Studying the Impact of More Complete Server Information on Web Caching

Craig E. Wills and Mikhail Mikhailov*

Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
{cew,mikhail}@cs.wpi.edu

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 result in a large number of unnecessary validations, and prior results show potential for better reuse of cached Web content. This work studies a more deterministic approach to caching of Web objects. The idea is to view HTML pages as containers, holding distinct objects with heterogeneous type and change characteristics. Servers compile information about relationships between containers and embedded objects and piggyback it onto existing request/response traffic. Our results indicate that these techniques significantly 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 [13, 15] and focuses on investigating the possibilities of increasing the reuse of cached content and reducing the number of cache validation requests (GET If-Modified-Since requests).

Caches currently rely on the "Last-Modified-Time" (LModTime) HTTP response header, if an expiration time is not available, to heuristically assign a Time-To-Live (TTL) value to an object. Caches are not allowed to serve stale objects to clients without consulting the origin server first. Studies showed that 15-18% [8], 30% [10] and 37% [1] of client requests

*Supported by ArrowPoint Communications.

resulted in responses with 304 code (cached copy is up-to-date). 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].

Content creation is often a complicated process. As designers compose pages from objects with different semantics, content types and change characteristics, boundaries between objects disappear, effectively producing a single monolithic object (HTML page). Often a small portion of such an object changes frequently, even on each access, making the use of heuristic TTLs impossible. Caches treat such objects as uncacheable. Relationships between objects composing a page represent valuable information which can be used to improve object management. Currently servers do not provide this information to caches.

Motivated by problems with heuristic approaches to caching and our prior work on the potential of dynamic content reuse [13, 14, 15], we propose an alternative approach to caching of Web objects. The idea is to take advantage of the fact that objects have different type and change characteristics and are not isolated from each other. Content types are already taken into account, but the information about object change characteristics and relationships between objects is lost. Composite objects should be constructed using a non-monolithic approach. They should be viewed as *container* objects: holders for other objects. Relationships between containers and embedded objects can be exploited at the server side to generate cache invalidation information. Invalidations might include explicit expiration times for a list of objects or a list of members of a server volume, and can be piggybacked onto existing request/response traffic [8]. A *volume* is a set of objects grouped by some relation. It might contain all objects required to construct a given composite object or all objects with a common dependency, such as on an underlying database.

We focus on two techniques—composition of objects and exploitation of their relationships—and show how they can be integrated to make caching more deterministic. We use two approaches to evaluate the effectiveness of these techniques. One is to actively monitor a set of URLs at a variety of sites and gather information about their content changes. The other is to use traces of actual client requests to estimate the reduction in the number of unnecessary validation checks. We also examine the change characteristics of objects that constitute current monolithic objects. We conclude with a summary of our work to date and ideas for future research.

2 Techniques for Study

2.1 Composition of Objects

The notion of composite objects is already present on the Web. An HTML document often serves as a *container* page with images, applets, audio and video clips inserted and retrieved as part of the page. This approach, however, is too simplistic because it only recognizes the need for making a distinction between objects based on type. Other object properties, such as change characteristics, are not taken into account. As observed in [9, 7, 14], large pieces of an HTML page often remain the same while smaller pieces change. Current Web caching techniques treat the entire page as a monolithic object—if any piece of it changes then the entire object must be re-retrieved. One part of our work is to study the amount

of content reuse that could occur if larger objects were properly composed from objects of homogeneous type and change characteristics. Two possible composition techniques are:

- *Embedded objects.* The most straightforward approach, already proposed in the HTML 4.0 specification, is to extend the notion of embedded images to generic objects. The `<object>` tag allows an arbitrary object to be included into an HTML page, but this tag is not supported by most browsers. The `<ilayer>` and `<iframe>` tags, introduced by Netscape and Microsoft respectively, are meant to perform a similar function, but are less general and powerful than the `<object>` tag.

- *Substitution tags.* This approach uses a special tag directive to mark locations within the composite object where other components should be inserted. Changes to inserted objects can be identified by the server simply as tag substitutions. This approach preserves the structure of the composite object and takes advantage of it. The idea is similar to the HTML pre-processor, proposed by Douglis, et. al., where the static portion of a page (template) contains macro-instructions for inserting dynamic information [7]. We envision, however, that tag substitution approach would not require a separate pre-processor.

