Empirical Quantification of Opportunities for Content Adaptation in Web Servers

Michael Gopshtein Dror G. Feitelson

School of Engineering and Computer Science
The Hebrew University, 91904 Jerusalem, Israel
mgopshtein@gmail.com, feit@cs.huji.ac.il

Abstract

A basic problem in the management of web servers is capacity planning: you want enough capacity to be able to serve peak loads, but not too much so as to avoid excessive costs. It is therefore important to know the load that web service places on the CPU, disk, and network. We analyze these loads for representative web sites, and find that with normal caching the disk is not expected to be a bottleneck, and that reducing the number of requests made is more important than reducing the total size. We then consider the option of trading off quality for throughput, as may be necessary to handle flash crowds. The suggested approaches include the elimination of graphical decorations and previews, the compression of large images, the consolidation of style sheets and JavaScript code in the main HTML page, and the removal of unimportant blocks from the design.

Categories and Subject Descriptors C.5.5 [*COMPUTER SYSTEM IMPLEMENTATION*]: servers; H.3.2 [*INFORMATION STORAGE AND RETRIEVAL*]: Information Storage—File organization

General Terms Design, Measurement, Performance

Keywords Web server, Overload, Throughput, Degraded service

©2010 ACM. This is the suthor's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in SYSTOR 2010.

1. Introduction

The success of many commercial as well as non-profit web sites depends on the number of visitors to the site and on the ability of these users to complete their intended *transaction*, be it purchase in an online store, or reading an article on a news site. This goal is reflected by the ability of the users to perform the whole sequence of HTTP requests required for such a transaction. Failure of the user to get a proper response in one of the steps will put the whole transactions at risk and possibly lead to financial loss. web site owners therefore spend considerable effort to be able to serve every single request. This includes various optimization techniques implemented in the web servers, and hardware overprovisioning which aims to provide high availability and acceptable response times in cases of peak load on the system.

Load fluctuations imply that a large fraction of the available facilities are actually unused most of the time. This is partly due to the normal periodic fluctuations in the average load, which are a function of the time of the day and day of the week. But a larger problem is the phenomenon of "flash crowds", where a very large number of web surfers converge to one site and cause a surge in the load. A possible cause for such flash crowds is the *Slashdot effect* [19]. This occurs when a popular web site posts a link to a smaller site, such that the number of users following that link largely exceeds the usual load on the smaller system, in some cases causing it to become unavailable.

Exceptionally high loads can also occur as a result of singular events. One rather extreme example is the traffic buildup at CNN.com during the 9/11 tragedy [12]. Measurements showed that traffic increased exponentially, and the number of HTTP requests was dou-

bled every 7 minutes: it grew from less than 85,000 hits/second to 229,000 hits/second in just 15 minutes. The description of the actions taken by CNN staff to continue to serve all incoming requests is illuminating. Among them were increasing the number of web servers from 10 to 52 in a short time interval, and shutting down monitoring software to free additional resources for the web server processes. In addition, the content of the home page was reduced, until the whole page consisted of only 1247 bytes of HTML, a logo, and a small picture.

Following this example, our goal is to understand *how the structure of a web page influences the work required to serve it*. We start with a detailed analysis of the costs of different types of HTTP requests, in terms of resources consumed by the web server to process them. This is augmented by a survey of a set of real web sites in order to estimate the distribution of the different types, both as a number of individual HTTP requests and the total size of retrieved data. Next we consider a number of techniques for content adaptation, and estimate the overall performance gain of each, based on the model of the costs as resulting from the initial analysis. This can be used to create optimized versions of web pages, and guide or even automate actions like those performed by the administrators of CNN.com. The scope of our current work is limited to static resources and results do not automatically apply to dynamically generated content, although some of the conclusions are useful in the general case.

2. Related Work

Most of the research published so far on the subject of improving the performance of web sites focuses on functionally transparent optimization techniques, which generally allow the servers to cope with higher numbers of concurrent clients and to achieve lower response times, while preserving the same level of content quality. Optimizations of this type usually aim to reduce the resources needed to serve the expected load, and are not used to handle peak loads. In particular, it would be always beneficial to implement such optimization methods in the areas where they are applicable. Sounders [18] summarizes some of these techniques, including the following:

- Make fewer HTTP requests, by combining multiple script files or multiple stylesheets into a single file. It is also possible to pack several small images into a single image file, and crop the required images from it on the client side. For example, this is done by Google: all the graphical elements needed to render a results page are downloaded by the browser as a single image file as shown in Figure 1.

Figure 1. *Composite image containing multiple graphical elements used to render the Google search results page*

- Add an “expires” header with a suitably long horizon that allows the browser to cache the content instead of requesting it again if the user returns to the same page later. This is especially useful for resources that are needed repeatedly, such as logos, script files, and stylesheets.
- Compress components in order to reduce network bandwidth. For example, compression typically reduces the size of textual content by 70%. On the other hand, compression requires additional CPU cycles on the servers, so it may not improve performance of the server when the CPU is the system bottleneck.
- Make JavaScript and CSS external to web pages, to enable them to be cached and reused. This is more efficient than embedding them in the HTML file.
- Configure ETags. These are part of the HTTP/1.1 specification, and allow to tag HTTP resources with additional information such as version number or checksum. Such information enables browsers to verify whether they already have the latest version of a page; in such cases the server is spared the need to send the data again.

In addition to this general set of rules of thumb which can be considered for any web site, all modern web servers can be tuned using various configuration parameters to the best performance based on specific hardware resources and functional requirements. The configuration includes parametrization of in-memory caching of resources by the server, operating system objects (e.g. the number of processes and threads), network parameters (TCP connection timeouts), enabling

various HTTP protocol features (compression, keep-alive connections), etc.

All the above optimizations are transparent to the user. If this is not enough, two approaches are possible: admission control or content adaptation. Admission control reflects the notion that it is better to serve only a subset of the clients, but maintain the quality of service they each receive [5, 11, 13]. One approach to do this is to use a model of the response time perceived by the user, and how it relates to events that can be monitored at the server [15]. If the model indicates that response times are becoming unacceptable, a fraction of the requests can be refused. However, doing this at the level of individual requests may be counter-productive, as users typically engage in sessions that involve multiple requests, and moreover, the longer sessions may be the more important ones. therefore session-based admission control should be considered [5].

