Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content[*]

Mikhail Mikhailov
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
mikhail@cs.wpi.edu

Craig E. Wills
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
cew@cs.wpi.edu

ABSTRACT

The problem of Web cache consistency continues to be an important one. Current Web caches use heuristic-based policies for determining the freshness of cached objects, often forcing content providers to unnecessarily mark their content as uncacheable simply to retain control over it. Server-driven invalidation has been proposed as a mechanism for providing strong cache consistency for Web objects, but it requires servers to maintain per-client state even for infrequently changing objects. We propose an alternative approach to strong cache consistency, called MONARCH, which does not require servers to maintain per-client state. In this work we focus on a new approach for evaluation of MONARCH in comparison with current practice and other cache consistency policies. This approach uses snapshots of content collected from real Web sites as input to a simulator. Results of the evaluation show MONARCH generates little more request traffic than an optimal cache coherency policy.

Keywords

Web Caching, Cache Consistency, Change Characteristics, Object Relationships, Object Composition, Collected Content, Server Invalidation.

1. INTRODUCTION

The problem of Web cache consistency continues to be an important one. Current Web caches use heuristic-based policies for determining the freshness of cached objects, often forcing content providers to unnecessarily mark their content as uncacheable simply to retain control over it. These heuristic policies also generate unnecessary validation requests for content that is marked as cacheable. Server-driven invalidation has been proposed as a mechanism for providing strong cache consistency for Web objects, but it requires servers to maintain per-client state even for infrequently changing objects [6, 10, 11, 12].

We propose an alternative approach to maintaining strong cache consistency for Web objects, called MONARCH (Management of Objects in a Network using Assembly, Relationships and Change cHaracteristics). An initial version of this approach was presented in [7]. MONARCH exploits the fact that Web pages are often constructed from a set of objects with a mix of change characteristics. Instead of treating all objects independently, as is done in other approaches, MONARCH identifies useful relationships among objects composing a page and uses these relationships to keep all objects strongly consistent.

In [7] we evaluated the performance of an initial version of MONARCH that worked for Web pages with no shared objects. Our evaluation in [7] used simulations with synthetic content and compared MONARCH with a limited set of other policies. It did not allow us to evaluate MONARCH and other policies on content from real Web sites and did not allow for comparison with current practice.

The de facto evaluation standard in the Web research community is to perform trace-driven simulations using Web or proxy server logs. It is well-known that such logs are hard to find and they tend to be from smaller or research-oriented sites. They also tend to be dated. It is difficult to obtain (recent) traces from large and popular commercial Web sites. Yet, evaluation of new proposals on precisely these types of sites is of most interest.

In this work we propose a new evaluation methodology that can be applied to the content from any Web site of interest. The idea is to collect a snapshot of content for the site by using the home page as a starting point and following a sample of links available on that page. This sample includes both links that persist on the page over long periods of time and links that come and go over shorter periods of time. By taking a snapshot at frequent intervals we can capture the dynamics of the site. We then use the collected data as an input to a simulator and evaluate a wide range of cache consistency policies over a range of access patterns. Downloading content from a site also allows us to obtain HTTP directives related to caching as used by the site. The availability of these directives makes it possible to compare policies with current practice for cache consistency.

In evaluating cache consistency policies using our simulator, we examine the amount of request traffic between a client cache and a server as well as the number of stale objects served from the cache. We also examine the overhead, both at the server and in message traffic, incurred by a policy in the process of maintain strong cache consistency. We use results from another study [5] to better understand how the relative performance differences between policies are expected to translate into the differences in the end user response time.

The rest of the paper is organized as follows. In Section 2 we describe our approach to strong cache consistency on the Web. In Section 3 we describe the set of Web sites that served as the source of real content for this study. In the same section we also present our methodology for content collection and converting it into the format used by our simulator. In Section 4 we discuss the simulation methodology that we used and describe cache consistency policies studied in this work. In Section 5 we present the results of

[*]This work is partially supported by the National Science Foundation Grant CCR-9988250.

our study. We then discuss related work in Section 6 and conclude the paper.

2. THE MONARCH APPROACH TO STRONG CACHE CONSISTENCY

MONARCH is an approach to strong Web cache consistency motivated by two observations. We first observe that the majority of Web pages consist of multiple objects and the retrieval of all objects is required for proper page rendering. Current approaches to caching on the Web treat all such objects independently from each other. We believe ignoring relationships between objects is a lost opportunity. We also observe that objects composing Web pages change at different and identifiable rates. The container object may be changing every few minutes while embedded objects remain unmodified for months. The essence of our approach to strong cache consistency is to examine objects composing a Web page, select the most frequently changing object on that page and have the cache request or validate that object on every access. Such requests also carry enough information for the server to determine related objects the client is likely to have cached and to determine if invalidations for these related objects need to be sent as part of the reply.

