WPI-CS-TR-99-36 November 1999

Studying the Impact of More Complete Server Information on Web Caching

by

Craig E. Wills
Mikhail Mikhailov

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Abstract

Caching of objects in the World Wide Web is a widely used technique to reduce end user latencies, network and server load. Currently deployed heuristic-based approaches to caching are not ideal, and prior results show potential for better reuse of cached Web content. This work studies a more deterministic approach to caching of Web objects where Web servers supply more complete information to client caches about the objects served. The idea is to view container (HTML) objects as a collection of distinct objects with heterogeneous change characteristics. Relationships between these and embedded objects (images) can be exploited at the server side. Servers can generate cache invalidation information and piggyback it onto existing request/response traffic. The results indicate that these techniques can improve existing cache management strategies.

Keywords: Web Caching, Content Reuse, Object Composition, Object Relationships, Change Characteristics.

1 Introduction

This work is part of a project examining the effectiveness of current caching techniques in light of more complete data. The goal is to understand the potential of caching if improved techniques were used by Web caches and servers. This paper extends our initial work [14, 16] and focuses on investigating the potential of improved caching techniques.

Caching of objects in the World Wide Web is a widely used technique to reduce latencies for clients accessing these objects while reducing network and server load. Exported Web pages are composed of multiple components of different types and characteristics. A prime example is a "container" page, which serves as a composite object containing a multimedia of text, image, programming script, audio and video objects. Ideally the relationships between objects would be taken into account for caching. Currently servers do not provided this information to caches. Instead caches use heuristic-based approaches for deciding what to cache, how long to keep it and when cached objects may become stale.

Caches treat many objects as uncacheable because some portion of the object changes on each access. Caches use the last modified time of objects with no expiration times to assign a heuristic time-to-live value. with no expiration times. Studies showed that 15-18% [7], 30% [10] and 37% [1] of client requests resulted in 304 (cached copy is up-to-date) responses. A recent study of the 1998 Soccer World Cup Web site indicates "the lack of an efficient consistency mechanism is preventing Web servers from fully benefiting from caching" [1]. The authors note that servers must respond to a large number of cache consistency checks (GET If Modified Since requests).

Motivated by problems with heuristic approaches and our own prior work on the potential reuse of Web content [14, 15, 16], we propose an alternate approach to caching of web objects where Web servers supply more complete information to client caches about the objects served. The idea is to take better advantage of composite objects, which group a set of distinct objects each with is own type and change characteristic (changes on each access, at regular intervals, based on a data base changing, etc). Relationships between these and embedded objects (images) can be exploited at the server side to generate cache invalidation information, which can be piggybacked onto existing request/response traffic. Such information might include explicit expiration times for caching a list of objects or invalidations based on the membership in a server volume. A volume contains a set of objects grouped by some relation. It might consist of all the objects within a composite object or all of the objects dependent on another object, such as a changing database. A server can then piggyback volume invalidations onto existing request/response traffic between itself a client cache [7].

The goal of our work is to study how three techniques—composition of objects, classification of their change characteristics and exploitation of their relationships—can be integrated to make caching more deterministic. Our approach for evaluating the effectiveness of these techniques is to study a set of URLs at a variety of sites and gather information about content changes of these URLs. In addition, we gather response header information reported by the servers with each retrieved objects. To study the potential reduction of validation requests we examine logs of actual client requests. The results of our study are presented along with a discussion of their implications. We conclude with a summary of our work to date and ideas for future work.

2 Techniques for Study

2.1 Composition of Objects

The notion of composite objects is already present on the Web as an HTML document often serves as a container page where images and applets can be inserted as part of the page. However, this simplistic approach only recognizes the need for different objects based on type, rather than other object characteristics such as when an object changes. As observed in [9, 15], large pieces of an HTML object often remain the same while smaller pieces of the object actually change. Current Web caching techniques treat entire object as a whole—if any piece of it changes then the entire object must be retrieved. One part of our work is to study the amount of content reuse that could occur if larger objects were properly composed into objects of homogeneous change characteristics. Because Web objects are currently not composed in this way our approach for study is to *decompose* current HTML objects into a set of HTML objects and then examine potential improvements if these hypothesized objects were then composed back together.

There are a number of alternatives of how to represent the composition of these hypothesized object in constructing the composite object.

1. Embedded objects. The most straightforward approach is to extend the notion of embedded images to generic objects. Such an approach has already been proposed. For example, the `<ilayer>` tag introduced by Netscape allows an HTML page to be composed of multiple objects. The `<iframe>` tag proposed by Microsoft embeds one object within another, and is similar to the `<img>` tag. According to the HTML 4.0 specification the `<object>` tag allows an arbitrary object of any media type to be included into an HTML page, but this tag is not supported by most browsers.