Web pages are currently not composed in this way. Since we do not have control over or access to internal architectures of Web sites, our approach is to *decompose* existing HTML pages into a set of hypothesized objects and then examine potential improvements in content reuse if the composition of these objects was used instead of the original monolithic page.

2.2 Exploiting Object Relationships

Currently all Web objects are considered to be independent and are treated as such: one retrieval or validation request is used per object. In reality objects are not isolated from each other, they are involved in different types of relationships. For example, a set of objects contained within a composite object form one type of relationship; objects derived from a particular database comprise another type of relationship; objects most likely to be accessed after the current object form yet another type of relationship. These relationships are important because they can be combined with object change characteristics to yield a more deterministic approach to object management. Servers can *compile* description of relationships between objects, object expiration times, or volumes for invalidation and pass that information to client caches.

To show how this technique works, we provide a specific example in Figure 1 of a composite object and a set of embedded objects with various change characteristics. The example is modeled after a Web news portal. The composite object c1 is relatively dynamic because its layout and contents change frequently. c1 contains six embedded objects: o1 rotates to a new ad on each access; o2 and o3 are the top news story and photo that do not change themselves, but may be replaced within the composite object; o4 is a stock quote that is updated periodically until becoming static at the end of the business day; o5 and o6 are unlikely to change. Variation in object change characteristics forces the use of different cache control mechanisms. Some objects could be assigned an expiration time, others cannot be cached at all and will have to be retrieved on every access. We can exploit the fact, however,

that objects o1-o6 are united by their containment relationship since they compose object c1. Using separate retrievals and validations for each object is wasteful in this case. Client caches must validate the composite object c1 on each access because it is relatively dynamic. When they do so, the server can piggyback invalidations of relatively static objects o5 and o6 from the volume of objects associated with the composite object c1. 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 has changed at the server.

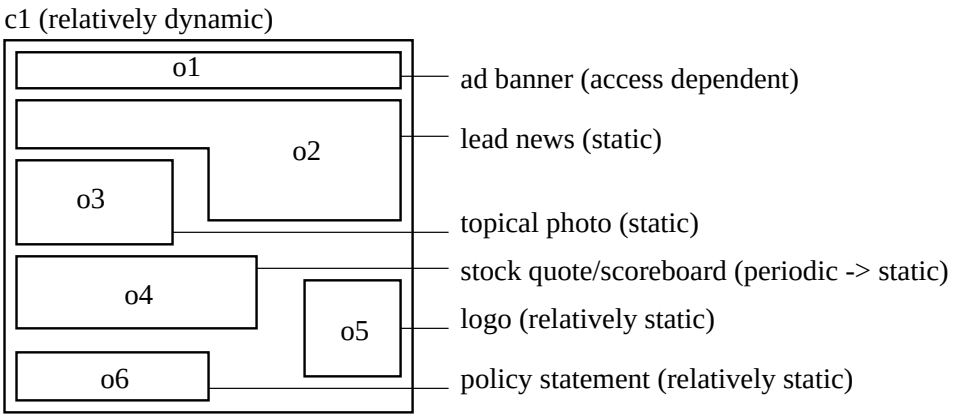

Figure 1: Current News Composite Object

The use of piggybacked server invalidations [8] allows clients to avoid heuristic validations of relatively static objects and keep them fresh. It also results in the reduction in the number of unneeded validation checks and more deterministic caching.

Changes in the contents of composite objects can also be exploited. If objects o2 and o3 are replaced by other objects then it may no longer be worthwhile for a proxy cache to keep copies of these objects. Servers need to maintain a set of pointers to an object from one or more composite objects, and notify client caches when objects become unreferenced. Client caches should mark such objects as ready for removal. Proxy caches could also maintain pointers between just the set of objects they cache and mark unreferenced objects as ready for removal. While these objects could be reclaimed in the future, they are obvious candidates for replacement.

This idea of exploiting relationships between objects with heterogeneous change characteristics leads 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 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 would piggyback invalidations if any changes did occur.

3 Methodology

In previous work we studied URLs for the popular Web sites [15] and frequently requested URLs [13]. In this study we constructed our test sets from both types of URLs but included HTML URLs only. Being primarily interested in investigating potential for dynamic content

reuse, we specifically examined two types of popular sites—search engine sites and business-to-consumer electronic commerce sites. For these, we studied their home pages and two sample queries. We also identified a subset of the URLs from our test sets that returned a cookie. For these URLs, we investigated their dependency on the cookie value.