2.1 Object Change Characteristics

We classify objects into four categories, shown in Figure 1, based on how frequently objects change and whether their changes are predictable. The three categories on the left side of the figure represent predictably changing objects, for which the server has a priori knowledge at what time or upon which event the change will occur. We define the three predictable categories as follows. The **Static (St)** category includes objects that never change and can be cached for as long as necessary. The **Periodic (Per)** category includes objects that the server updates at predetermined intervals. Periodic objects can be assigned explicit expiration times. The **Born-On-Access (BoA)** category covers objects that the server can only generate at the time of the client's access. The **Non-Deterministic (ND)** category on the right side of the figure, subdivided into **Relatively Dynamic (RDyn)** and **Relatively Static (RSt)** subcategories, represents unpredictably changing objects.
The **RDyn** and **RSt** subcategories divide **ND** objects into those that are relatively more and relatively less likely to change between accesses. As we will show, the distinction is useful, but not critical, for the MONARCH approach.

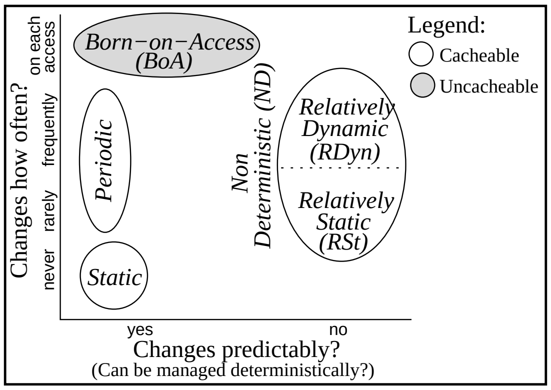


Figure 1: Classification of Object Change Characteristics

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. Servers already automatically generate many objects based on measurable events or at regular intervals. To mark the resulting objects with appropriate change characteristics is a trivial addition to these automated tasks. The type of an object or its location within a file system may define its change characteristic. For example, a manually created image, such as a digital photograph, may be marked as static. A set of objects that are known to change frequently, but unpredictably, may be grouped together and marked as RDyn. If the change characteristics of an object are not known then it should be marked as RSt by default. Another observation, that we believe makes this approach feasible, is that MONARCH need only be applied to the most popular pages at a site.

2.2 Combining Object Relationships with Object Change Characteristics

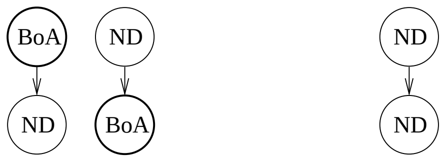
Once objects are classified based on their change characteristics, the server examines relationships between objects at the site. In [7] we only considered relationships between objects on the same page. In this work we have extended MONARCH to also consider cases when objects are shared across pages. The server performs the following tasks: 1) it groups all ND objects that are embedded on more than one page into a single *global volume*; 2) it constructs per-page *local volumes* by grouping page-specific ND objects; 3) it determines the most frequently changing object—the *manager*—within each page; 4) it assigns each object on the page a *Content Control Command (CCC)* providing explicit instructions to caches on how to handle that object.

The CCC commands that MONARCH assigns to St and Per objects and to BoA objects that have no related ND objects are shown in Table 1. MONARCH uses CCC commands for objects with these change characteristics for uniformity reasons. These objects could be managed using mechanisms currently available in the HTTP protocol [2]: Per objects can be assigned explicit expiration time via the `Expires` or the "`Cache-Control: max-age`" header, and St objects can also be assigned an expiration time that is far into the future; BoA objects with no related ND objects could be marked as uncacheable using the
"`Cache-Control: no-cache`" header.

Table 1: CCC Commands Used by MONARCH for St and Per Objects and BoA Objects with no Related ND Objects (bold font highlights text used to identify commands in our implementation)

Object Change Characteristic	CCC Command
St	**C**ache
Per	**C**ache, **V**alidate after TTL or Expires
BoA	**N**ot **C**ache

The goal of MONARCH is to provide strong cache consistency for non-deterministically changing objects because other types of objects can already be managed with strong cache consistency, as shown in Table 1. Thus, the most useful relationships are between a frequently changing object, such as a BoA object, and an ND (RSt or RDyn) object. MONARCH ignores the relationships that St and Per objects have with other objects, since these relationships do not help manage ND objects. Two possible relationships between a BoA and an ND object are shown in Figure 2a. The first example in Figure 2a shows a BoA container object with an embedded ND object. The other example in Figure 2a shows an ND container object with an embedded BoA object. When a BoA object is not available on the page, MONARCH selects one ND object to manage the other ND objects on the page, as discussed below. The relationship between two ND objects is shown in Figure 2b. The three combinations shown in Figure 2 depict important combinations of the composition relationship and object change characteristics that are useful for deterministic object management. Adding more objects to each of the three combinations, irrespective of the change characteristics of the newly added objects, does not fundamentally affect the behavior of the relationship. We call each of the three combinations a *pattern* (a *page pattern* in the context of a Web page).



