WPI-CS-TR-99-29 October 1999

Exploiting Object Relationships for 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 latencies for clients accessing these objects while reducing network and server load. Motivated by results that show much potential for better reuse of Web content, this work describes 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 its own type and change characteristic. Relationships between these objects are "compiled" at a server into information that can be piggybacked by the server onto existing request/response traffic. Specific examples are given to show how current caching techniques can be extended to improve caching effectiveness.

1 Introduction

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. Increasingly, exported Web objects 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 at a server site would be taken into account for caching, however these relationships are often not exploited. Instead caches use heuristic-based approaches for deciding what to cache, how long to keep it and when cached objects may be stale.

Motivated by the results of our study examining the potential reuse of Web content, 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 objects are "compiled" at a server into information that the server can pass to clients. 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.

We examine this approach by first considering the factors that influence the current state of web caching. We go on to present the results of a study of Web objects to examine the amount of cached content that could be reused. We discuss how three techniques—composition of objects, classification of their change characteristics and exploitation of their relationships—can be integrated to make caching more deterministic. Specific examples of how current caching techniques can be extended to improve performance are provided. We conclude with a summary of our work to date.

2 Web Caching Techniques

Although individual Web browsers often cache retrieved Web objects, the study of Web caching is primarily focused on proxy caches. A proxy cache acts as an intermediary between a large number of clients and remote web servers by forwarding requests from clients to various servers. In the process, the proxy caches frequently requested objects to avoid contacting the server repeatedly for the same object if it knows, or heuristically decides, that the content in the object has not changed at the server.

Proxy caches must make decisions about what objects to cache and what objects to replace when the cache becomes full. These decisions are based on user access patterns using an approximation of the Least Recently Used (LRU) policy. A number of alternatives to the LRU policy have been proposed in the literature [1], but are not widely used.

Caches must also make decisions on whether a cached object is up-to-date relative to the copy at the origin server. In some cases a server knows how long an object will stay current and reports this information to clients in the form of an HTTP expiration time header when the object is served. In other cases the origin server indicates an object should not be cached using other HTTP directives. More commonly, a retrieved object has no clear expiration time. Table 1 shows the use of these directives by current Web servers determined by a study of frequently referenced URLs [9]. As shown, an explicit no-cache or expiration directive occurs infrequently for both HTML objects and images. In the remaining cases, the last modified time (lmodtime) is almost always supplied for images, but most of the time it is not available for HTML objects.

Table 1: Use of HTTP Cache Directives (%)

Object Type	No Cache	Expires	LModTime	None
HTML	10.43	6.59	25.82	57.15
Image	0.31	3.16	92.91	3.62

In the cases where the server does not supply no-cache or expiration time directives, caches use one of two approaches. In the first approach, a proxy cache always validates the contents of the cached object with the origin

server before serving it to the client. This approach provides strong cache consistency, but can lead to many 304 responses (HTTP response code for "Not Modified") by the server if the object does not actually change. In the second approach, the proxy cache assigns a time-to-live (TTL) to each cached object. If the lmodtime is not available then the cache assigns a fixed TTL or does not cache the object at all. If the lmodtime is available then the cache heuristically assigns an adaptive TTL to each cached object based on the age of the object [4]. The older the object, the longer the time period between validations. This adaptive TTL heuristic reduces the number of validations, but potentially leads to stale objects served from the cache. It is important to reduce unnecessary validations. Our own studies showed that 15-18% of client requests resulted in 304 responses [5]. Data from IBM's corporate Web server indicates that 30% of the server responses return a 304 status [8].

Server invalidation is one other approach for maintaining cache consistency. It has not been employed. This approach depends on servers to propagate object changes to all its clients that accessed an object since its previous modification [6]. While this approach guarantees strong cache coherency, it requires the server to maintain a client list for each object, which could become out-of-date as clients may no longer have previously accessed objects in their caches. The number of invalidations can be reduced by using the concept of "leases." When a server grants a lease on an object it agrees to sends updates to the client cache for a specific period of time, but once the lease expires the client must validate the object before reuse [11]. This approach reduces the number of invalidation messages generated by a server at the expense of more validation requests generated by clients.

