measured bottleneck resource utilization should reflect load by quantifying the consumed percentage of bottleneck resource capacity. We developed two methods for measuring utilization that have the aforementioned property.

- *The Linear Approximation Method:* We have established that bottleneck resource utilization, U, consumed by processing client requests can be generally approximated by a linear function of measured request rate R, and delivered bandwidth, BW, such that:

  $$U = aR + bBW$$

  where constants a and b can be computed by either on-line or off-line profiling and regression analysis. The function is good for estimating offered load as long as it does not exceed server capacity. When capacity is exceeded either due to communication bandwidth saturation or CPU overload, the server will spend most of its time either blocked on communication I/O or busy with CPU execution. As a results the input request queue (the server's TCP socket listen queue) overflows and some requests are dropped. The dropped request rate consumes resources but cannot be measured outside the OS posing difficuly in measuring R accurately in the above equation under overload. We therefore combine the aforementioned linear approximation with response time monitoring to determine the load on the server. The idea is to separate out the overload condition, and use the linear approximation only when the server is underloaded. The combined utilization measurement function is as follows:
  - If measured response time is above threshold (overload) then let $U = 100\%$
  - If measured response time is below threshold (no overload) then let $U = aR + bBW$.

  where U is the utilization of the bottleneck resource, which may be the communication link bandwidth. One advantage of this method is that it is easy to implement in middleware. We replace the normal read() and write() socket library calls with our augmented versions. These library routines run in the context of each server process or thread, T_j, and record the observed request rate R_j, delivered bandwidth BW_j, and computed utilization U_j, then write it periodically to the adaptation agent process, e.g., via shared memory. The adaptation agent process reads and sums up the recorded values to obtain the aggregate request rate

R, bandwidth BW, and utilization U of the server. Access synchronization to shared data structures is not required since there is only one writer to any piece of recorded data. The scheme can be implemented transparently to the server with the help of an appropriately modified socket library.

The method provides means for converting a desired request rate and bandwidth into a corresponding resource capacity allocation that can be used by the adaptation agent to decide which tree to serve content from. On the disadvantage side, it requires computing the constants a and b via profiling.

- *The Gap Estimation Method:* A simpler method for estimating bottleneck resource utilization is to increment, in middleware, a global counter upon every request arrival and decrement it upon every response departure. Essentially, the counter keeps track of the number of requests being served concurrently at any given time. The counter will return to its initial (zero) value only when a gap is present, i.e., when all current requests have been served, and no additional requests have arrived yet. The gap ends with the arrival of the next request. The algorithm timestamps the beginning and end of each gap. The counter and gap timestamps are maintained in shared memory accessible to the adaptation agent. Gap beginning/end timestamps are used by the adaptation agent to compute the total idle time, G, in an observation period, T, where idle time is the time where no requests were pending. Bottleneck resource utilization is estimated as $U = (T - G)/T$. It is very important to note that this utilization includes the time the server is blocked on I/O. For example, if the communication link is the bottleneck, the server will spend most of its time blocked, waiting for low bandwidth TCP connections to complete the sending of response data. Since this waiting time occurs between request arrival and completion of response, it will be counted towards "busy time" in our utilization measurements. Thus the utilization can reflect overload of the bottleneck resource even when it's not the CPU. The method is illustrated in Figure 6, which shows concurrent processing of request bursts separated with idle time. The method does not require *a priori* profiling, and does not require monitoring response time. However, it has the disadvantage of using a global counter that may be updated by multiple writers and thus requires some form of locking or access synchronization between server processes and the adaptation agent process. Such synchronization may offer a performance penalty.