2. Delta-encoding. Another approach that has been proposed for communicating changes in objects from a server to client is delta-encoding [9]. While this technique is general and works well for relatively small changes it does not take advantage of structure inherent in the content of an object. It can also be expensive for the server either in requiring differences to be calculated on an as-needed basis (computationally expensive) or to be pre-computed and stored (space expensive).

3. Substitution tags. An alternate to using embedded objects, which require separate object identifiers and potentially separate object retrievals, is to introduce a special tag directive in objects of homogeneous type, but heterogeneous change characteristics. This directive allows the structure of the composite object to be preserved and changes to portions of the object can be identified by the server simply as substitutions for particular tags. This approach allows the server to take advantage of the object structure in specifying the portions that have changed. This idea is similar to the one by Douglis, et al who propose an HTML pre-processor where the static portion of a page contains macro-instructions for inserting dynamic information [6]. However, we envision that a substitution tag directive would not need a separate pre-processor.

The first alternative is the most general case for handling objects of different types and change characteristics, but delta-encoding and tag substitution offer potential performance improvements in communicating object changes. As part of our work, we compare the number of bytes that need to be communicated by each technique.

2.2 Object Change Characteristics

In this study, we also explore the change characteristics of objects. In prior work we have identified a number of categories for objects based on their change characteristics [15]. These object change categories and their descriptions are:

- Periodic. These objects change at predictable periods of time such as every hour, day or week. These objects can both be cached with explicit expiration times attached to each object.
- Dependency-based. Changes in these objects are dependent on other measurable events at the server such as a file or database changing. These objects may be dynamically generated or pre-computed. These objects can be cached, but if the dependency changes then copies need to be validated before reuse.
- Dynamic. These objects change each time they are retrieved regardless of the time frame between retrievals. The implication is that these objects can not be cached. Examples include dynamically generated objects that are time-dependent.
- Access Dependent. These objects change each time they are retrieved, but the change is *access* and *not time* dependent, hence the difference from the dynamic category. An example of this type of object is one that rotates through a set of values as is done with ad banners on Web pages. The implication is that these objects cannot be cached, but it is possible that a proxy cache could prefetch an access dependent object.
- Input Dependent. These objects have a dependency on an input parameter included as part of the request for an object. In the Web, two such examples are the use of query parameters and cookies. These objects are likely to change for different query parameters, but as we found in earlier work [16], Web objects retrieved with cookies often do not vary based on different cookies. Such objects can be cached, but need validation before being reused.
- Relatively Dynamic. These objects are expected to change frequently at irregular intervals with no measurable dependency on which changes can be tracked. One example is a composite object that is being frequently edited manually to change content and layout. These objects may be cached, but copies always need to be validated before being reused.
- Static. These objects never change. They can be cached with no need to validate content before reuse. Examples include archived news and publications.
- Relatively Static. These objects may change, but changes are infrequent and irregular, based on no measurable event. A common example would be a user's web page. One method to determine when such changes occur is to track local access events as users

editing the contents of an object may access the object to verify changes [8]. Caching of relatively static objects can be done, but the lifetime in which the cached object is fresh is uncertain. Validation based on the age of the object is often used to balance performance and staleness concerns.

In this work, we use various techniques to try and identify the extent to which objects with these different change characteristics are used. We consider relatively static to be the default category for objects so we do not concentrate on those objects and for the period of our study it is difficult to determine which are truly static. Instead we focus on studying the content that is periodically generated, dynamic or access dependent, dependency-based and input dependent.

2.3 Exploiting Object Relationships

We are also interested in studying how to exploit the relationships between objects for their management. Objects have different types of relationships. The set of objects contained within a composite object form one type of relationship. The set of objects derived from a particular database comprise another type of relationship. The objects most likely to be accessed after the current object form yet another type of relationship—these could be determined from client access patterns and could possibly include analysis of traversal links between objects.

These relationships are important because they can be combined with object change characteristics to yield a more deterministic approach to manage objects at a site for caching. This knowledge can be "compiled" by a server to generate information that the server can pass to client caches. Such information might include explicit expiration times for caching the object or to exploit the relationships between objects to define volumes for invalidation at a cache.

We motivate this idea by using a specific example for a set of objects of different types, change characteristics and relationships. The example, shown in Figure 1, is a composite object modeling a current news portal Web page. The composite object contains a variety of objects both in terms of their type and change characteristics.

The composite object (c1) in Figure 1 is relatively dynamic in terms of its change characteristics because the layout and contents of the object change frequently. If these changes were known to remain constant for some minimum time granularity then the object could be classified as periodic with a short expiration time. The composite object contains six embedded objects—an ad banner (o1) that rotates to a new ad on each access; a lead news article (o2) and a topical photo object (o3) that do not change themselves, but may be replaced within the composite object; a tracking object (o4) for current stock or sports scores, which is periodically updated, but changes to static at the end of the business day; a logo image (o5), which is unlikely to change; and a policy statement for the site (o6), which is also unlikely to change.