a. BoA-ND and ND-BoA Pattern (BoA Object is the Manager)   b. ND-ND Pattern (One ND Object is the Manager)

Figure 2: Useful Page Patterns

A given page pattern determines which object MONARCH selects as the manager. If a pattern is BoA-ND, then the BoA container object is the manager, as shown in Figure 2a. If a pattern is ND-BoA, then one of the BoA children of the ND container object is the manager. Finally, if a pattern is ND-ND the distinction between RSt and RDyn objects becomes important. If the ND-ND pattern can be represented as a RDyn-RSt pattern, then the RDyn container object is the manager. If the ND-ND pattern can be represented as a RSt-RDyn pattern, then one of the child RDyn objects is the manager. If, however, the ND-ND pattern is represented by either RSt-RSt or RDyn-RDyn pattern, then the container object is selected as the manager.

The CCC commands that MONARCH assigns to the manager and managed objects are listed in Table 2. All of these commands stem directly from the three page patterns. If the container object is BoA with ND embedded objects (BoA-ND pattern) the server instructs caches to keep meta information (global and local volume versions) for the container and cache embedded objects until the server explicitly invalidates them. If the container is ND with at least one BoA embedded object (ND-BoA pattern), the server instructs the cache to cache the container, and to satisfy a precondition—the retrieval of the BoA object—before re-using the cached copy of the container. If both container and embedded objects are ND (ND-ND pattern), the server instructs the cache to cache all objects, but validate a RDyn object (expected to change more frequently) on each access. If all objects are RSt or RDyn then the container page is selected for validation. In all three cases, upon subsequent requests for the page, the cache contacts the server and presents it with the version of the cached global and local volumes. The server satisfies the request and piggybacks [4] invalidations for those global and local objects that have changed since the last request from the cache. The CCC command assigned to the manager object in the ND-ND pattern is the same as the one assigned to Per objects in Table 1, except it does not explicitly specify a Time-To-Live or expiration time. Lack of such explicit value indicates validation on every access by default. The CCC command assigned to the managed ND objects is actually the same as the one assigned to St objects in Table 1, except invalidation can never occur for St objects. When a page contains no ND objects, including cases when the page is described by the BoA-BoA pattern, all objects on the page are assigned CCC commands shown in Table 1.

Table 2: CCC Commands Used by MONARCH for the Manager and Managed Objects (bold font highlights text used to identify commands in our implementation)

Page	CCC for the	
Pattern	Manager Object	Managed ND Objects
BoA-ND	**C**ache **M**eta info	**C**ache until invalidated
ND-BoA	**N**ot **C**ache	**C**ache with **pre**condition
ND-ND	**C**ache, **V**alidate	**C**ache until invalidated

2.3 Prototype Implementation

We have designed and implemented a MONARCH prototype system on top of Apache Web server. The system is comprised of three components: MONARCH Content Management System, which handles volume management, analysis of object change characteristics and relationships, manager selection and CCC assignment; MONARCH Web Server, responsible for serving content to clients and performing volume invalidation; and MONARCH Proxy Server, responsible for caching individual page components and assembling pages before serving them to clients. MONARCH requires no changes to the HTTP protocol. Volume identifiers and object and volume versions are implemented as extra HTTP headers. The CCC commands, the ones shown in Tables 1 and 2 and those that carry a list of invalidated objects, are also implemented as extra HTTP response headers. The syntax of our CCC commands, shown in Figure 3, is described by the augmented Backus-Naur Form discussed in RFC 2616 and used for the HTTP/1.1 protocol [2]. Abbreviations identifying each CCC command use the text in Tables 1 and 2 shown in bold font.

```
CCC = ‘‘cmd=’’ cmd-C | cmd-NC | cmd-CM |
               cmd-CV | cmd-INV
cmd-C = ‘‘C’’ [‘‘; pre=’’ token]
cmd-NC = ‘‘NC’’
cmd-CM = ‘‘CM’’
cmd-CV = ‘‘CV’’ [‘‘;’’ ‘‘ttl | expires’’ ‘‘=’’ 1*DIGIT]
cmd-INV = ‘‘INV; objs=’’ <‘‘> invalidation-list <’’>
invalidation-list =  invalidated-object
                     [‘‘;’’ *invalidated-object]
invalidated-object = token ‘‘ˆ’’ 1*DIGIT
```

Figure 3: Grammar for the CCC Commands

3. COLLECTION METHODOLOGY

Having defined the MONARCH approach, an important issue is to evaluate its performance relative to current and proposed cache coherency policies. In an ideal world we would be able to obtain current information about the content dynamics and access patterns at busy Web sites. We could then use this information in a trace-driven simulation to evaluate our policy against others. However, obtaining such information from a single site is difficult, and obtaining it from a variety of sites is not realistic.