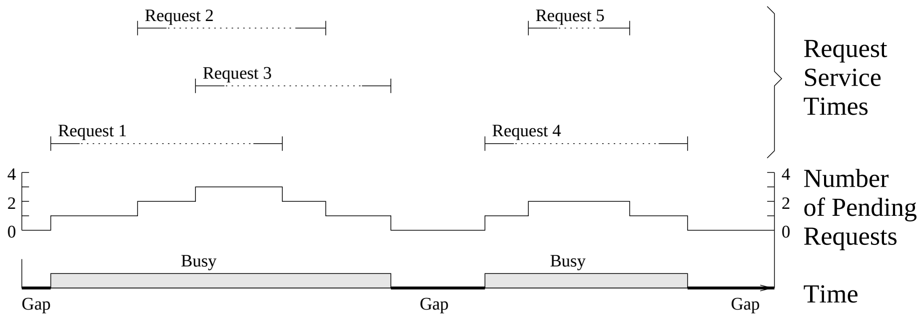

Figure 6: Measuring Utilization via Gap Estimation

4.2.2 Bottleneck Resource Utilization Control

Adaptation software described so far implements mechanisms for measuring server load that can be used to toggle between two modes of operation; a high quality delivered content mode, and a degraded content mode. This bi-modal nature makes it impossible to achieve adequate server utilization when adaptation takes place. When overload is detected all client requests are adapted potentially making the server underloaded. This effect is shown in Figure 7 by comparing the delivered bandwidth of our server to that of a non-adaptive server for different requests rates. Requests were for 64KB URLs, whose adapted version was only 8KB. Each data point in this figure represents a separate experiment where the two servers were subjected to a fixed request rate (indicated on the horizontal axis) for a fixed time duration (2 minutes). The delivered bandwidth of each server was then averaged over the duration of the experiment and plotted for the corresponding request rate. Note that the average bandwidth delivered by our server is much lower than that of the non-adaptive server at higher request rates. This is due to adaptation by switching to less resource-intensive content. Adaptation is triggered at slightly different times within each experiment (typically after 2-5 seconds from the start) which explains why the lower curve is not smooth. (It includes slightly different fractions of unadapted traffic at different data points.) Finally, note that starting at about 460 req/s even the adapted content saturates the machine, as noted by the decline in the delivered bandwidth of the adaptive server after 460 req/s. Compare this figure to Figure 5 which shows the percentage of failed requests for each server during the same experiments. It can be seen that while the adaptive server continues to serve all requests successfully until a much higher rate, it delivers less aggregate bendwidth thus underutilizing resources.

We would like to avoid overload without underutilizing the server. Our approach to achieve this goal is to degrade only a fraction (rather than the entire population) of clients. The fraction can range anywhere from "no clients degraded" to "all clients degraded". It is determined by a utilization control loop that regulates automatically the number of clients degraded so that the desired server utilization is achieved. We used a simple PI controller to stabilize the utilization control loop. The set point of the control loop is the desired server utilization. The controller compares this set point to the measured utilization and outputs a number G in the range [0,1] that controls the fraction of all clients to be degraded. The transfer function of the controller and its tuning approach are omitted for space limitations. The reader may consult a standard controller tuning reference. The controller is implemented in the adaptation agent process. It is executed periodically and its output, G, is written into shared memory. When a server invokes the *read*() socket library call to receive the next request, the call will inspect the current value of G to decide which content tree the request should be served from. The URL name in the request header is then prepended by the corresponding tree name forming the actual URL name the server will serve. This mechanism subsumes symbolic link manipulation described in Section 4.1. It is more flexible in that it allows serving different requests concurrently from different content trees. Below, we present two different ways to choose a content tree for each request in accordance with controller output, G:

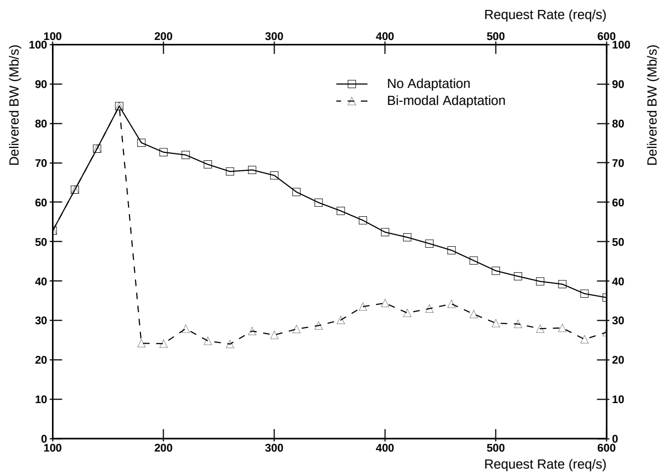

Figure 7: Server Underutilization

- *Random (with uniform distribution):* Upon the arrival of each request a random number N is generated with uniform distribution in the range [0, 1]. If $N > G$ the requested URL will be appended by the high quality tree name "/full_content". Otherwise it will be appended by "/degraded_content". Thus, N determines which tree the client is served from. Since N has uniform distribution $G\%$ of all requests will be degraded. From the client's perspective this method may result in fluctuations between good and degraded content over the duration of the client's session, as the quality of content is determined independently for each request in the session using a random function. Such fluctuations may be distracting. It may be more desirable to be consistent in the quality of content presented to a given client.
- *Client-identifier hashing:* A client can be identified using a cookie, or an IP address. A hashing function

$h()$ is applied to the client's identifier that transforms it into the number N in the range [0, 1]. As before if $N > G$ the requested URL will be appended by the high quality tree name "/full_content". Otherwise it will be appended by "/degraded_content". This method provides a consistent quality of content for each client accessing the virtual server since each client is always hashed into the same number. As long as the load on the server does not change all requests of a given client will always be served from the same content tree. The method does not guarantee that G% of the requests will be degraded but it does ensure that when G increases the percentage of degraded clients will monotonically increase which is the condition needed for the self-regulating action of the PI controller to stabilize the control loop.

Figure 8 depicts the achieved utilization of a server that uses our utilization measurement and control mechanisms. The degree of degradation is managed by the PI controller. In this experiment, the request rate on the server was increased suddenly, at $time = 13$, from zero to a rate that offers a load equivalent to 300% of server capacity. Such a sudden load change is much more difficult to deal with than small incremental changes, thereby stress-testing the responsiveness of our control loop. The target utilization, U_t, was chosen to be 85%. As shown in Figure 8, the controller was successful in finding the right degree of degradation such that measured server utilization remains successfully around the target for the duration of the experiment. The experiment demonstrates the responsiveness and efficacy of the utilization control loop.

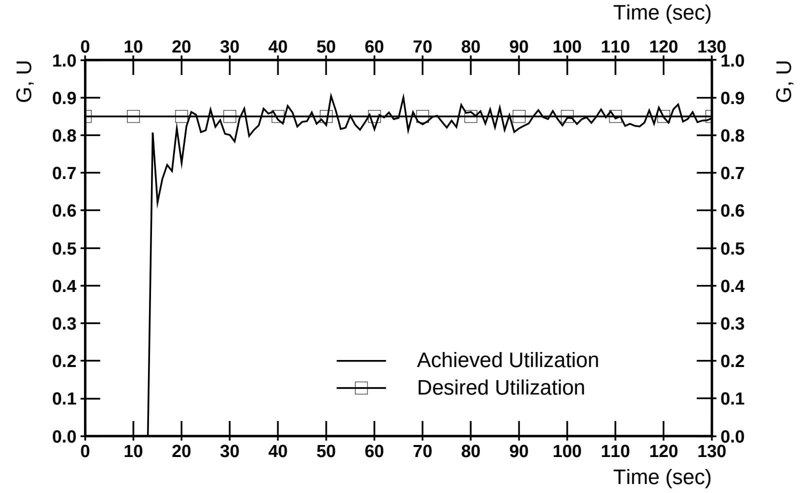

Figure 8: Utilization Control Performance

5 Conclusions

In this paper we evaluated the potential for content adaptation on the web and presented a solution to the web server overload problem that relies on adapting delivered content. Unlike present day non-adaptive servers, and unlike servers that implement admission control, content adaptation enables a server to continue providing less resource-intensive service to *all* clients during overload periods. No modifications to existing server software is required, and adequate potential is demonstrated for automating adaptation techniques to create degraded content at minimal or no additional cost to the content provider. The expected resource savings may be significant as suggested by an 80 site web survey. In this paper we report measurements using the HTTP 1.0 protocol version. It should be interesting to see how persistent connections in HTTP 1.1 affect our results. Sending small objects in bulk may achieve high performance gains. More measurements are required to assess the potential of this approach.