The variation in change characteristics for the set of objects can be exploited in their management. Client caches must validate the composite object on each access because it is relatively dynamic. However, when clients retrieve or validate the contents of the composite object, the server can use the validator as a version indicator to piggyback any invalidations

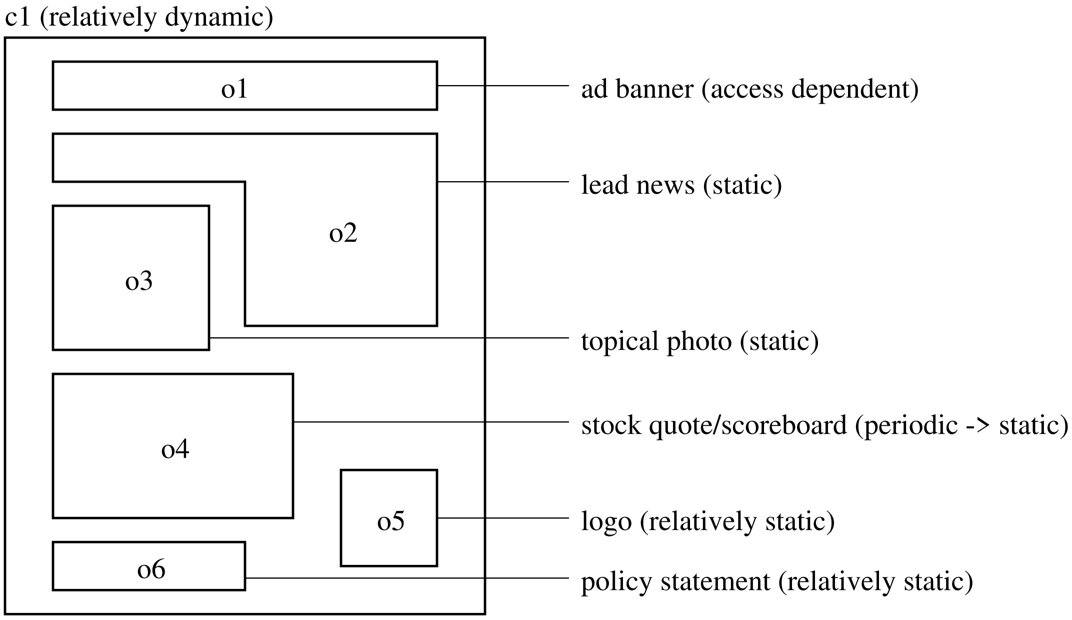

Figure 1: Current News Composite Object

of relatively static objects for the volume of objects associated with the composite object—in this case objects o5 and o6. The server could also pipeline new objects needed to render all objects referenced by the composite object or just let the client request them as it parses the composite object contents.

The use of piggybacked invalidations allows clients to avoid heuristic validation of relatively static objects. Such cached objects are maintained as coherent using piggybacked server invalidations [7]. The result is a reduction in the number of unneeded validation checks for these objects and makes caching more deterministic. Objects such as o5 may be contained in other server volumes so that the retrieval of any one of a number of objects could invalidate cached copies of o5 if it is changed at the server.

Changes in the contents of a composite object can also be exploited in managing objects. If the lead news and photo objects (o2 and o3) are replaced in the composite object by other objects then it may no longer be worthwhile for a proxy cache to keep copies of these objects. More specifically, the server needs to maintain a set of "pointers" to an object from one or more composite objects. If none of the composite objects reference the object then the server should piggyback this information to client caches. The client cache should mark these objects as ready for removal if garbage collection is needed. Likewise, proxy caches could maintain these pointers between just the set of objects it caches and mark unreferenced objects as ready for removal. While these objects could potentially be reclaimed in the future, they are obvious candidates for replacement.

This idea of exploiting relationships between objects of different change characteristics, leads to a to examining the potential reduction of unnecessary validation checks generated by current heuristic cache consistency techniques. Specifically we investigate if validation checks for image objects to a server are made within a small time window of full object retrievals from the same server. If so, these validation checks could be eliminated if the client cache knew the server to would piggyback invalidations if any changes did occur.

3 Methodology

In our previous work we identified and studied popular Web sites [16] and frequently requested URLs [14]. The methodology of this study is to use both types, with one test set being the home pages for popular sites and another test set for popular URLs from a current NLANR proxy log [11]. The methodology of the study is to perform an unconditional HTTP GET for each of the URLs in a test set on a daily basis. For each retrieved resource, we store response headers and calculate an MD5 checksum on the contents. Contents of HTML and text resources are stored if changed from the previous retrieval. Once an HTML resource is retrieved, it is parsed and all embedded images and traversal links are recorded. Image objects are retrieved and their header information is stored, but we do not try to track changes to image objects in this work. Previous work has shown that images rarely change [5, 16].