3 Study of Object Reuse

The first part of our work was to study the amount of Web content that can be reused by a cache. The initial part of this study examined the rate of change for frequently retrieved Web objects as done by Douglis, et al [2]. We performed two studies, one where we retrieved objects from popular web sites [10], and two we retrieved frequently referenced objects from a current Web proxy log [9]. In both cases the objects were retrieved on a daily basis.

Our first step in analyzing the data was to repeat the rate of change calculations as done by Douglis, et al [2]. The images for all test sets show

virtually no change as found in [2] while the HTML objects show much variation in change characteristics. The .net, .org and .edu domain site results showed 60-70% of the HTML objects did not change (comparable to the HTML results in [2]), but the HTML objects for the commercial sets showed much more volatility. Only 10-20% of these objects did not change during the study while 70-80% of these objects changed on each retrieval. 100% of the query HTML objects changed on each retrieval. As a comparison, 40-50% of HTML objects never changed in the subsequent study or frequently referenced objects [9].

Next we examined HTML objects from the standpoint they are often "containers" for embedded images. We looked at the frequency at which embedded images remain in an HTML objects between successive retrievals. The results show that the percentage of images remaining is a little over half for the commercial sites tested (a similar percentage was found for frequently requested URLs in [9]). The results for the .net, .org and .edu test sites show that for 70-80% of objects the set of images remains the same between retrievals. A query test set, consisting of a test query string sent to a number of search engines, yielded the least amount of reuse on average, although the median is still close to 50%.

We also examined the extent of changes to HTML objects. If the changes to an object are relatively small then techniques such as delta-encoding [7] or techniques to separate larger objects into smaller objects become relevant. In our initial work we manually examined a selected set of popular HTML objects that change frequently. For each object in the case study we first retrieved the object, then retrieved it again to find changes that occur over a short period of time. We then retrieved the object again after a longer period of time (a few hours to a day) to discover long term changes. We found that in many cases the only short-term changes were to the ads contained on the page. Longer term changes included new news stories, new dates and some changes in the set of images. However, the structure of the page remained largely the same.

Subsequent to this work we have developed a tool for automated analysis of changes to a Web HTML object. The tool decomposes an HTML object into smaller "chunks" based on the inherent structure of the object. For example, the `<table>` tag is commonly used to define structure for an HTML object. The methodology we use with the tool is to periodically retrieve copies of a Web object, such as the home page of popular sites, and

decompose each retrieval into chunks using the tool. We then calculate the MD5 checksum of each chunk and use the results 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 provide an estimate for the amount of content reuse that does occur.

To illustrate the use of this methodology, Figure 1 shows the results for changes to a number of commercial Web sites over a 24-hour period during midweek in September, 1999. The results for each data point are the average amount of reuse for each interval for each site. The sites shown are representative of results we obtained from a larger group of sites tested.

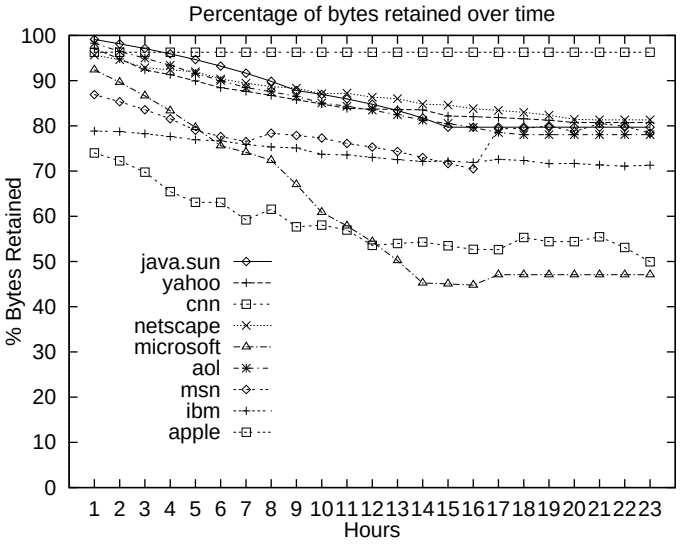

Figure 1: Percentage of Web Home Pages Retained Over Time

The results show that for all sites there is a relatively high amount of reuse over the course of a day, even though some content changes on each access. For example, the amount of reuse was over 90% for the Netscape home page after one hour with the page remaining 80% the same after one day. The amount of reuse on the CNN home page fell from approximately 80% after an hour to about 50% after a day. In a related study we measured the amount of reuse between results obtained for different search strings at different search engine sites. Even though the search results were completely different, the

resulting page still showed 20-30% content reuse for the Altavista, Excite and Hotbot search engines.