References

[1] T. Abdelzaher and N. Bhatti, "Adaptive content delivery for web server qos," in *submitted to the International Workshop on Quality of Service*, London, UK, May 1999.

[2] D. Andersen and T. McCune, "Towards a hierarchical scheduling system for distributed www server clusters," in *Proceedings The Seventh International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.

[3] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *Third Symposium on Operating Systems Design and Implementation*, New Orleans, Louisiana, February 1999.

[4] M. Colajanni, P. S. Yu, V. Cardellini, M. P. Papazoglou, M. Takizawa, B. Kramer, and S. Chanson, "Dynamic load balancing in geographically distributed heterogeneous web servers," in *Proceedings of 18th International Conference on Distributed Computing Systems*, pp. 295–302, Amsterdam, Netherlands, May 1998.

[5] M. Colajanni, P. Yu, and D. M. Dias, "Scheduling algorithms for distributed web servers," in *Proceedings of 17th International Conference on Distributed Computing Systems*, pp. 169–176, Baltimore, MD, May 1997.

[6] P. Druschel and G. Banga, "Lazy receiver processing (lrp): A network subsystem architecture for server systems," in *OSDI '96*, Seattle, Washington, October 1996.

[7] R. S. Engelschall, "Balancing your web site. practical approaches for distributing http traffic," *WEB-Techniques*, vol. 3, no. 5, pp. 45–52, May 1998.

[8] R. T. Fielding and G. Kaiser, "The apache http server project," *IEEE-Internet-Computing*, vol. 1, no. 4, pp. 88–90, July 1997.

[9] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir, "Adapting to network and client variability via on-demand dynamic distillation," in *Proceedings of the Seventh ACM Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, Massachusetts, October 1996.

[10] A. Iyengar, E. MacNair, and T. Nguyen, "An analysis of web server performance," in *GLOBECOM*, volume 3, pp. 1943–1947, Phoenix, AZ, Nov 1997.

[11] J. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," in *USENIX '96*, San Diego, CA, 1996.

[12] J. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, no. 3, , August 1997.

[13] A. Mourad and Huiqun-Liu, "Scalable web server architectures," in *Proceedings Second IEEE Symposium on Computer and Communications*, pp. 12–16, Alexandria, Egypt, July 1997.

[14] S. Schechter, M. Krishnan, and M. D. Smith, "Using path profiles to predict http requests," in *7th International World Wide Web Conference*, pp. 457–467, Brisbane, Qld., Australia, April 1998.

[15] A. Vahadat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa, "Webos: operating system services for wide area applications," in *Proceedings The Seventh International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.

6 Vitae

Tarek Abdelzaher received his B.Sc. and M.Sc. degrees in Electrical and Computer Engineering from Ain Shams University, Cairo, Egypt, in 1990 and 1994. Since 1994 he has been a Ph.D. student and research assistant in the Department of Electrical Engineering and Computer Science, at the University of Michigan. His research interests are in the field of QoS-provisioning, web server design, multimedia, and real-time computing. He is a recepient of the Distinguished Student Achievement Award in Computer Science and Engineering and a member of the IEEE Computer Society.

Nina Bhatti received the A.B. degree in Computer Science and Pure Mathematics from the University of California, Berkeley in 1985, and the M.S. and Ph.D. from the University of Arizona, in 1990 and 1996. Since 1996 she has been a researcher at Hewlett-Packard Laboratories in Palo Alto, California. Her primary research interests are web servers, Internet services, quality of service, communication protocols, and distributed systems. Nina Bhatti is a member of IEEE and ACM.