In this section we describe an alternate to this ideal, a methodology that we developed and used for collecting content from Web sites. This content is converted into a format appropriate for a simulator we use to evaluate MONARCH against current and proposed coherency policies. The methodology must address a number of important issues: 1) what is the set of Web sites from which to collect content; 2) what content is collected from each site; and 3) how frequently is it collected. In this section we address all three issues. In the following section we discuss how accesses to this content are generated.

3.1 Source Web Sites

The number of existing Web sites is large and is growing continuously. Evaluation of a newly proposed approach that improves some aspect of the Web cannot possibly be carried out on the content of the entire Web. However, a relatively small number of recognizable Web sites are responsible for much of Web traffic. Thus to evaluate the usefulness of a new proposal it is only necessary to investigate whether it offers improvements for a sampling of such sites. We also argue that sites with semantically different types of content—news site vs. educational site vs. corporate site—may use different page construction mechanisms and have different content update patterns. It is thus important to ensure that the sites selected for a study cover a range of content characteristics. Given these site selection guidelines, our approach was to pick *recognizable* Web sites that offer semantically different types of content. Table 3 lists the eleven Web sites that we selected for this study. The Web sites in our set vary widely in the number of embedded objects that their pages have, in the frequency of updates and in the use of the HTTP directives related to caching. More information about the dynamics of these sites is provided in subsequent sections.

Table 3: Web Sites Used in Study

Web Site	Type of Site
amazon.com	large e-commerce site
boston.com	international/national/local news
cisco.com	corporate site
cnn.com	international/national news site
espn.com	sports scores/news
ora.com	corporate/publishing site
photo.net	graphics heavy discussion site
slashdot.org	discussion site
usenix.org	technical/scientific association
wpi.edu	educational site
yahoo.com	all inclusive portal

3.2 Content to Collect

Having identified a set of sites to study, we needed to decide on the set of objects to study at each site. While we could perform an exhaustive study of a site, we did not want to turn the study into a denial of service attack. In addition, only a small fraction of all objects at a site are responsible for the majority of client requests. Therefore, we focused on collecting the dynamics for a subset of content at a site that is likely to be requested by clients. Providing strong cache consistency for such content would be a significant improvement.

We used the home page for a site as the starting point for our collection. While it is possible to access a specific page within a site directly, by finding the link using a search engine or receiving a pointer via e-mail, many users "enter" a site and search engines navigate from the home page. Home pages of popular sites are also likely to change frequently as sites add more information, add pointers to new resources, or simply rotate existing content to create the feeling of frequent updates so users return often.

We also wanted to collect a sample of content that could be accessed via the home page. Rather than follow all links on the page or a random subset of links, we took a two-pronged approach. We first identified links on the home page that are always present. We label these links *static*. These links represent aspects of the site that are constant features such as the world news for a news site or admissions information for an academic site. Some users may frequently visit the site because they monitor this aspect of the site.

We also identified links on the page that change over time. We label these links *transient*. These links are of interest to repeat visitors to a site because they do change. They include breaking news stories or new corporate press releases.

While examining different Web sites for this study we realized that home pages of sites known to be *portals* often serve as aggregators for links to real sites. To make sure that static and transient links contain content related to that of the site home page, we required these URLs to have the same hostname as the site itself.

In our methodology we explicitly divided the links on each home page retrieval for a Web site into static and transient. We then needed to decide how many of each type of link to follow for content collection. We believed that following only a single link was too little and that following all links was too much, in addition to potentially causing denial of service issues if collection was too frequent. For the study we decided to use up to three of each type of links (not all sites had three transient links). We believe this number allows us to track the dynamics of a subset of popular pages at a site while not overwhelming the site with requests nor our content collector with data. An obvious direction for future work is to examine the effect of alternate criteria for picking the number and type of pages to study at a site.

3.3 Content Collection Methodology

For each Web page URL, the Content Collector that we wrote fetches the container page and all objects embedded in it. The Content Collector does not interpret or parse JavaScript code embedded within HTML, and thus it misses those objects that need to be retrieved because of the JavaScript code execution. The Content Collector detects frames, iframes and layers, then fetches them along with their embedded objects. For each retrieval of each object, the Content Collector stores the current time, complete set of the HTTP response headers, length of the response body, and MD5 checksum that it computes on the object's body. The Content Collector also keeps the content of HTML objects.

We started the Content Collector on June 20, 2002 and it collected content every 15 minutes from 9 am EST until 9 pm EST daily for 14 days, until July 3, 2002. We focused on the daytime hours as the primary time for user and server activity. The 15-minute interval was deliberately chosen to be small enough to capture site dynamics and large enough to avoid any appearance of a denial of service attack. At the beginning of each daily retrieval cycle, the Content Collector saved the 9 am version of all home pages so that the next day it could decide on transient links. We also prefetched and stored all home pages on June 19—one day before we started data gathering.