Our methodology is to perform an unconditional HTTP GET for each of the base URLs in a test set once per day. We wrote software to make such retrievals at the same time of day for 11 days, parse the obtained HTML objects and retrieve all embedded images. When our software detects the presence of frames, it fetches each frame along with the images embedded in it. In a negligibly small number of cases, images could not be retrieved due to use of "https" within the protocol portion of the URL.

An additional initialization step was required for the experiment with URLs which returned cookies. We first retrieved each URL twice and recorded the cookie returned each time. On subsequent iterations, using the methodology described above, two retrievals were made: one using first cookie and the other using second cookie. Analysis of the two returned resources allows us to better understand if and how varying the cookie changes the response.

For each retrieved resource, we stored the response headers and calculated an MD5 checksum on the contents. HTML resources were kept if the checksum differed from the previous retrieval. Images were discarded after their size was determined and saved locally. We keep track of images remaining on the page between successive retrievals and consider them static because previous work has shown that images rarely change [6, 15].

We developed a tool to automatically analyze changes to HTML objects. The tool decomposes HTML objects into smaller "chunks" (hypothesized objects) based on their inherent structure. For example, the `<table>` tag is commonly used to define structure. We then calculate an MD5 checksum for each chunk and use it to determine the number of chunks and number of bytes common between two retrievals.

4 Results

This section provides information about the test sets used in our study, experimental results and a discussion of the effectiveness of the 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], collected at the end of October, 1999 from proxy servers spread across the country. Aggregate number of entries in all logs was over 8.5 million. Over 7.4 million of those were accesses that resulted in HTTP response codes of 200 or 304. Our study focused on HTML objects and images embedded in them—we thus ignored image URLs in proxy traces. After all entries without a valid content type were filtered out, the test set contained 866,000 distinct URLs. Considering that our study would retrieve all embedded images, having such large initial data set is quite unmanageable (images account for over 90% in all of our final data sets). We further pruned the data set to eliminate entries with fewer than 300 accesses and query URLs. The NLANR caches sanitize all parameters after the

"?" in query URLs, making replication of such requests impossible. The resulting data set contained 128 URLs.

Three other 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). The third test set, Srcheng, lists home pages of eleven well-known search engines.

Two more 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 the Ecom data set. Some of these are not precisely queries but rather links pointing to various products. All URLs, however, do have a "?" followed by query parameters. The SrchengQ test set consists of two queries submitted to each of the search engines in the Srcheng set.

The total number of resources retrieved was 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. Some base HTML pages were constructed using frames, which resulted in fetching all HTML resources necessary to render those pages. More details about the test sets can be found in [14].

4.2 Content Reuse

The first experiment was designed to examine the potential of HTML content reuse if the proposed object composition technique was used. Using our chunking tool, we decomposed HTML objects from the Cnt300, Top50, Srcheng and Ecom test sets into distinct hypothesized objects. This approach allows static portions of each HTML object to be identified and considered reusable between successive retrievals.

Table 1 shows the results of this experiment. 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 bytes that could have been reused from the previous retrieval if objects were composed from our chunks. The second value in parentheses is the percentage of bytes that can be reused if we only considered cases when objects do not change between retrievals. The high rate of change for the HTML objects is consistent with previous results [6, 15]. Our approach of decomposing frequently changing objects into their static and dynamic portions offers a significantly higher amount of reuse across all test sets.

Table 1: Content Reuse for Popular Web Pages

Test Set	HTML Bytes (% Reuse, No Chg)	Images (% Reuse)	Image Bytes (% Reuse)	Total Bytes (% Reuse)	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 1 show the average number of embedded objects

and embedded object bytes respectively for each test set. The percentages in parentheses show the amount of reuse from the previous retrieval of the page. Based on the previous results that images rarely change (95%-100% the same) [6, 15], 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 shows 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 their reuse is indicated in column 5. 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 measured only changes to the object content and ignored any cache control directives returned by the server. Our previous studies indicate that overly restrictive or missing cache directives further limit the potential for content caching and reuse [13, 15].

The last column of Table 1 shows the percentage of base object bytes saved if the difference is computed between successively retrieved copies of the base object. The technique of transmitting a list of differences between cached and updated content to caches was studied in [9] and is called *delta encoding*. The differencing is performed using UNIX *diff* command with option *-e*—the form required for 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 slightly 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 that do not.