Admission control may be required if service-level agreements that mandate a certain quality of service are in effect. However, it may be ill-advised in cases where it is important not to reject any clients so as not to lose business opportunities, as may be the case in e-commerce applications. The alternative is therefore to consider *trading off quality for throughput*. The danger is that if this is not done explicitly, the server will be overwhelmed and lose control over the level of service it provides. A number of articles have suggested this approach, using the notion of content adaptation. The common feature of all these methods is the attempt to create a *lighter* version of the original web site which will allow serving a larger number of users [1, 2, 17].

For example, a number of content adaptation techniques are presented in [1], such as *adaptation tags* inside HTML files, and image quality degradation by lossy compression. The authors emphasize the importance of the number of embedded objects on the overall site's performance, and suggest eliminating small cosmetic items from the page. Another technique is reduction in the number of internal links, thus making the users consume less content from the given web site, albeit it is debatable whether web-site owners would adopt such a policy. The authors further present a complete system for semi-automated content adaptation, which includes measuring server response time as a trigger to adaptation activation, and a content adapter which creates a copy of the original directory tree while replacing the content with adapted versions.

Our work also subscribes to this approach. Our contribution is in measurement-based analysis of the contents of web sites and the costs to serve them. The data that we obtain can then be used to create effective adaptation schemes that achieve the best improvement in throughput for the least degradation in quality.

3. Cost of Serving HTTP Requests

In this section we will estimate the system resources required to serve an HTTP request by a web server. These quantities are highly dependent on the properties of the requested web object, e.g. the size of returned data, computational resources for dynamic pages, etc. Thus we will present a model of resource consumption of HTTP requests as a function of the request's parameters. The model will be used later in order to estimate the expected benefits of various content modifications, that are supposed to reduce resource usage.

3.1 Context and Assumptions

The system resources that we will measure are:

1. Network bandwidth
2. CPU time
3. Disk utilization

In addition, we will measure response time. Although this quantity can't be classified as a system resource, it is important to include this parameter in the model for a number or reasons:

- The response times experienced by the user largely contributes to the perceived quality of the web site;
- Response time reflects the time a request spends on the server, and can be used for estimating the number of concurrent requests served by the server as a function of total number of requests per time unit.

At this stage we focus on the delivery of static web content. While dynamic web sites that generate content from databases are increasing in number, static content is still prevalent. Moreover, our measurements for static content can also be used to guide the scripts that create dynamic content. The main difference is the need to factor in the resources needed by the scripts themselves.

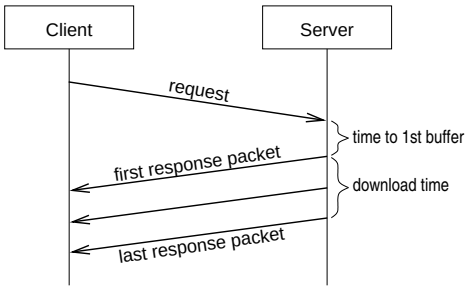

Figure 2. *Elements of serving an HTTP request*

For static web pages, images, etc. the most dominant parameter would be the total size of the returned file, e.g. the amount of resources required to serve a request for a 5 KB image file is identical to that required to serve a 5 KB HTML page. As such most of the resource costs will be given as a function of the size of the returned file.

The major exception to this rule is requests for textual files (HTML, CSS, etc.) when a data compression algorithm is utilized by the server. The data can be compressed on the fly for each request, or there can be a cache of compressed content, allowing reducing the CPU resources required for data compression. For example, caching of compressed content for static pages is available on IIS servers starting from version 6, and is enabled by default for static content in version 7.

One of our major assumptions is that the site is already optimized using methods which do not cause quality degradations. In particular, we'll assume that for frequently accessed pages the content is already compressed in advance and is stored in the cache. Thus we'll refer to the compressed size as the file's size, and ignore the issue of compression in the sequel.

3.2 Measuring Response Time

Our cost estimates are based on measuring the response time for downloading various elements of web pages. Figure 2 shows a high-level overview of a single HTTP request and its response. In this example the time the request is active on the server consists of

1. Time to 1^{st} buffer
2. Download time

The time to 1^{st} buffer is, generally, a measure for the time it takes for the server to parse the request and to locate the response content (a file for static web sites). The download time is the time required to transfer the file back to the client, and (given certain server's state and properties) is a function of the size of the file being transferred, as well as of the quality of the network connection between the server and the client. As seen in the Figure, the download time can be effectively measured on the client side, assuming that on average the latency between the server and the client does not significantly change in the course of a single response.

It is important to note that the *perceived* response time may include TCP connection establishment time, as well as the time required for the browser to render the web page. In addition, the client usually measures the total time taken to download a whole page, including all additional components. But we can avoid this by designing a tool that just measures the time to download specific files.

In order to analyze web server response times, an automatic tool was created to measure download times from a random sampling of web sites. Previous approaches to random sampling of the web use random walks [3]. However, given the size of the web this is strongly affected by where one starts. We therefore emphasize randomization of start points, as follows:

1. Select a random web site. This step is based on the "Random article" link available on Wikipedia (http:// en.wikipedia.org/wiki/Special:Random). The title of the returned article is used as a search term in Google. The first link is then chosen from the results set, and only the "host" part of the URL is taken, in order to make a request to the home page of the site.
2. Request the home page of selected sites. The URL selected in the 1^{st} step is presented to Internet Explorer, which makes the request to the server, and also performs all subsequent requests to additional resources required to render the page.
3. Collect statistical data. The tool intercepts all system calls made by the browser to open a connection to the server and send and receive data. This information is aggregated for each HTTP request and processed to measure the required parameters. The parameters collected for each request are:
   - The requested URL
   - The domain ("Host" HTTP header)
   - The transfer data size ("Content-Length" HTTP header or size of chunked response)
   - The content type ("Content-Type" HTTP header)