These collective results indicate that although the pages of these popular sites exhibit a high rate of change, there is still a significant amount of reuse in the content and set of embedded images. These results motivate the need to create objects to not only include content of homogeneous type, but also of homogeneous change characteristics. This approach leads to better reuse of object contents.

4 Composition of Objects

The results of this study indicate a clear need to consider the composition of Web objects in their management. The need to divide larger objects into structured components is a well-known technique. Examples include dividing source code into file modules or constructing a book as a sequence of chapters. The capability to synthesize different types of objects is part of MPEG-4, a new multimedia standard. The notion of composite objects is already present in Web pages 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 shown in Figure 1, pieces of a larger HTML object often remain the same while smaller pieces of the object actually change. However, as current Web caching techniques are implemented, the larger object is treated as a whole—if any piece of it changes then the entire object must be retrieved.

There are a number of alternatives to this approach that consider the change characteristics of individual pieces in constructing the composite object.

1. Embedded objects. The most straightforward approach is to extend the notion of embedded images to more generalized 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 ob-

ject 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 [7]. 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 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 [3]. However, we envision that just 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. This model is used for subsequent discussion and examples, but delta-encoding and tag substitution also offer potential performance improvements in communicating object changes.

5 Object Change Characteristics

The study identified the portions of Web pages that changed over time, but did not explicitly identify the causes of these changes. Subsequent work has examined different Web objects for classification based on change characteristics. These object change categories, their descriptions and implications for their management and caching 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 any 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 [10], 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 [6]. 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.

Having defined these categories, 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. Specifying the dependencies for the generation of an object is a similar problem to maintaining dependencies for software modules as is done with tools such as Make.

In addition, the type of an object may define its change characteristic. For example, a non-generated image may be marked as static. However, what really 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, as discussed in the next section, to provide more deterministic management of these objects in comparison to existing heuristic-based approaches.

6 Exploiting Object Relationships

The final implication of our study is the need 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. In the following, we elaborate on this approach with specific examples of objects of different types, change characteristics and relationships.

6.1 Current News Object Example

The first example, shown in Figure 2, is a composite object modeling a current news portal site. The composite object contains a variety of objects both in terms of their type and change characteristics.