The Content Collector also retrieved objects that it had seen within the last hour, even if these objects were no longer embedded on any of the pages in our sets. Having information about an object's updates for one hour after that object was first accessed allows us to model server invalidation with the length of a volume lease for up to one hour.

3.4 Content Conversion

We wrote the Content Converter software to convert collected content into format required by our simulator. The Content Con-

Table 4: Dynamics of the Collected Objects

Site	Total	local and global objects (global objects)			
		BoA	RDyn	RSt	St
amazon	2642	1071	23 (2)	1548 (942)	0
boston	3987	1788 (5)	687 (10)	1342 (443)	172 (163)
cisco	76	2	5	69 (5)	0
cnn	1448	58 (20)	87 (3)	1303 (378)	0
espn	2095	742 (14)	71 (9)	1222 (529)	60 (30)
ora	180	17	13 (2)	150 (35)	0
photonet	5256	321	141	4794 (319)	0
slashdot	8830	358 (8)	151 (5)	8321 (907)	0
usenix	56	0	0	56 (18)	0
wpi	86	0	3	83 (26)	0
yahoo	1890	231	30 (7)	423 (88)	1206 (444)

verter first detects object updates by comparing MD5 checksums of the successive retrievals, and then uses the value of the `Last-Modified` HTTP response header, if it was present, or the retrieval timestamp as the time of the update. We are aware that the latter approach provides only an estimate of the exact update time and potentially underestimates the number of updates to an object, but it matches the granularity of our study. The Content Converter creates a list of updates for each object, keeping track of update time, new size, and the set of added and deleted embedded objects.

The Content Converter assigns collected objects appropriate change characteristics using the following rules: 1) an object is BoA if it changes on every retrieval; 2) an object is St if it does not change over the course of the retrievals *and* has `Expires` HTTP response header with a value of one year or more; 3) an object is RSt if the median time between object updates is 24 hours or more; 4) an object is RDyn if the median time between object updates was less than 24 hours.

We did not observe any periodic objects in our data sets. The Cumulative Distribution Function (CDF) of the median times between object updates for all ND objects across all sites is shown in Figure 4. The graph shows that in our data about 15% of all nondeterministic objects are classified as RDyn, with the remaining 85% classified as RSt. The smallest time between object updates that we were able to detect for objects that did not have last modification timestamps was 15 minutes. The graph thus shows virtually no objects with update intervals smaller than 15 minutes. Also, we did not have enough information to determine the median time between updates for those ND objects that did not change during the course of our retrievals. For these ND objects we set the time between updates to one year for the purposes of including these objects in the CDF; that value has no effect on the simulations. As Figure 4 shows, about 85% of all ND objects in our data set did not change over the two-week period of our study.

Table 4 shows the total number of objects collected from each site along with the number of objects in each category of change characteristic. Numbers in parenthesis indicate how many of these objects appeared on more than one page at the site. The Content Converter marks shared objects as *global*. If a frame or a layer is shared between multiple pages, the Content Converter marks all embedded objects of the container as global.

The Content Collector encountered a non-trivial number of redirects, some pointing to the same location on every retrieval and others pointing to different locations. We modeled the former by creating a permanent mapping between the original URL and the new URL. We modeled the latter by introducing a zero-size layer object that on every access contains a different embedded object.

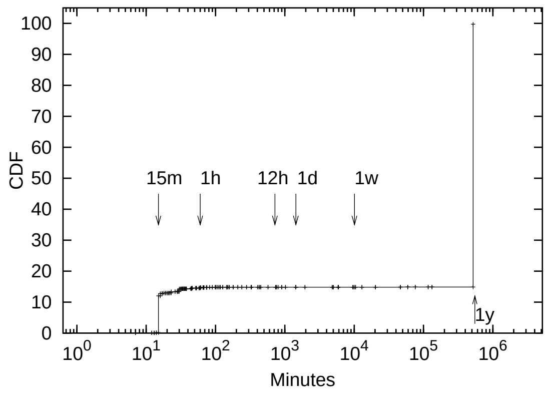


Figure 4: CDF of the Median Time between Non-Deterministic Object Updates

Objects composing Web pages may reside on servers other than the origin server. For this work, we treat all objects composing a page as if they came from the same server. We believe this approach is justified because content served by other servers, such as image or CDN servers, is often under the same control as that from the origin server.

4. PERFORMANCE EVALUATION

Our goal in this work is to evaluate and compare the performance of the cache consistency policies that are currently deployed and that were proposed in research literature. In this section we describe our simulation methodology, the cache consistency policies that we studied, and the performance metrics used.