Not only did we focus on popular URLs and home pages from popular sites, we also focused on two specific types of popular sites—search engines and electronic commerce sites that provide home shopping. For these sites we both studied the home page and the results of two sample queries we created for each site.

We identified a subset of the URLs from our other test sets that returned a cookie. For these URLs, we did further investigation of the dependence of the retrieved resource on the cookie value. The approach is to initially retrieve each URL twice, recording the cookie returned each time. On subsequent iterations, two retrievals are made, one with the first cookie sent and one with the second cookie sent. Analysis of the two resources returned allows us to better understand if and how varying the cookie changes the response.

Finally, we developed a tool for automated analysis of changes to an HTML object. The tool decomposes an HTML object into smaller "chunks" (hypothesized objects) based on the inherent structure of the object. For example, the `<table>` tag is commonly used to define structure for an HTML object. An MD5 checksum is calculated for each chunk and is used to determine the number of chunks and number of bytes common between two separate retrievals. While this approach does not capture the structure of these objects as well as could be done by the designer of these objects, it does use the page structure to provide an approximation of how an HTML designer might decompose it into smaller objects.

4 Results

This section provides information about the test sets used in our study and examines the effectiveness of techniques described in Section 2.

4.1 Test Sets

Six data sets were used in this study. One of them, Cnt300 was derived from seven NLANR proxy logs [11]. Logs were collected at the end of October, 1999 from 7 proxy servers spread across the country. Aggregate number of entries in all 7 logs was over 8.5 million. Over 7.4 million of those were accesses that resulted in 200 or 304 HTTP response codes. All entries without a valid content type and image URLs were filtered out. This study focused on HTML objects, so images are only of interest if they are embedded in relevant HTML pages. The resulting data set contained over 866,000 distinct URLs. All entries with fewer than 10 accesses were eliminated as well as query URLs. Such queries could not be used because all parameters after the question mark were sanitized in the trace data making replication of such requests impossible. The resulting set contained almost 9,000 URLs. Considering that the study would retrieve HTML objects and all embedded images having the initial data set of 9,000 is quite unmanageable (images account for over 90% in all of our final data sets). It was further pruned to eliminate URLs with fewer than 300 accesses. The final set had 128 URLs.

Two separate data sets were obtained from 100hot.com [12] on November 2, 1999. The first, Top50, contains the 50 most popular Web sites. Some sites are represented by more than one host name, so this data set has 71 URLs. The second is Ecom, which contains the home pages of the 50 largest shopping sites (business-to-consumer). In addition, we defined the Srcheng test set, which contains home pages of eleven well-known search engines.

Two other data sets were created to test query results. The EcomQ test set represents two queries submitted to each of the top ten shopping sites from Ecom data set. Notice that some of these are not precisely queries but rather links pointing to various products. But all URLs do have "?" followed by query parameters. The SrchengQ test set consists of two queries submitted to each of the search engines in Srcheng set.

For each data set base HTML objects and all

embedded images were retrieved every day at the same time for 11 days. Images from a few Ecom sites were not retrieved due to use of "https" for the protocol within the URL. All response headers were saved and catalogued for later examination. HTML content was also saved for all retrievals. Images were discarded after their size was determined and saved locally. Doing a HEAD request on images instead of a GET was not an option since Content-Length was not always available (also in a number of cases a HEAD request behaved exactly like a GET—resulted in a full body response). An MD5 checksum was computed and saved for each retrieved object.

Table 1 gives caching related summary statistics for each data set. Notice that the total number of resources is much larger than the number of base URLs. This is due to the fact that many base HTML objects had a number of embedded images—almost 25 images per HTML on average. Also some base HTML pages were constructed using frames, which resulted in fetching all HTML resources necessary to render that page.

4.2 Content Reuse

The first step of our analysis examined the amount of reuse for the Cnt300, Top50, Srcheng and Ecom test sets. Our approach is similar to what we previously used [16], except we used our chunking tool to examine the potential reuse for of HTML object if it was decomposed into distinct objects. This approach allows static portions of each HTML object to be identified and considered reusable between successive retrievals.