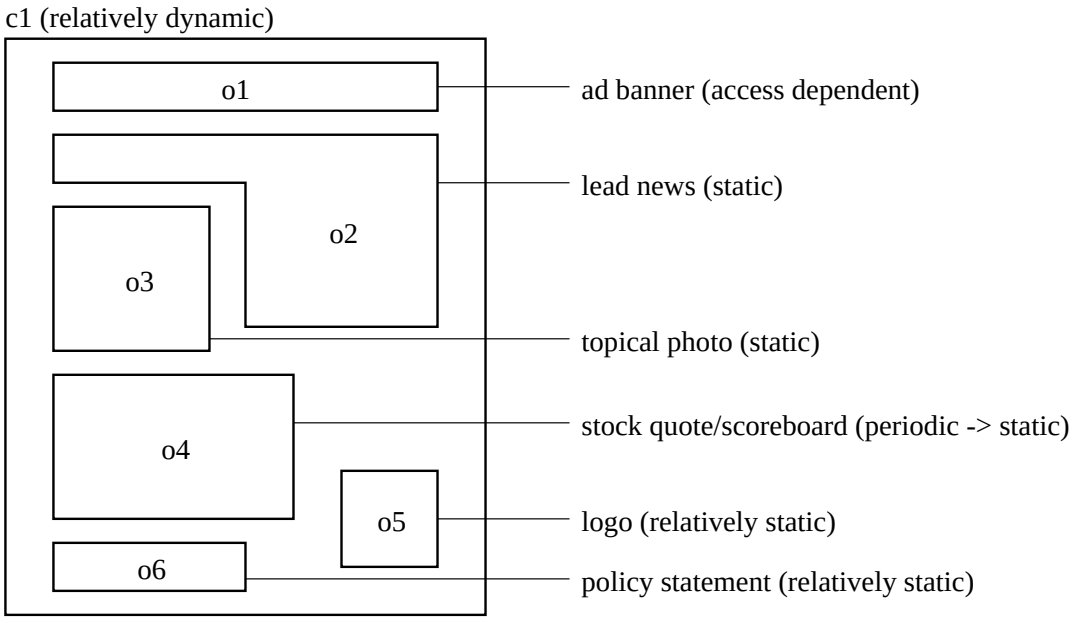

Figure 2: Current News Composite Object

The composite object in Figure 2 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 in 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 perform two important functions:

1. pipeline new objects needed to render all objects referenced by the composite object, and

2. piggyback any invalidations of relatively static objects for the volume of objects associated with the composite object—in this case objects o5 and o6.

The first function allows all needed objects to be retrieved with a single request over a persistent connection with the server. This approach assumes that preexisting objects have been cached and that only objects that are new to the composite object since the given validator are retrieved. The second function allows clients to avoid heuristic validation of relatively static objects. Such cached objects are maintained as coherent using piggybacked server invalidations [5]. 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.

The identification of the access dependent object o1 in Figure 2 presents another opportunity for performance improvement. Rather than use a traditional time-based expirations for cached objects, access dependent objects can be cached with a count-based approach to allow such objects to expire after a pre-determined number of accesses. In the simplest case, the count could be one, which still requires the client to retrieve a new ad on each access of the page. However, the advantage of such a count-based expiration is that it can be combined with client-based prefetching where a client cache can prefetch such objects immediately after they expire. The newly retrieved and cached objects then have an expiration count of one. The next access of the same object is serviced by the client cache, which itself can trigger a new copy of the object to be retrieved from the server for the next access.

6.1.1 Database Dependency Example

Another type of relationship exists between objects when they are all derived from a database. When the database changes, these derived objects may potentially change. An example of this type of dependency is shown in Figure 3 where four objects contained within three composite objects are each dependent on a database.

The approach for managing these dependency-based objects is to group them as part of a volume and include this volume information when one of these objects is retrieved. When the dependency (in this case the database) is modified then all objects within this volume are potentially changed. This invalidation for the volume identifier is piggybacked to client caches on subsequent responses from the server. This invalidation indicates that any volume object cached by a client is potentially invalid and needs to be validated before reuse.