The key advantage of using the chunk (object) composition technique over delta encoding, however, is that only the changed chunks need to be communicated to clients. If an updated object is part of many pages at a site, our approach will utilize one retrieval, while delta encoding will require a separate retrieval for each changed page. Servers using delta-encoding must either have a pre-computed difference between the client version of the page and their own new copy or generate the difference upon request. This can be computationally expensive, if done in real-time, or could require large amounts of storage, if all pairwise version differences are kept. Delta encoding does not have to be orthogonal to our composition approach: clients could obtain object deltas instead of retrieving updated objects.

4.3 Exploiting Object Relationships

The next part of our work examined the potential for eliminating unneeded cache validation requests (resulting in 304 response code). If servers exploited relationships between objects contained within a Web page, they could piggyback invalidations for any relatively static object onto responses to requests for other objects from that same page. This protocol would allow clients to reuse objects not invalidated by the server, without depending on a heuristic approach.

We used data from the NLANR proxy logs [11] for this study, but server logs could also be used. We focused on objects with 200 and 304 response codes served by origin servers. Our assumption is that each 304 response could be eliminated if a 200 response came from the same server within a 10 second window on either side of the 304 response (approximating the same page). This technique will be most beneficial for elimination of validation requests

for images. Looking for variations of the strings "gif," "jpg" and "image" in the URLs, we further classified the set of 304 responses to requests for images.

Results for each of the seven NLANR proxy logs are shown in Table 2. The percentage of 304 responses is high, even though the NLANR caches generally serve as second-level caches. Column 3 shows that 15-32% of all requests result in a 304 response, which is consistent with previously published results [8, 10, 1].

Table 2: Occurrence and Potential Elimination of Validation Checks

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 important for our study is the large percentage of 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. Column 6 shows that 11-22% of all object requests could be eliminated by removing image validations that fall within a window of a 200 response from the same server. Even more requests, 13-25%, could be eliminated if all object types are considered. 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 technique is better than the more general piggybacking mechanism of [8] because it produces tighter volumes with fewer invalidations for objects not present in the cache. The result is a performance improvement for clients, caches, servers and the network.

4.4 Object Change Characteristics and Content Reuse

Web objects can exhibit a variety of change characteristics, as illustrated in Figure 1. This leads to the use of different approaches for managing these objects. We study the presence of heterogeneous objects in the current Web pages by identifying changes that occur in different versions of pages obtained with a particular strategy. We use our chunking tool to decompose HTML pages into chunks, which we treat as embedded objects.

We analyzed the Top50 test set to detect the presence of *dynamic* and *access dependent* objects in Web pages. These objects change on each access. The strategy was to perform two consecutive retrievals for each URL in the test set, and the third retrieval fifteen minutes later.

The first row of Table 3 shows the amount of content reuse between the first and the second retrievals. 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 because 91% of the previous page could be subsequently reused if dynamic content was separated out. The second row of the table shows the results of comparing the second and the third retrievals. The amount of content reuse is slightly smaller in this case. These results indicate that most of the short-term content changes occur immediately.

Table 3: Content Reuse for pages with Access Dependent, Dynamic, Dependency-Based and Input Dependent Objects

Test Set	HTML Bytes (% Reuse, No Chg)	Images (% Reuse)	Image Bytes (% Reuse)	Total Bytes (% Reuse)	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%
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%
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%

To detect the amount of change for *dependency-based* objects we examined a set of pages that are likely to depend on a database. We used the SrchengQ and the EcomQ test sets to measure the changes between successively retrieved pages for the same search engine query or product inquiry. These retrievals were made on a daily basis as described in Section 3. Rows 3 and 4 of Table 3 show that the base object changes on each retrieval for the set of search engine sites and almost as often for the electronic commerce sites. The results show, however, that the majority of content (56% and 59%) of pages that contain dependency-based objects can be reused. Further study is needed to analyze the constituency of objects contained on such pages.

Finally, we examined the amount of content reuse for pages containing *input dependent* objects. We characterize an object as input dependent if the request for that object includes an input parameter. Two examples of input parameters on the Web are search engine queries and cookies. We used data from the SrchengQ and the EcomQ data sets to study the impact of different input parameters.