Table 2 shows the results of our study for the popular Web pages contained in four test sets. The second column shows the average number of bytes for the base HTML object in each test set. The first value in parentheses is the percentage of those chunk bytes that could

Table 1: Summary Information for Test Sets

Item	Test Set					
	Cnt300	Top50	srcheng	ecom	srchengQ	ecomQ
Number of Base URLs	128	71	11	50	22	20
Number of Resources	1747	2362	180	2071	352	536
ETag	811 (46.4%)	1200 (50.8%)	105 (58.3%)	1547 (74.7%)	214 (60.8%)	408 (76.1%)
Expires	314 (18.0%)	155 (6.6%)	13 (7.2%)	144 (7.0%)	11 (3.1%)	49 (9.1%)
Last-Modified Time	1538 (88.0%)	2147 (90.9%)	157 (87.2%)	1976 (95.4%)	290 (82.4%)	490 (91.4%)
Pragma/ Cache-Control	200 (11.4%)	256 (10.8%)	5 (2.8%)	181 (8.7%)	8 (2.3%)	69 (12.9%)
Set-Cookie	224 (12.8%)	282 (11.9%)	14 (7.8%)	595 (28.7%)	54 (15.3%)	103 (19.2%)
Content-Type: html/text	157 (9.0%)	153 (6.5%)	11 (6.1%)	104 (5.0%)	22 (6.2%)	42 (7.8%)
Content-Type: image	1590 (91.0%)	2209 (93.5%)	169 (93.9%)	1967 (95.0%)	330 (93.8%)	494 (92.2%)

have been reused from the previous retrieval of the object. The second value in parentheses is the percentage of object bytes exhibiting no change from the previous retrieval. The results for all test sets show a relatively high amount of reuse for our approach compared to low amounts of reuse when only reusing unchanged objects. This high rate of change for the HTML objects is consistent consistent with previous results [5, 16]. Our results show that if these frequently changing objects were decomposed into their static and dynamic portions then a much higher degree of reuse is possible.

Table 2: Content Reuse for Popular Web Pages

Test Set	Base Object Bytes (% Reuse, No Change)	Embedded Objects (% Reuse)	Embedded Objects Bytes (% Reuse)	Total Objects Bytes (% Reuse)	% Base Object Bytes Saved From diff -e
cnt300	11495.1 (77%, 23%)	5.6 (85%)	14023.8 (70%)	25519.0 (73%)	84%
top50	17276.7 (75%, 16%)	12.1 (86%)	23921.0 (75%)	41197.6 (75%)	82%
srcheng	14977.8 (75%, 6%)	8.4 (89%)	10686.5 (79%)	25664.3 (77%)	81%
ecom	16826.4 (70%, 16%)	16.2 (92%)	33061.0 (83%)	49887.4 (79%)	76%

The third and fourth columns in Table 2 show the average numbers of embedded objects and object bytes for each test set. The percentages in parentheses show the amount of reuse from the previous retrieval of the page. Based on previous results that images rarely change [5, 16], we assume that content of embedded images does not change and the images can be reused between successive retrievals of the Web page. The number of objects show a high rate of reuse with a bit less for the object bytes.

The total number of bytes (HTML and embedded objects) along with the percentage of reuse is indicate in column 5. The results show that approximately 75% of the bytes needed for these popular Web pages could be reused from the previous retrieval of the page. Note that in determining these figures we measure only changes to the object content and ignore any cache control directives returned by the server with the object content. Results from our previous studies indicate that overally restrictive or missing cache directives limit the potential reuse of current cached content [14, 16].

The last column of Table 2 shows the percentage of base object bytes saved if the difference is computed between the base object and previous retrieval of the base object. This difference is done using the Unix command *diff* with option *-e* to generate input to the *ed* text editor. This was not the best differencing tool tested in [9], but it is widely available. As shown in the table, the *diff -e* output is a bit more efficient in representing the content reuse than our chunking tool. These better results can occur because even chunks that change may have large portions of content that do not change. However, the key difference between identifying chunks that change versus calculating deltas is that only the new contents of chunks need to be communicated to the client versus delta-encoding where the server must calculate the difference between the client version and the new version of the page. These differences can either be costly in terms of computation costs if done in real-time or costly in terms of storage if maintained for all pairwise version differences.

4.3 Exploiting Object Relationships to Eliminate Unneeded Validation Requests

The next part of our work examined the potential of eliminating unneeded validation requests (resulting in 304 responses) from a client cache request stream. If servers exploited relationships between objects contained within the same Web page then invalidations for any relatively static object could be piggybacked on to responses for other objects from that same page. This protocol would allow the client to reuse these objects, such as images and relatively static text, identified by the server without depending on a heuristic approach.

Our approach to study the potential of this technique is to not use data from our test sets, but rather use request data from current logs. Either proxy or server logs could be used for this study, but since we obtained the NLANR logs for determining popular URLs, we used these same logs for analysis of this technique.

We focused on all objects with 200 and 304 server responses that was served by an origin server. For each 304 response from a server, we marked that it could be eliminated if a 200 response from the same server came within a 10-second window on either side of the 304 response. This heuristic is intended to identify unneeded validation checks as any invalidations that could be piggybacked by the server on a needed 200 response. Because we think this technique will have primary benefit for elimination of validations for image objects we further classified the 304 responses as images based on variations of the strings "gif," "jpg" and "image" in the URL name.