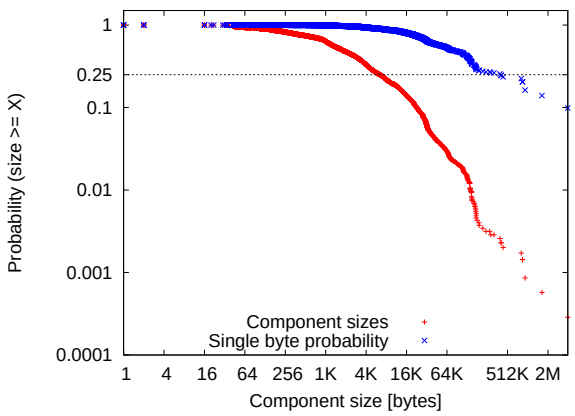

Figure 3. *HTTP component sizes distribution*

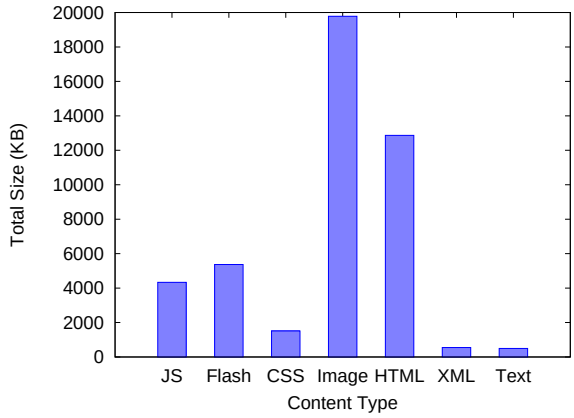

Figure 4. *Distribution of content types*

- The download time (the time between the first and last successful "read" operations on the TCP socket)

4. Perform off-line analysis. The results of the previous step are stored in a database for further off-line analysis.

We used this procedure to collect data from 95 web sites. Figure 3 shows the log-log complementary distribution (LLCD) of sizes of files returned by these servers (for each size the graph shows the probability that files will be larger than this size). The tail of the distribution of sizes is approximately straight, indicating a Pareto distribution. We also show the "single byte probability", i.e. for each size x what is the probability that a transferred byte belongs to a web-page component that is larger than x. This shows that about a quarter of the total traffic bytes come from files larger than about 200 KB, which have a probability of only about 0.2%. This information can be used as a hint later in the work, as it shows that by eliminating a small number of the largest files we can save a large percentage of the required bandwidth.

Figure 4 shows the distribution of requests by content type. Textual content (including HTML, XML, JavaScript, CSS and general text) is 44% out of the total number of downloaded components, and the fraction of multi-media files is 56%.

In order to see the response time as a function of component size, we have to choose only the requests made to the same web server, otherwise the differences in network latency and bandwidth would affect the accuracy of the results. Figure 5 shows results for servers *i.usatoday.net* and *image.timeinc.net* (note: for some

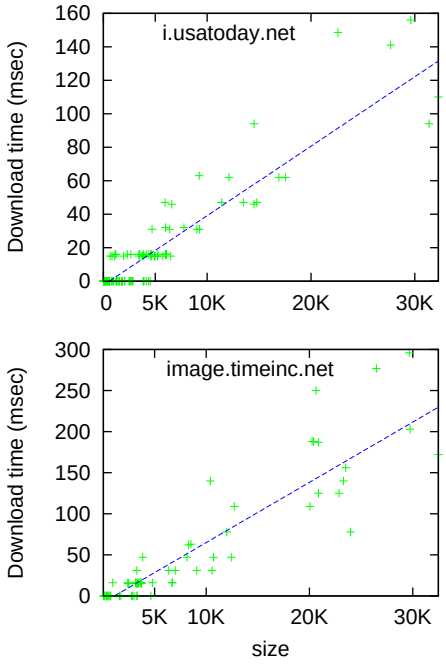

Figure 5. *HTTP component download times*

small objects the time appears as 0 msec; this happens when all the data is received by the browser in single read system call). The download times appear to have values which are multiple of 18 msec. The reason for this phenomena is unknown, but we assume it is related to scheduling mechanisms on the Windows operating system. Nevertheless, we can see the trend — as expected, the download time is approximately a linear function of the file size.

3.3 Bandwidth

As was shown in [16], the time required to route a single IP packet from source to destination is hardly

affected by the size of the packet. Therefore the more relevant metric for content size is the number of packets required to transfer it. One immediate implication is that once the whole response can be sent in a single packet, the bandwidth consumed by this request (as well its response time) can't be reduced any more by reducing the size of the response. In order to estimate the size of data which can be sent in the first response packet, we should take into account:

1. The total size of packets that can be transmitted. As servers are typically connected by Ethernet LANs, this is 1538 bytes. This includes a 12-byte gap between successive packets.
2. The size of various headers. Ethernet itself requires 14 bytes of headers. IP/TCP require an additional 40 bytes.
3. The average size of HTTP response headers. Examples of HTTP header lengths in our measurements of sites using the Apache web server are 280–325 bytes from `www.cnn.com` and 296–300 bytes from `www.top500.com`.

Thus we can conclude that files under a size of 920 bytes are expected to be downloaded in a single packet. The general formula for the maximal file size transmitted in N packets would be $920 + (N-1) * 1258$ bytes. According to data shown in Figure 3, approximately 43% of the responses fit into a single response packet, and an additional 24% fit into two packets.

3.4 Locality of Requests and Disk Utilization

Understanding the most common patterns of websites' usage is crucial to our ability to produce realistic estimates for consumption of resources by the web server. Our goal is to derive guidelines for handling cases of unexpectedly high peaks in the number of users visiting the site; in cases when a site is continuously overloaded it probably means that it should perform a more accurate sizing of the system.

It is reasonable to expect that in cases of extreme load the requests will show a high locality — the vast majority of requests will map to a very small number of web pages or files. The examples shown above support this assumption: visitors of CNN.com during the events on 9/11 were mainly interested in viewing the home page of the site only, and when a peak is caused by a Slashdot effect, the majority of visitors follow an external link, leading to a specific page in the site.