4.1 Simulation Methodology

For this study we extended the Web Object Management Simulator (WOMS) used in our earlier work [7] to support collected content. Once the collected content is converted, it is presented to WOMS in the form of a (large) configuration file that contains all the necessary information about objects at a site, their relationships, sizes, HTTP response headers and updates. WOMS is also presented with information about what URLs the Content Collector fetched and when.

WOMS takes additional input parameters instructing it which of the collected pages to simulate requests for (container and embedded objects) and at what times to make the requests. Ideally we would base this input on the specific access patterns for pages at a Web site, but without server logs from the site we needed an alternate approach. The approach we used was to investigate a range of possibilities for what content was retrieved on each access and the frequency of these accesses. We simulated the following sets of pages for a site:

- home page only to represent the minimal set,
- home page and the static links to represent a user interested in regular features of the site, and
- home page, the static links and the transient links for that retrieval time to represent the maximal set of the collected content.

In addition, we simulated the following retrieval times based on our collection period of every 15 minutes from 9 am to 9 pm each day:

- every 15 minutes, the maximum rate possible with the granularity of our collection data, which could simulate requests from a proxy server for a pooled set of users,
- once a day at 9 am representing a regular, but relatively infrequent visitor to the site, and
- multiple times a day at 9 am, noon, 4 pm and 8 pm representing a frequent visitor to the site.

These content and frequency patterns yield a total of nine combinations that were simulated for the collected content of each site.

4.2 Cache Consistency Policies

As the simulator processes the requests, it simulates all implemented cache consistency policies. All policies studied in this work faithfully obey object expiration times and do not cache objects marked as uncacheable. All simulations assume an infinite capacity cache. In addition to **MONARCH (M)**, we simulated eight other policies. The **No Cache (NC)** policy mimics a non-caching proxy positioned between a client and a server counting messages and bytes transferred. The **Never Validate (NV)** policy never validates cached objects. The **Always Validate (AV)** policy validates cached nondeterministically changing objects on every access. The **Optimal (Opt)** policy has perfect knowledge of object updates, maintains strong consistency, and contacts the server only when necessary.

We studied the de facto standard **Heuristic** policy, with 5% **(H5)** and 10% **(H10)** of the object's age used as adaptive expiration time for non-deterministic objects. In addition, we examined the **Current Practice (CP)** policy, which is identical to the H5 policy, except the CP policy also faithfully obeys the HTTP directives related to caching, such as `Cache-Control`, `Expires`, and `Last-Modified`. The CP policy exemplifies the behavior of a caching device deployed on the Internet today. The CP policy is the *only* policy in our study that is aware of the HTTP response headers. All other policies use only change characteristics identified by the Content Converter.

We also studied a form of server invalidation—the **Object and Volume Leases (OVL)** policy [10], where servers maintain per-client volume and object leases. Clients must hold valid volume and object leases to reuse a cached copy of an object and to receive object updates from the server. In this work we use one hour as the volume lease length and set the object lease length to be longer than the duration of the simulation. The server sends out updates only for non-deterministic objects. Our simulation assumes reliable and timely delivery of invalidation messages to client caches. We do not account for the details of how to handle updates in the face of slow or unavailable clients [12].

4.3 Performance Metrics

In order to evaluate the performance of each cache consistency policy and to compare the policies we used the following performance metrics. For each policy we computed the number of stale objects served from the cache, the number of requests that the cache sent to the server, and the number of bytes served by the server.

For the MONARCH and server invalidation policies we computed the amount of server state that must be maintained, the number of separate invalidation messages, the number of invalidation messages piggybacked onto server responses, and the average number of objects invalidated in a piggybacked invalidation message. We discuss these state-related metrics in more detail in Section 5.3.

5. RESULTS

For each of the eleven sites we performed simulations with all scenarios discussed in Section 4.1. For the purpose of the discussion, unless indicated otherwise, all results in this section are from the scenario where the home page, static links and transient links are retrieved from a site four times each day. This scenario was chosen for focus from the nine described in Section 4.1 because it represents the maximal amount of content at an intermediate access frequency. The relative performance of different policies for the other scenarios is generally consistent in tone with those shown. More frequent accesses result in more reusable cache content, less frequent accesses result in more content that must be retrieved from the server. In general, the content on site home pages is more dynamic than the content of linked pages.

We first discuss the effectiveness of the policy that models the behavior of modern caches. Then we compare the performance of different policies in terms of the amount of generated traffic and staleness. After that we discuss the amount of overhead that the two stateful policies incur at the server. Finally, we examine the extent to which cache consistency policies affect end user response time.