Results for each of the seven NLANR proxy logs we obtained are shown in Table 3. Given that NLANR caches generally serve as second-level caches for other caches, the percentage of 304 responses is still relatively high. The table shows that 15-32% of all requests result in a 304 response, which is consistent with previously published results.

Table 3: Occurence and Potential Elimination of Validation Checks

NLANR Proxy	Object 200/304 Response Total Cnt.	Object 304 Response Cnt. (% Total)	Image 304 Response Cnt. (% Total)	Object 304 Response in Window Cnt. (% Total)	Image 304 Response in Window Cnt. (% Total)
bo1	541230	155762 (29%)	140591 (26%)	120650 (22%)	109532 (20%)
bo2	797307	185935 (23%)	167992 (21%)	148340 (19%)	134897 (17%)
lj	413911	78283 (19%)	68999 (17%)	69870 (17%)	62077 (15%)
pa	418206	62377 (15%)	55188 (13%)	53682 (13%)	47979 (11%)
pb	505556	161539 (32%)	140653 (28%)	128410 (25%)	112003 (22%)
sd	228852	72909 (32%)	65296 (29%)	54448 (24%)	49077 (21%)
sv	872419	172385 (20%)	153216 (18%)	151441 (17%)	135262 (16%)

More importantly for our study, is the large percentage of these 304 responses that contain a related 200 response in their window (virtually no variation was found for larger window sizes) and the large percentage of these 304 responses that are images. The results show that 11-22% of all object requests could be eliminated by removing image validations that are in the same window with a 200 response request to the same server. 13-25% of requests could potentially be eliminated when considering all object types. We believe these results are significant. They show this technique can be used to eliminate the majority of validation requests currently handled by servers due to inefficient cache consistency mechanisms. The result of this reduction is a performance improvement for clients, caches, servers and the network.

4.4 Object Change Characteristics

As illustrated in the example shown in Figure 1, the objects on a Web page can exhibit a number of different change characteristics, which lead to different approaches for managing these objects in our proposed approach. In the following, we study the amount and implications of different types of objects. For this work we assume the chunks we identify are treated as embedded objects within a Web page.

4.4.1 Periodic Objects

The HTTP protocol already has a mechanism for servers to identify objects that change at periodic intervals or at least have a fixed lifetime. The expiration header can be used for this purpose. As shown in Table 1, however, relatively few objects currently are delivered with an expiration time. Also our previous study indicates that these expiration times may be misleading: expiration times in the past or short for objects that do not change.

4.4.2 Dynamic and Access Dependent Objects

We characterize objects as dynamic if they change each time they are retrieved regardless of the time period between retrievals. We characterize objects as access dependent if they change each time they are retrieved, but the change is access and not time dependent.

To detect the presense of these types of objects contained in Web pages we analyzed the Top50 test set using a modified strategy. Each URL in the test set was retrieved twice: the second retrieved immediately after the first. Approximately fifteen minutes later a third copy was retrieved. The first row of Table 4 shows content reuse results from comparing the first and the second retrievals. The second row of the table shows results of comparing the latter two retrievals.

The results show that only 30% of the Web pages remained the same between successive retrievals. This indicates that most of the pages contain dynamic or access-dependent content. Identification of this content is important as 91% of the previous page could be subsequently reused if the dynamic content separated out. The set of embedded objects also changes on each request. The results for the 15 minute interval show just a bit less reuse is possible. These relative results indicate that most of the short-term content changes occur immediately.

Table 4: Content Reuse for Access Dependent and Dynamic Pages

Test Set	Base Object Bytes (% Reuse, No Change)	Embedded Objects (% Reuse)	Embedded Objects Bytes (% Reuse)	Total Objects Bytes (% Reuse)	% Base Object Bytes Saved From diff -e
top50-imm	17747.5 (91%, 30%)	13.0 (94%)	29976.9 (86%)	47724.5 (88%)	94%
top50-15m	17716.5 (89%, 28%)	11.6 (93%)	24749.0 (85%)	42465.4 (87%)	93%

4.4.3 Dependency-Based Objects

We characterize objects as dependency-based if they change based on a measureable event at a server such as a file or database changing. For these types of objects we used the SrchengQ and EcomQ test sets to measure the changes for the same search engine query or shopping site product inquiry over our testing period. The results for these tests sets are shown in Table 5.

Table 5: Content Reuse for Dependency-Based Web Pages

Test Set	Base Object Bytes (% Reuse, No Change)	Embedded Objects (% Reuse)	Embedded Objects Bytes (% Reuse)	Total Objects Bytes (% Reuse)	% Base Object Bytes Saved From diff -e
srchengQ	25025.5 (56%, 0%)	7.4 (83%)	17788.6 (52%)	42814.1 (55%)	65%
ecomQ	16292.4 (59%, 13%)	10.7 (95%)	17549.8 (85%)	33842.2 (73%)	72%

The results show that the base object changes on each retrieval for the set of search engines and almost each time for the ecommerce sites. However, the majority of the content on each of these objects (56% and 59%) remains the same between retrievals. These results indicate some amount of dependency-based changes are occurring between each retrieval.