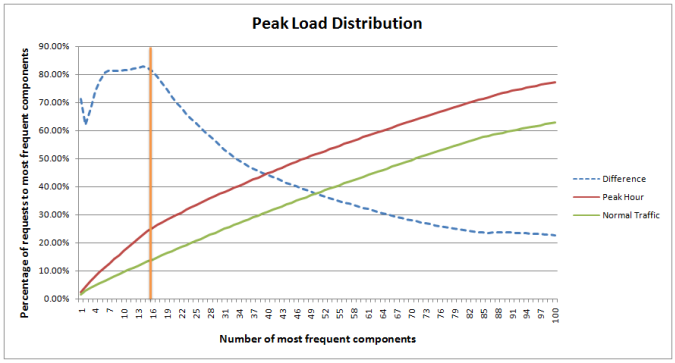

Figure 6. *Peak load distribution from the official site of the World Cup '98 games*

In order to validate the assumption of locality, we've analyzed web-logs collected from web servers serving the official site of the World Cup '98 soccer games in France, which was created for a short time interval and was dedicated to that event. Figure 6 shows the percentage of HTTP requests to the most frequent files from the total number of requests to the site. The 15 most frequent components make 13.4% of the total requests during the normal load, compared to 24.2% during the peak hours (the most frequent components might be different in normal and peak loads). Starting from the 30^{th} most popular component, the graphs begin to be approximately parallel, which shows that frequencies of non-top pages do not significantly change on peak loads.

The most noticeable implication of locality of requests is when caching of response data is applicable, because a high percentage of cache hits may be expected even with moderate cache sizes. This has implications especially for disk utilization. If we consider the default configuration for IIS 6 server [14], it allocates approximately half of available physical memory for caching, while a size of a cached file is limited by default to 256 KB. Assuming that a modern server is equipped with at least 4 GB of memory, it can cache more than 8,000 files of maximal size, and on average the fraction of files larger than that size is only 1/3%.

Based on this data we conclude that the disk utilization won't be a bottleneck for performance of the web server at peak loads, and it can be taken out of the model.

3.5 CPU Utilization

As stated above, the type of file being requested does not affect the amount of web server's resources re-

				jiffies		
	C	P	T	user	syst	total
New connection	+			68	88	156
Large file (once)		+	+	11	403	414
Small file	+	+	+	333	269	602
Medium file	+	+	+	345	277	622
Non-existing file	+	+		335	250	585

Table 1. *Summary of CPU consumption in test cases, typically involving 10,000 repetitions. C=connection, P=processing, T=transmission*

quired to serve it. In order to reach a more accurate model, there's a need to break down the CPU time into different components of a single HTTP request. The major parts of request processing by a web server, when referring to static files, are:

1. Opening a new TCP connection
2. Parsing request headers
3. Locating the requested file
4. Network communications: receive the request and transmit the response

The first three are largely constant, but the last depends on the file size.

To measure the different components, a number of different load types were applied to a web server. For each test case the total CPU time of all the server's processes was measured, and the difference in this time before and after the load is considered as the CPU time required by the server. The server used for the tests was Apache 2.0 running on Red Hat Enterprise Linux 5.0. All *httpd* processes were monitored, and for each of them the data was taken from the */proc/PID/stat* pseudo-file. In particular we used *utime* and *stime*, indicating the number of jiffies that the process has been scheduled in user and kernel modes respectively. Preliminary tests have shown that such measurements show consistent results with only a slight differences (up to 4%) between different tests with the same load on the server. The client used in all tests was Internet Explorer 7.0.

The measurement results are summarized in Table 1. The test cases used are

1. Opening a new connection. The purpose of this test is estimating the cost of opening a new connection by a client. In the final calculations we should take into account that usually the same TCP connection is reused by the browser to download multiple HTTP objects. In the test a new TCP connection was opened to the server, and closed immediately afterwards, repeated 10,000 times.
2. Downloading a big file. This test comes to estimate the cost of processing time required for data transfer, when the same TCP connection is used to deliver large data volumes. In this test a single textual file of 1GB was downloaded by the client.
3. Requesting a small file. In this case a small HTML file (73 bytes) was requested 10,000 times. The sizes of request and response HTTP headers were 315 and 253 bytes respectively, so a total of approximately 6.1 MB of traffic was transferred, and a new TCP connection was opened for each HTTP request.
4. Requesting a medium file. This case is similar to previous one, but a larger file was downloaded.
5. Requesting a non-existing file. This final test case comes to measure the amount of CPU time needed to return a "File not found" response (HTTP code 404). The server's response also contains a short HTML content describing the error to the user (287 bytes). The sizes of request and response headers were 310 and 180 bytes respectively.

From comparison of test cases (3) and (4) we can estimate the amount of CPU required to process the request and to deliver the file contents. We first eliminate the cost of opening the new connections, by subtracting the CPU time measured in test (1). The conclusion is that the total CPU time required for 10,000 requests for 10 KB files is 466 jiffies, 446 of them (96%) to parse and process the request, and only 20 (4%) to transfer 10 KB of cached data.

If we take into account the CPU time required to open a TCP connection, we find that an HTTP request for a 10 KB file requires 25% of its CPU time for connection initiation, 72% to parse and process the request, and the remaining 3% to transfer the response. According to [7], the average number of requests per connection is 27.6; taking this into account reduces the relative weight of connection establishment and leads to a CPU time distribution of 1% to establish the connection, 95% for parsing and processing, and 4% to transfer the response content.

The CPU consumption for file transfer generally depends on its size: the smaller it is, the lesser is the

percentage of CPU cycles required for network transfer out of total cycles used to process the request. We have shown that for a 10 KB file the parsing required 24 times more CPU time than the transfer, so by extrapolating this result, we conclude that in order for the network transfer to take half of the total CPU consumption, the size of a file should be 240 KB (24 times 10 KB). The analysis of HTTP component sizes from random websites shown above (Figure 3) indicates that the probability of a component to be larger than 10 KB is less than 20%, and to be larger than 240 KB only 0.3%. Thus we see that the lion's share of the time is spent on parsing and processing, and only a small part on transfer. Therefore, when CPU utilization is the bottleneck, we should look for ways to reduce the number of HTTP requests made to the server, as the expected benefit of this approach is significantly larger than that of reducing the size of individual components.