5.1 Effectiveness of Current Practice

One of the performance goals in this work is to evaluate the effectiveness of the cache consistency policy that reflects the current practice. Performance of the CP policy across all eleven sites in terms of the average number of requests that the server received and the average number of KBytes that the server served per page retrieval is shown in Table 5. For comparison, the table also shows the best (Opt) and the worst (NC) case policies. The results indicate that the CP policy avoids transferring 50–60% of bytes (up to 96% for the `usenix` site) as compared to the NC policy. For seven sites in our set the CP policy also transfers only marginally larger number of bytes than the Opt policy. For the other four sites, however, the CP policy transfers 1.2–7 times more bytes than the Opt policy. We investigated the reason for such discrepancy and discovered that sites often mark objects that change infrequently as uncacheable or generate such objects upon request and provide no information that caches can use to subsequently validate these objects. The results further indicate that the CP policy issues 1.8–5.4 times more requests to the server than the Opt policy. In terms of staleness, the CP policy serves stale objects for eight sites in at least one simulation scenario. For two sites (`usenix` and `ora`) the CP policy serves stale objects under all simulation scenarios.

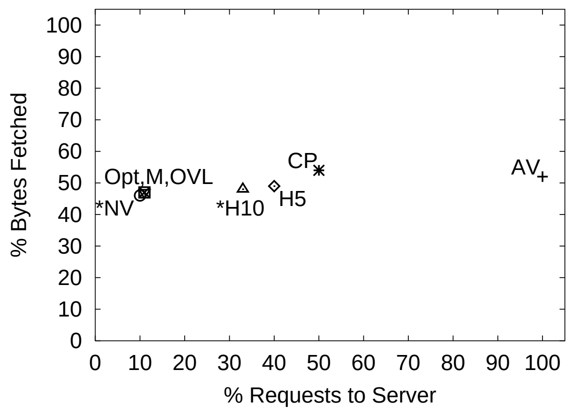


Figure 6: ESPN, 38.7 Requests, 159.5KB under the NC Policy

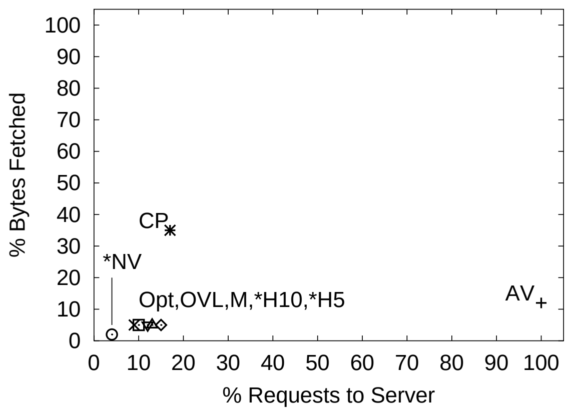


Figure 7: Cisco, 19.2 Requests, 55.4 KB under the NC Policy

Table 5: Performance of the Current Practice Policy (* indicates stale content served in at least one simulation scenario)

Site	Requests and KB served by Server					
	Opt		CP		NC	
amazon*	3.2	45.1	5.9	45.7	35.2	107.5
boston*	3.6	50.6	19.3	54.4	25.5	113.5
cisco	1.9	2.9	3.5	19.6	19.2	55.4
cnn*	6.3	56.1	16.4	77.6	31.4	190.8
espn*	4.3	75.3	19.4	85.4	38.7	159.5
ora*	0.9	13.8	2.7	14.2	19.9	97.1
photonet	3.1	33.4	3.7	34.5	8.5	55.5
slashdot*	3.1	38.4	7.8	39.5	15.0	70.5
usenix*	0.3	0.8	0.9	0.9	21.1	28.4
wpi*	0.5	2.9	2.3	15.3	25.8	61.2
yahoo	3.7	34.1	6.5	39.2	15.6	71.0

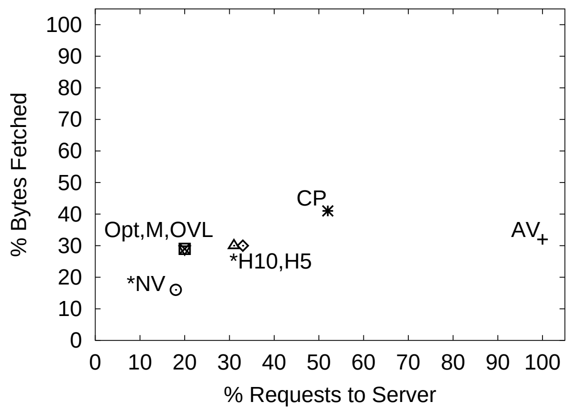


Figure 5: CNN, 31.4 Requests, 190.8 KB under the NC Policy

5.2 Comparison of Cache Consistency Policies

We examined performance of policies other than CP on all sites and present results for three sites—cnn, espn, and cisco—in Figures 5–7. These sites represent a range of policy results. The horizontal and vertical axes are expressed in percentages relative to the NC policy. On the horizontal axis we plot the percentage of requests that each policy sent to the server per page retrieval, and on the vertical axis we plot the percentage of bytes that the server served under each policy. Policies that served a non-zero number of stale objects to clients in these simulation are marked with asterisks. The number of requests and bytes under the NC policy is shown in the figure captions.