4.4.4 Input Dependent Objects

We characterize an object as input dependent if an input parameter is included as part of the request for the object. In the Web, two such examples are the use of query parameters and cookies. To study the impact of different parameters we used the data from the SrchengQ and EcomQ data sets. In this case we did not compare the results between successive retrievals for the same query, but rather we compared the results from two separate queries to determine the potential reuse between two queries at close points in time.

The results from this study of separate query pairs are shown in the first two rows of Table 6. As expected, the results show that contents always changes based on the query parameter, but there is some amount of reuse possible, particularly for the search engine results. These results are consistent with a preliminary study where we found 20-30% of content reusable between different queries submitted to the same search engine [15].

Table 6: Content Reuse for Parameter- and Cookie-Dependent Pages

Test Set	Base Object Bytes (% Reuse, No Change)	Embedded Objects (% Reuse)	Embedded Objects Bytes (% Reuse)	Total Objects Bytes (% Reuse)	% Base Object Bytes Saved From diff -e
srchengQ-p	25446.9 (22%, 0%)	7.3 (76%)	17356.3 (39%)	42803.2 (29%)	31%
ecomQ-p	29856.0 (6%, 0%)	11.4 (29%)	21179.5 (14%)	51035.5 (9%)	9%
cnt300-c	7524.7 (81%, 0%)	2.9 (79%)	6812.5 (47%)	14337.1 (65%)	95%
top50-c	19614.8 (83%, 41%)	11.7 (95%)	24743.1 (86%)	44357.8 (84%)	89%
ecom-c	25966.1 (71%, 8%)	20.9 (97%)	42367.4 (92%)	68333.5 (84%)	69%

We used the three largest test sets, Cnt300, Top50 and Ecom, to study the impact of cookies on the returned content. The study was done on all HTML objects in these test sets that returned cookies as part of our regular study. The result was 13, 12 and 16 HTML objects for the three respective test sets. Two cookie-less requests were generated for each object to obtain two different cookies. Each cookie was then sent along with an object request and the two responses were compared. Results of these comparisons are shown in the last three rows of Table 6. We first note that the number of objects exhibiting no change is less than 50%. This result is lower than what we found in previous studies [14, 16], although we do not know what percentage of time the change was dependent on the cookie. The results also show a high amount of potential reuse despite the frequency of changes.

5 Related Work

Previous work has proposed the use of server invalidations for invalidating cached content where each server propagates resource changes to all clients that accessed a resource since its previous modification [8]. While this approach guarantees strong cache coherency, it requires the server to maintain a client list for each resource, which could become out-of-date as clients may no longer have previously accessed resources in their cache. In the event of network failures, even the guarantee of strong cache coherency could be lost. The number of invalidations can be reduced by using the concept of "leases" where a server grants a lease whenever a client uses a validation request indicating a subsequent access for the resource [17]. This approach reduces the number of invalidation messages generated by a server at the expense of more validation requests generated by clients. Recent work has proposed extensions for maintaining consistency in a hierarchy of caches [18, 19], but does not specifically exploit relationships between objects within a Web page or at a server site.

Some work has been done to construct volumes based on directory structure and client access patterns [4], but we believe that better volumes can be constructed by explicitly identifying the object relationships at server sites.

Work by Kroeger, et al explored the bounds of web latency reduction from caching and prefetching. Their work found much potential improvement from "local prefetching" where a client cache receives multiple requests for an object that changes between requests so it could be potentially prefetched by the client. These results are encouraging for our work because they indicate that if a client cache could reuse pieces of these changing objects rather than prefetching them then overall performance could be improved.

Douglis, et al [5] did early work on determining the frequency at which resources change. Many of these researchers were also responsible for a study on delta encoding [9], which allows servers to transmit changes to a resource as a list of differences. Delta encoding is an alternate to using embedded objects in a page. It is more general, but in cases where portions of a page change frequently, but predictably the embedded objects can receive updates without the server having to explicitly compute a difference between the new and old versions. Cao, et al [2] have proposed a more general method for cachelets to handle small changes to cached objects, but the results show this approach incurs non-trivial computational costs at a proxy cache. Douglis, et al [6] propose an idea for separating static and dynamic portions of a page, which is similar to our idea of substitution tags.