Note that these results were obtains for static web pages. With dynamic web pages, the CPU consumption is expected to be even higher. However, specific optimizations related to the dynamic generation of web pages are beyond the scope of this work.

4. Optimizations to Increase Throughput

In this section we present different methods for optimizing web pages in terms of server resources required to handle all related HTTP requests. Each optimization method is applicable to certain types of page components. Using the results from the previous section, we will estimate the expected impact of each optimization on the utilization of each resource type. We also consider the expected reduction in quality of the web page as perceived by the end user. In order to perform calculation of the over-all effect of the suggested optimizations, we will consider a representative distribution of component types in the page.

4.1 General Approach

Considering the focus on handling flash crowds that converge on a limited number of pages that may be cached by the server, we conclude that the bottleneck resources are either the network bandwidth or the CPU. For each component of the original web page, the following optimization approaches are possible:

1. Reduce the size of the returned file
2. Eliminate the HTTP request to the component

For small and medium file sizes we expect that reducing the file size will lead to some positive effect only on network bandwidth requirements, but not to a significant reduction in CPU utilization, as most of the processing resources are consumed by the server to parse the request and locate the file to be sent as response to the client. There's a much higher motivation to reduce the size of large files. As shown in Figure 3, large components, although being rare, are responsible for a large portion of the total traffic, hence we are interested in reducing their size as much as we can. When transmitting a large file, the CPU consumption also represents a considerable overhead.

Alternatively, by reducing the number of HTTP requests made to the server, we can significantly reduce the CPU time consumption, as was shown in the previous section. Depending on the size of the eliminated component, total network bandwidth will also be reduced, as in the case of size reduction.

Although we make the assumption that websites are generally optimized using techniques that are transparent to the users (i.e. do not reduce the quality of the web page), observations show that optimization of the number of HTTP requests is rarely performed. For example, it is relatively simple to consolidate multiple HTTP requests into a single one by combining a number of graphical elements into one large image, and cropping the relevant areas on the client side using one of the scripting technologies. The only site we know that performs such an optimization is `www.google.com` (Figure 1). This leads us to conclude that for most sites removing visual elements from the web page will actually reduce the number of HTTP requests made by the client while rendering the page.

A major issue with our suggested optimizations is their effect on the perceived quality of the web site. In general, our methods make a tradeoff between the performance of the web server and the quality of the web page. But how does one measure this quality? In order to be able to measure the impact of certain content changes, we first have to identify the main features by which the user judges the quality of the site.

The literature contains several papers that attempt to identify parameters that correlate with perceived quality. One class of parameters is those relating to speed and responsiveness [4]. In particular, one can break down the time needed to retrieve and render a page into contributions by the server, the network and the

browser. Our optimizations all reduce the amount of work done and the amount of data transferred. While the focus is on reducing the load on the server, this has the byproduct of also reducing the work required of the network and browser. Based on this we can conclude that from the "response time" perspective, the user-perceived quality will only increase when applying optimization methods as suggested here.

Another dimension of quality of a web site is the quality of its contents. Quality aspects include [10]

1. Accuracy of the information and the qualifications of the author relative to the given subject
2. Authority and standing of the publisher or institution who host the site
3. Objectivity of the provided information
4. Currency of the information, as reflected by the last update time and validity of outgoing links
5. Coverage and linking to additional information

All of the aspects listed above refer to the textual content of the page, which is not altered by our suggested optimizations, so we can deduce that the quality, as reflected by these considerations, is preserved.

Finally, the most elusive quality indicators are related to design of the page, including the use of text, links, and graphical elements to create a design that supports information delivery, navigation, and promotes a positive user experience [9]. Optimizations discussed in the current work preserve the general layout of a web page, including links and text formatting, but introduce various changes to non-textual content elements, from elimination of some of them to reduction the quality of the others. The subject of visual quality is rather subjective, so for us the guideline was to try and make the optimized version look as close as possible to the original page. Below we show examples of both acceptable and unacceptable optimizations.

4.2 Images

Image files make up about a half of a website's content total size, thus becoming a prime target for optimization. In order to select the best optimization method for each image, we have first to classify all images according to their *function*. Based on previous works on the subject, and observation of different websites, we identify the following classes of images [8]:

Figure 7. *Classification of images from www.cnn.com: (a) story, (b) preview, (c) host, (d) logo, (e) decoration, (f) commercial*

Story : along with related textual information, such images form a central part of the web page, being an illustration for an article, or in some cases becoming the main content.

Preview : has a similar purpose as the previous item, but is used with references to a different content, e.g. linking to another article on the same site.

Commercial : used in advertisements or links to sponsors of the site.

Host : a photo of the content's author, used in the main content, in links to other articles, in blogs, etc.

Logo : the logo of the website, usually containing the publisher's trade mark.

Decoration : images used as elements of the website's design, e.g. drawing a frame or serving as bullets for a list of items

Navigation : a special class of decorations used on navigational buttons.

Text : such images contain text in graphically enhanced formats, usually used for short text sequences such as titles or menu items.

Figure 7 shows examples of different image classes.

In order to assess the potential for eliminating or reducing the sizes of images, we need a database of classified images found on different web sites. To obtain this a special interactive tool was created, with the following functionality:

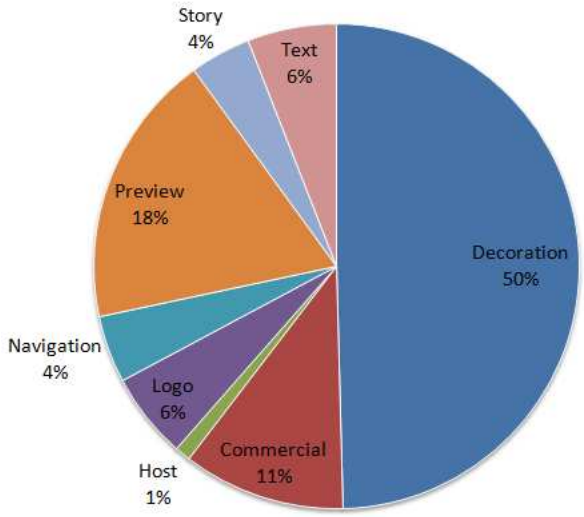

Figure 8. *Image types distribution*

1. Open and display the home page of a random website (using the methodology of Section 3.2).
2. Find and highlight the first image, which can appear in the HTML code as an <IMG> tag, or as a background of other tags.
3. The user clicks on a shortcut key to select one of the classification types, which describes the current image most accurately.
4. The class of the image, as well as the following parameters, are stored in the database:
   - HTML tag
   - Image source URL
   - ALT property of the image (if present)
   - Image file format (GIF, PNG, BMP, JPEG)
   - Number of colors (for GIF images only)
   - URL of the link associated with the image (if any)
   - Size of the image in the browser after rendering
   - Size of the original image file
   - Number of times this image appears on the page
5. This process is repeated for all images on the web page, and then the next website is chosen.