In this experiment, we did not compare the results of successive retrievals of the *same* query. Instead, we compared the results of two *different* queries. Rows 5 and 6 of Table 3 show that page contents always changes based on the query parameter, as expected, but there is still some amount of reuse possible, particularly for the search engine data set. These results are consistent with our preliminary study [14], where we found that 20-30% of

content is reusable between different queries submitted to the same search engine.

In the second part of this experiment, we studied the impact of cookies on the retrieved content. We selected all the HTML objects with cookies from the three largest test sets: Cnt300, Top50 and Ecom, which resulted in three data sets with 13, 12 and 16 URLs respectively. This experiment required an initialization step, as described in Section 3.

Two retrievals were made for each URL, one using first cookie and the other using second cookie. Responses were compared. Results of these comparisons are shown in the last three rows of Table 3. The number of objects exhibiting no change is less than 50%. This is lower than what we found in previous studies [13, 15], although we do not know what fraction of the change is due to cookies. The results show that a high amount of reuse is possible if composition of objects with different change characteristics was used.

5 Related Work

Previous work [5] investigated the construction of volumes for server invalidations based on directory structure and client access patterns. We believe that better volumes can be constructed by explicitly identifying object relationships at server sites.

Douglis, et. al. [6] worked on determining the frequency at which resources change. The same researchers are also responsible for a study on delta encoding [9], which allows servers to transmit changes to a resource as a list of differences. 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 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. [7] proposed an idea of separating static and dynamic portions of a page, which is similar to our idea of substitution tags.

Work on updating Web pages of the Nagano 1998 Winter Olympics site [3, 4] shows some similarities to our ideas, in terms of using dependencies between objects for their management. That work involved the development of a Data Update Propagation mechanism to automatically update the contents of mirrored servers when underlying data changed.

6 Summary and Future Work

The results of this work show that our proposed techniques can significantly improve the performance of current caching mechanisms in a number of ways. First, our results show a 50% increase in the amount of content reuse in comparison to only reusing an HTML object if no changes occur. Furthermore, we believe this result is conservative as it is obtained based on decomposition of current Web pages rather than measuring reuse for the set of objects composed by the designer of the composite object.

Second, the results show that exploiting the relationships between objects contained within the same Web page can eliminate the majority of validation requests. Many of these requests are currently due to inefficient cache consistency mechanisms and have to be handled by servers. If caches could depend on servers to piggyback invalidations in the case of changes then most of these heuristically-generated validations could be eliminated.

Third, our results show that objects with different types of change characteristics can be detected within current monolithic pages. A key question is whether it is possible and feasible to classify the often large number of objects at a server site into these categories. Our viewpoint is that many objects are already generated automatically based on measurable events or at regular intervals. It is a trivial addition for these automated tasks to mark the change characteristic of the resulting objects. In addition, the type of an object may define its change characteristic. For example, a non-generated image may be marked as static. The final aspect of the problem, which we believe makes this approach possible, is that only the most popular objects and composite objects require special attention for classification. By default, all objects can be marked as relatively static, with other techniques exploiting the relationships between objects.

We are currently working to understand how to combine object change characteristics with object relationships to determine a strategy for managing the set of objects served by a server. We are also working to extend current server and cache software to support our proposed techniques and measure their impact on performance. We believe our techniques will lead to more deterministic, rather than heuristic, management of Web objects—a key improvement on existing approaches.

7 Acknowledgements

We thank the National Laboratory for Applied Network Research for making proxy logs available. Data collection at the NLANR is supported by National Science Foundation grants NCR-9616602 and NCR-9521745.

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] Pei Cao, Jin Zhang, and Kevin 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.

[4] Jim Challenger, Arun Iyengar, Karen Witting, Cameron Ferstat, and Paul Reed. A publishing system for efficiently creating dynamic Web content. In *Proceedings of the IEEE Infocom 2000 Conference*, Tel-Aviv, Israel, March 2000.

[5] 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.

[6] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

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

[8] 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.

[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*, Cannes, France, 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. `http://www.100hot.com`.

[13] 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.

[14] 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`.

[15] 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.

Vitae

Craig E. Wills is an associate professor in the Computer Science Department at Worcester Polytechnic Institute. His research interests include distributed computing, operating systems, networking and user interfaces.

Mikhail Mikhailov is a Ph.D. student in the Computer Science Department at Worcester Polytechnic Institute. His research interests include distributed computing, operating systems, networking and computer system performance evaluation.