In terms of using dependencies between objects for their management, previous work for updating result web pages during the Nagano 1998 Winter Olympics shows some similarities [3]. This work involved the development of a Data Update Propagation protocol for automatically updating the contents of mirrored servers when score data was changed. Other work has demonstrated better results for caching of dynamic content when geographical parameters in queries lead to the identification of equivalent or partially equivalent requests [13].

6 Summary and Future Work

The results of our study show that the techniques we propose have real potential for improving the amount of content reuse for many Web pages and reducing the amount of traffic between client caches and servers. Results from four test sets of popular Web pages show the amount of content reuse can increase by over 50% in comparison to reusing an HTML object only if no change occurs. More importantly, content providers can realize this savings simply by decomposing objects based on homogeneous change characteristics rather than more expensive techniques such as delta encoding.

The results also show that exploiting the relationships for objects contained within the same Web page can eliminate the majority of validation requests currently handled by servers due to inefficient cache consistency mechanisms. Servers can send invalidations for objects on the page that are unlikely to change so that clients do not need to waste network and server resources, as well as increase user response time, to validate these objects.

Finally, examination of different object change characteristics shows that many of these popular Web pages change on each retrieval, but the majority of the content can be reused. Input dependent objects show much less shared content between independent queries, but some reuse is still possible, particularly for search engine queries. Results for objects with different cookies show much potential for reuse.

In the future we plan to extend our study on elimination of unnecessary validations to other proxy and server logs. We would alsolike to study the usefulness of content reuse from not just the immediately past retrieval. We are also working to extend the implementation of current server and cache software to measure its specific performance impact. We believe the performance improvements from implementation of these techniques will lead to more deterministic, rather than heuristic, management of Web objects—a key advancement on existing approaches.

7 Acknowledgements

This work would be impossible but for the logs provided by the National Laboratory for Applied Network Research supported by National Science Foundation grants NCR-9616602 and NCR-9521745. Thank you.

References

[1] Martin Arlitt and Tai Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-1999-35, HP Laboratories Palo Alto, February 1999.

[2] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents (objects) on the web. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, The Lake District, England, September 1998.

[3] Jim Challenger, Arun Iyengar, and Paul Dantzig. A scalable system for consistently caching dynamic web data. In *Proceedings of the IEEE Infocom '99 Conference*, New York, NY, March 1999. IEEE.

[4] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM '98 Conference*, Vancouver, British Columbia Canada, September 1998. ACM.

[5] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: a live study of the world wide web. In *Symposium on Internet Technology and Systems*. USENIX Association, December 1997.

[6] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *USENIX Symposium on Internet Technology and Systems*, Monterey, California, USA, December 1997. USENIX Association.

[7] Balachander Krishnamurthy and Craig E. Wills. Piggyback server invalidation for proxy cache coherency. In *Seventh International World Wide Web Conference*, pages 185–193, Brisbane, Australia, April 1998.

[8] Chengjie Liu and Pei Cao. Maintaining strong cache consistency in the world-wide web. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, May 1997.

[9] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *ACM SIGCOMM'97 Conference*, September 1997.

[10] Eric Nahum. WWW workload characterization work at IBM research. In *Web Characterization Workshop*, Cambridge, MA, November 1998. World Wide Web Consortium.

[11] NLANR. Proxy cache log traces, October 1999. `ftp://ircache.nlanr.net/Traces/`.

[12] 100hot.com.

[13] Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu. Exploiting result equivalence in caching dynamic web content. In *USENIX Symposium on Internet Technology and Systems*, Boulder, Colorado, USA, October 1999. USENIX Association.

[14] Craig E. Wills and Mikhail Mikhailov. Examining the cacheability of user-requested web resources. In *Proceedings of the 4th International Web Caching Workshop*, San Diego, CA, March/April 1999.

[15] Craig E. Wills and Mikhail Mikhailov. Exploiting object relationships for web caching. Technical Report WPI-CS-TR-99-29, Computer Science Department, Worcester Polytechnic Institute, October 1999. `http://www.cs.wpi.edu/~cew/papers/tr99-29.ps.gz`.

[16] Craig E. Wills and Mikhail Mikhailov. Towards a better understanding of web resources and server responses for improved caching. In *Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.

[17] Jian Yin, Lorenzo Alvisi, Michael Dahlin, and Calvin Lin. Using leases to support server-driven consistency in large-scale systems. In *Proceedings of the 18th International Conference on Distributed Systems*. IEEE, May 1998.

[18] Jian Yin, Lorenzo Alvisi, Mike Dahlin, and Calvin Lin. Hierarchical cache consistency in a WAN. In *USENIX Symposium on Internet Technology and Systems*, Boulder, Colorado, USA, October 1999. USENIX Association.

[19] Haobo Yu, Lee Breslau, and Scott Shenker. A scalable web cache consistency architecture. In *Proceedings of the ACM SIGCOMM '99 Conference*, Cambridge, Massachusetts USA, September 1999. ACM.