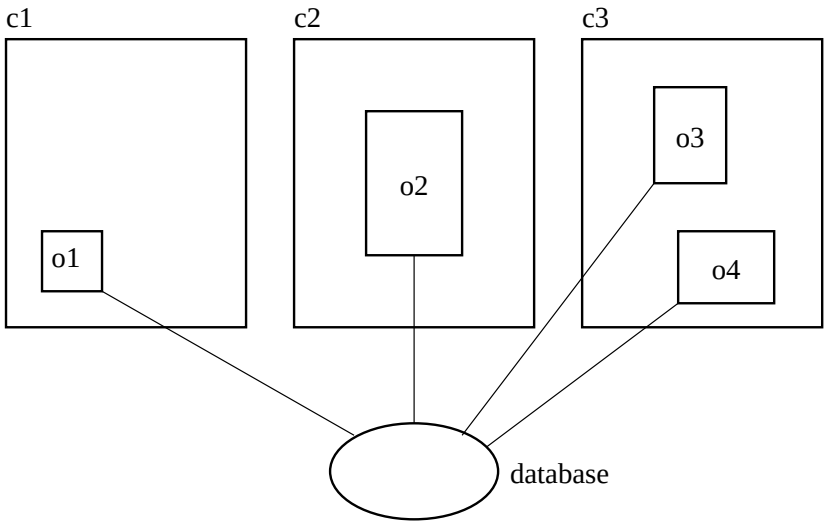

Figure 3: Database Dependency Example

6.2 Input Dependent Object Example

The last example we show is one where a composite object contains an input dependent object. This example is shown in Figure 4 where the content of the composite object c1 is expected to remain the same while the object o3 is input dependent. The other objects contain content that is "wrapped" around o3. If o3 is dependent on input parameters, such as keywords for a search query then the object is likely to change for each set of keywords. However, if the input is a user identifier, such as a Web cookie, then as we found in previous work [10], the object contents often stay the same despite different input parameters. In the latter case, the object can be cached, but needs to be validated before reuse. In addition, this validation can be used to invalidate any relatively static object. For example if a client cache validates object o3 in Figure 4 then the server can use this request to send any invalidations for objects c1, o1, o2, o4 and o5.

7 Summary

The results presented in this work show there is much potential for better reuse of Web content. These results motivate specific techniques for Web

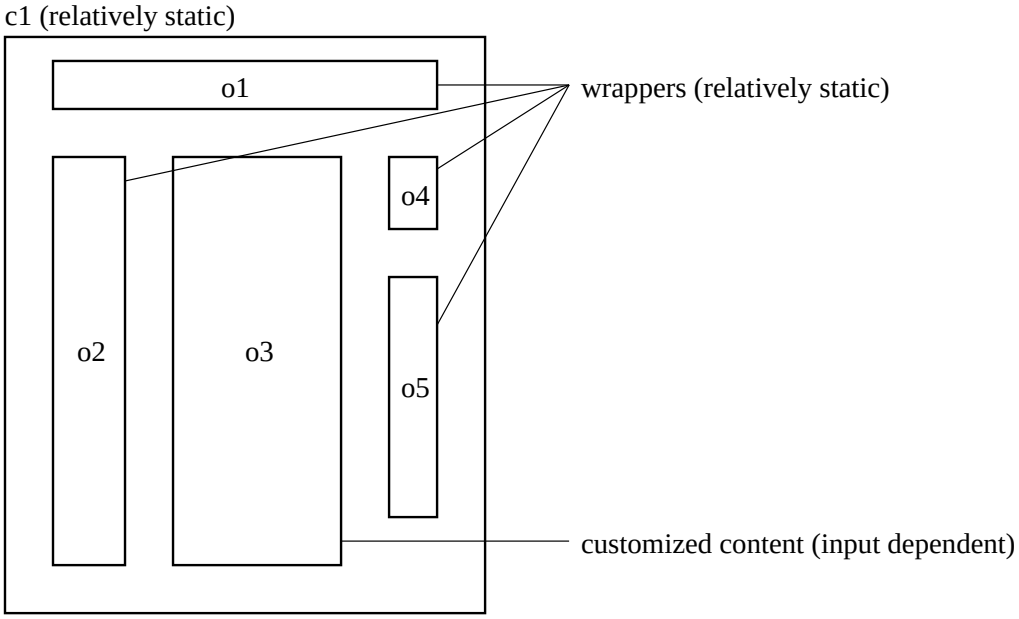

Figure 4: Input Dependent Object Example

caches to improve the effective performance of the Web using more complete information provided by servers. Exploitation of object relationships allows proxy caches to perform garbage collection on objects that are no longer embedded as part of server composite objects. The use of piggybacked invalidations allows proxy caches to learn of relatively static objects that do change as part of responses to requests for related objects from a server. This approach reduces the need for clients to make unnecessary validation requests to servers for cached objects. Finally the composition of objects based on homogeneous change characteristics allows unchanged portions of composite objects to be cached and reused.

We are currently working to study the performance improvements provided by these techniques using data from current Web servers and caches. 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 its implementation will lead to more deterministic, rather than heuristic, management of Web objects—a key advancement on existing approaches.

References

[1] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Symposium on Internet Technology and Systems*. USENIX Association, December 1997.

[2] 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.

[3] 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.

[4] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *Proceedings of the USENIX Technical Conference*, pages 141–152. USENIX Association, January 1996.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.