Using this tool, 959 images from 30 sites were analyzed. 50% of them were classified as decoration, 18% as preview, 11% as commercial, and 21% as other types, see Figure 8.

Considering these results and the semantics of each image type, we can define the strategy we would like to apply to images while optimizing a website.

Decoration and *preview* images are the best candidates to be removed from the page, as it will not affect the main content, and the expected number of such images constitutes about 70% of the total number of images on the page. This will lead to significant savings of both CPU and network capacity.

At the same time, we would like to preserve all *navigation* images, as the functionality of the page depends on them. The same is true regarding *text* images, although in principle such images can be replaced with a text box containing the same information, but embedded in HTML code and thus not requiring any additional HTTP requests to be made. We would also like to assure that *logo* and *story* images are not removed from the page because of their semantic importance.

The handling of *commercial* image content is an open question, and depends on the site. On one hand, these images are not an integral part of the content of the website. But on the other, profitability of the site may depend on them. It therefore seems advisable to leave this decision to the administrator of the website, making it a configurable parameter in the process of optimization.

The alternative to removal is reduction in size. All large images that are not removed should be made smaller by reducing the resolution and/or using lossy compression. This is especially relevant for story images, preview images, and commercial images, which tend to be big. As shown by [1], a reduction by a factor of 10 may be quite possible with little noticeable effect on aesthetics.

In order to apply the above strategies, one needs to correctly identify the function of each image. Doing it manually as we did for our research is not a viable option. We therefore attempt to define a set of rules which can be used to identify certain classes of images based on their observable properties. We will start from defining and validating some single-parameter association, and will then derive a combined definition that produces the best results. The suggested initial associations are as follows:

1. Most decoration images are encoded in *GIF* format. Specifically, there were 523 GIF images overall. 312 (60%) of them are decorations, covering 66% of all decorations, but producing 40% of false positives.

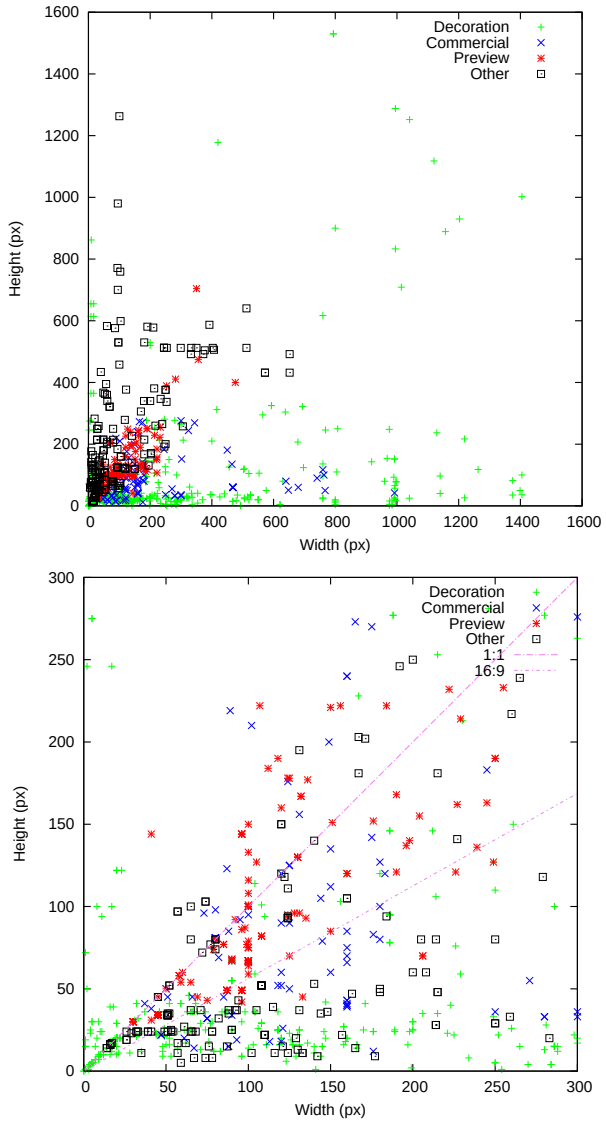

Figure 9. *Image sizes distribution and zoom in on smaller sizes*

2. Mainly decoration images appear in HTML tags other than IMG. There were a total number of 312 such images. 266 (82%) are decorations (56% of all decoration images).
3. The same decoration image can appear more than once on the same page. 148 images had multiple copies on the same page, 139 (94%) out of them were decorations (29% of all Decorations).
4. Tests for presence of an ALT property for images, or for presence of a link associated with the image, did not produce meaningful results.

The next set of associations is related to geometrical properties of images: pixel width and height of the

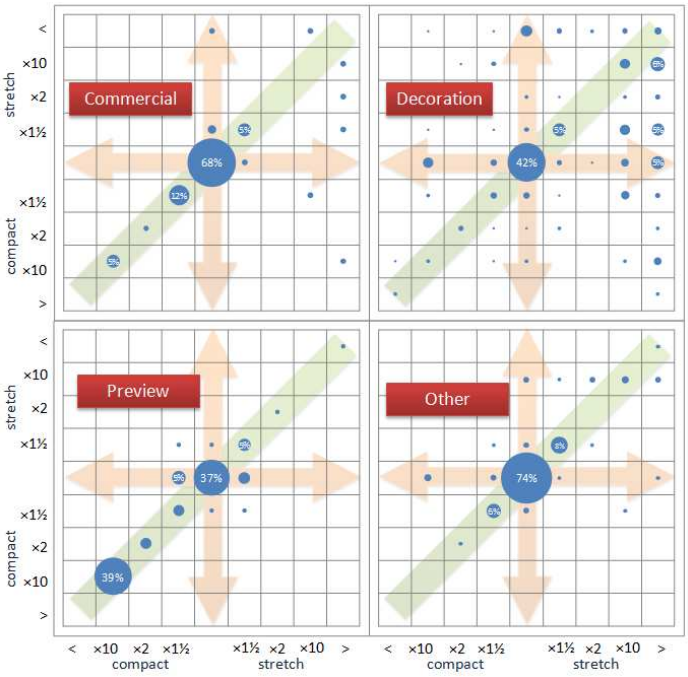

Figure 10. *Image sizes: original file vs. HTML*

original image file and of the area covered by the image in the rendered page. We pay special attention to the aspect ratio between the width and the height, and to how this metric changes from the original file to its final manifestation on the web page. Figure 9 shows the distribution of image sizes as a function of image type. Figure 10 shows how the size of the image changes when rendered by the browser, both horizontally and vertically, for different image classes. The size of the "bubble" is proportional to the number of images falling into a given bucket: the central square refers to the images whose size is the same both in original file and in the HTML page, next cell on the right includes images that were enlarged horizontally by the factor of up to 1.5, and so on. For the images falling into the main diagonal the aspect ratio between the width and the height is preserved.

As we can estimate based on the charts, most of the images that are enlarged significantly relative to their original size are decorations. Another noticeable feature is the large number of Preview images which are compressed inside the HTML page, while preserving the original aspect ratio. This can be explained by the fact that in some cases the same image file is used in its full size as a story image in one page, and as a preview image in another. It also shows that a large number of

preview images could be reduced in size without any reduction of the quality of the hosting web page.

Based on these observations, we can make additional assumptions regarding expected geometrical properties of different image types, and validate them against the images database.

1. Decoration images tend to be small. We checked the number of images having pixel size of original file (width $\times$height) less than 100. The results were that 162 such images found, and 156 of them (96%) were decorations (making 33% of all decorations).
2. A large change in an image's aspect ratio implies that it will be distorted. This would be unacceptable for a picture, but perfectly OK for a decoration that just provides background shading or a separating line. Therefore we expect the value of "file aspect ratio" / "HTML aspect ratio" to be in the range of 2/3 - 3/2 for most of the images, except for decorations. Indeed, of 235 images with large differences between aspect ratios in original file compared to that of rendered inside HTML page, 210 (89%) of them are decorations (44% of all decorations). Other image types with non-standard aspect ratios included text and commercial images.

In conclusion, we can define the following rule to identify decoration images:

- Appears more than once on the same page
- Pixel size of image file is less that 100
- The aspect ratio is changed by more than 2/3 when comparing that of original image with the aspect ratio of the image as it appears in the HTML page.

When applying these criteria on the image set, we achieve the following results. A total of 322 images passed this test. Of them, 289 (90%) are decorations (constituting 61% of all decoration images). Of the false positives, 12 are text (21% of all text images), 11 are commercial (11%), 4 are preview (2%), 3 are navigational (7%), 2 are logo (4%) and 1 is story (3%).

As shown in Figure 10, Preview images can be identified by the following factors:

- Aspect ratio of the image is preserved.
- The image is compacted by a factor of 2 to 10 times.

This identifies 71 images in total, 66 of which are preview (93%, and 37.5% of all preview images). Of

Figure 11. *Same page with and without style sheets*

the others 3 are commercials (3% of all commercials) and 2 decorations.

4.3 Auxiliary Content

In addition to images, web pages lead to downloads of two additional types of files: style sheets and JavaScript files. Web style sheets are a form of separation of presentation and content for web design, where the style is defined in an external stylesheet file using a language such as CSS or XSL. This also allows the same style to be easily applied to all the pages in a site. JavaScript is a scripting language used to enable development of enhanced user interfaces and dynamic websites.

In order to estimate the importance of style sheets for the quality of a web site, we took a sample page, removed all style sheet files and compared to the original

version of the page. As shown in Figure 11, removal of style sheets caused the whole structure of the page to be lost. We can conclude that style sheets may play an important part in defining the layout and format of a web page, and as such our optimization cannot eliminate these files.

JavaScript can be employed for making dynamic HTTP requests in response to certain client-side events, e.g. loading a new image when the mouse is moved over a graphical element. In some such cases we may prefer to disable this JavaScript functionality in order to reduce the number of HTTP requests when running in optimized mode. However, the analysis of JavaScript code is beyond the scope of the current work, and no JavaScript optimizations will be proposed.

It should be noticed, however, that HTML supports embedding auxiliary content inside the HTML file itself, in addition to having a reference to a separate file. Thus in order to reduce the number of HTTP requests it may be advisable to embed all auxiliary content inside an optimized version of the HTML page, even if it would enlarge the size of the file.

The principle of locality should also be taken into consideration. The main advantage (from a performance perspective) of having CSS and JavaScript code as separate files is in the ability of web browsers to reuse these files for different pages that are viewed by the client, as this content is usually static and can be effectively cached. In cases of a peak load, the average number of pages in a single user's session is usually smaller than that of regular browsing of a site, thus the rule of separating CSS and JavaScript content from HTML is less applicable in this scenario.

4.4 Textual Content

Hyper-Text Markup Language (HTML) is the predominant markup language for web pages. The markup tags affect both the formatting and the layout of the displayed page. Formatting includes visual cues such as making headings larger with a boldface font. Layout includes elements like menu bars and blocks of navigational links.

In the context of our work, an interesting option is to rewrite source pages in a way that certain parts of the page are eliminated. This is especially relevant when the page uses a modular layout [6], where the content is arranged inside vertical and horizontal shapes. Moreover, in a typical website there are a number of predefined layouts, and each page follows one of them, e.g. locating a navigation pane on the left column of each page. When generating an optimized version of the site, some of these layout modules can be eliminated from all relevant pages, such that all components of those areas are effectively removed and the total number of HTTP requests required to render an optimized page is reduced.

Automatic semantic classification of different areas of a web page is a hard problem and out of the scope of the current work. Indeed, [1] suggest that web page designers include special tags in their designs, pointing out parts of the page that are important to retain and parts that may be eliminated. However, any such scheme that relies on added burden on developers risks not being used in practice. In addition, modern web development tools put the feasibility of such a solution to question, because in many cases designers are not required to type HTML Markup manually, but rather generate it automatically using a visual tool.

A closely-related alternative is to rely on existing markup. In many cases, the layout is defined in a centralized HTML file, and the content resides in different HTML pages which are loaded as frames in the main page. This makes it easy to control the presence of whole frames in the web page. In addition, many tools and layouts use tag names or IDs that have important semantic content. Thus we can base our optimizations on eliminating certain tags (with all internal structure) based on tag name and ID. For example, in case of the `www.top500.org` site we can configure the tool to hide DIV tags having "sidebanners" as an ID, and the whole right-side pane containing advertisement images will be removed in all pages. The structure of the `www.cnn.com` site is also controlled by DIV tags, having meaningful IDs and class names, making it possible to define appropriate optimization rules. The advantage of this method, compared to insertion of special marks into the HTML file, is in its non-intrusiveness — there's no need to modify the original pages or to introduce any changes to a regular workflow of updates to the content of the site.

5. Conclusions

The only way to serve a flash crowd of users who flood a web server is to trade off quality for performance. This means that each user will receive a degraded version of the requested web page, but the server's

throughput will be improved enough so that at least they will all receive a relevant page. Such an intentional degradation of web pages is called content adaptation.

We have conducted an empirical study, based on data collected from a random selection of web sites, in order to establish guidelines for what types of adaptation may be expected to provide the most benefit in terms of reducing the load on the server. The following list summarizes the major potential areas for optimization:

- Elimination of decoration images. Decorations constitute roughly half of all images in a page, and can be identified with high accuracy by an automatic tool.
- Resizing large images. Large image files can be compressed to create a lower quality version with reduced file size.
- Consolidating textual content. It's possible to embed JavaScript files and style sheets into HTML code, thus reducing the total number of components of a page.
- Hiding whole blocks in a page layout. Based on relatively simple manual configuration, whole blocks can be removed from original pages, including all contained resources.

These optimizations reduce the number of HTTP requests, which is crucial in order to reduce CPU load. They also reduce the volume of data that needs to be sent, thereby reducing network load.

A verification of the effect of these recommendations requires a full implementation and experimental evaluation. We leave this to another paper.

Acknowledgments

This research was supported by a grant from the Israel Internet Association.

References

[1] T. F. Abdelzaher and N. Bhatti, "*Web content adaptation to improve server overload behavior*". *Comput. Networks* **31(11-16)**, pp. 1563–1577, May 1999.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "*Performance guarantees for web server end-systems: A control-theoretical approach*". *IEEE Trans. Parallel & Distributed Syst.* **13(1)**, pp. 80–96, Jan 2002.

[3] E. Baykan, S. de Castelberg, and M. Henzinger, "*A comparison of techniques for sampling web pages*". In 3rd *Workshop on Information Integration on the Web*, May 2006.

[4] N. Bhatti, A. Bouch, and A. Kuchinsky, "*Integrating user-percieved quality into web server design*". *Comput. Networks* **33(1-6)**, pp. 1–16, Jun 2000.

[5] L. Cherkasova and P. Phaal, "*Session-based admission control: A mechanism for peak load management of commercial web sites*". *IEEE Trans. Comput.* **51(6)**, pp. 669–685, Jun 2002.

[6] L. Francisco-Revilla and J. Crow, "*Interpreting the layout of web pages*". In 20th *Conf. Hypertext and Hypermedia*, pp. 157–166, Jun 2009.

[7] R. Hariharan and N. Sun, "*Workload characterization of SPECweb2005*". In *SPEC Benchmark Workshop*, Jan 2006.

[8] J. Hu and A. Bagga, "*Categorizing images in web documents*". *IEEE Multimedia* **11(1)**, pp. 22–30, 2004.

[9] M. Y. Ivory and M. A. Hearst, "*Improving web site design*". *IEEE Internet Comput.* **6(2)**, pp. 56–63, Mar/Apr 2002.

[10] J. Kapoun, "*Teaching undergrads web evaluation: A guide for library instruction*". *College & Research Libraries News* **59(7)**, pp. 522–523, Jul/Aug 1998.

[11] D. Kumar, D. P. Olshefski, and L. Zhang, "*Connection and performance model driven optimization of pageview response time*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009.

[12] W. LeFebvre, "*CNN.com: Facing a world crisis*". *;Login:* **27(1)**, p. 83, Feb 2002. (summary of invited talk at LISA 2001).

[13] N. Mi, G. Casale, Q. Zhang, A. Riska, and E. Smirni, "*Autocorrelation-driven load control in distributed systems*". In 17th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Sep 2009.

[14] Microsoft TechNet, "*Global registry entries (IIS 6.0)*". URL http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/60a90c91-a8d0-43b6-89db-a431d0ea0cb4.mspx?mfr=true. (Visited 1 Feb 2010).

[15] D. Olshefski and J. Nieh, "*Understanding the management of client percieved response time*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 240–251, Jun 2006.

[16] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "*Measurement and analysis of single-hop delay on an IP backbone network*". *IEEE J. Select Areas in Commun.* **21(6)**, pp. 908–921, Aug 2003.

[17] R. Pradhan and M. Claypool, "*Adaptive content delivery for scalable web servers*". In *Intl. Network Conf.*, Jul 2002.

[18] S. Sounders, "*High-performance web sites*". *Comm. ACM* **51(12)**, pp. 36–41, Dec 2008.

[19] Wikipedia, "*Slashdot effect*". URL http://en.wikipedia.org/wiki/Slashdot_effect. (visited 31 Jan 2010).