The graphs indicate that caching of content using any of the policies shown, including the AV policy, offers substantial (at least 50–60%) byte savings. The two heuristic policies H5 and H10 outperform the CP policy both in terms of requests and bytes. The results indicate that in terms of the traffic between the cache and the server both MONARCH and the OVL policies provide indistinguishable performance from the Opt policy.

Staleness results across all sites indicate that under at least one simulation scenario both H5 and H10 policies on average served a small number of stale objects per page retrieval. The amount of stale content served is substantially smaller than the upper bound on staleness provided by the NV policy. Across all sites and all simulation scenarios the NV policy served on average 0.4–3.0 stale objects per page retrieval, and for the simulation scenario used in this discussion it served 0.4–1.5 stale objects.

5.3 Server Overhead

The MONARCH and OVL policies provide strong cache consistency and also exhibit similar performance in terms of the generated traffic. However, compared to other policies studied, the MONARCH and OVL policies incur overhead at the servers because they require servers to maintain state and perform additional processing to achieve strong cache consistency. In this section we describe the metrics that we use to evaluate server overhead imposed by each of the two policies and present values for these metrics obtained from the simulations.

Both the MONARCH and OVL policies must keep track of updates to non-deterministically changing objects in order to maintain volume information and notify clients of such updates. As a measure of the overhead associated with these updates we compute the average number of daily updates to non-deterministic objects (**NDU**). The NDU results for each site are shown in Table 6.

Table 6: Server Overhead

Site	NDU	MONARCH		OVL (AOL)	
		VOL	VR	avg	max
amazon	4	1271	159	464	706
boston	300	138	218	409	657
cisco	125	4	121	67	70
cnn	109	84	198	580	941
espn	67	63	167	525	806
ora	29	22	13	122	151
photonet	220	434	211	432	770
slashdot	167	33	330	404	731
usenix	0.3	5	0	55	56
wpi	2	6	0	83	86
yahoo	110	51	187	145	248

MONARCH maintains per-page local volumes and one global volume that incorporates objects shared between pages. MONARCH increments a volume version when volume membership changes. We capture the overhead associated with volume maintenance using the average number of the unique volume revisions (**VR**) created daily. In computing the VR numbers we take into account original volume versions created at the start of the simulation. The average daily number of volume revisions and the total number of local volumes created (**VOL**) are shown in Table 6.

The OVL policy maintains a list of per-client volume leases and per-client object leases. We focus only on object leases as the overhead associated with volume leases is likely to be significantly smaller than that of object leases. We measure the overhead of object leases by recording the number of active object leases (**AOL**) held for one client after it makes requests. Table 6 shows the average and maximum number of active volume leases held for one client at each site.

As we examine the metrics shown in Table 6, we see that the rate of updates to non-deterministic objects varies from 0.3 to 300 per day. We further investigated the NDU results and discovered that for two sites, including `boston`, only a handful of objects (10 or fewer) are responsible for over 50% of all NDU updates. These frequently changing objects could be marked as BoA instead of RDyn to reduce the overhead associated with changing objects, albeit with diminished cached content reuse.

The highest daily number of volume revisions in our simulations is 330 and was observed for only one site. For seven other sites, the daily VR increase is under or slightly over 200. The number of volume revisions for less frequently changing sites, such as `ora`, `wpi` and `usenix`, is either small or zero. Our results show that for six sites the OVL policy must maintain over 400 active object leases per-client, and must also maintain leases even for sites that do not have many object changes. We also investigated the effect that frequency of request arrivals has on the overhead of the two policies. For four sites—`wpi`, `usenix`, `ora`, and `cisco`—the overhead of the OVL policy remains unchanged as the request arrival rate increases from 4 times a day to every 15 minutes. For the other seven sites the number of active object leases maintained by the OVL policy increases as follows. For five of the seven sites the average AOL grows by 27–58% and the maximum AOL grows by 29–48%. For the other two sites—`photonet` and `slashdot`—the increase is especially large. For the former site, the average and the maximum AOL increase by 5.6 and by 5.7 times respectively. For the latter site, the average and the maximum AOL increase by 9.6 and by 10.2 times respectively. The overhead of MONARCH is not affected by fluctuations in the request arrivals or the number of clients.

We also examined the overhead associated with the invalidation activity of the MONARCH and OVL policies. The invalidation behavior of the two policies is different and cannot be compared directly. MONARCH always piggybacks invalidations onto its responses to clients, while the OVL policy sends out invalidations both piggybacked onto other messages and as separate messages. Our results indicate, however, that these differences are not that important. Invalidation traffic in terms of separate messages, piggybacked messages and objects invalidated in one message is negligible for both policies across all sites and all simulation scenarios.

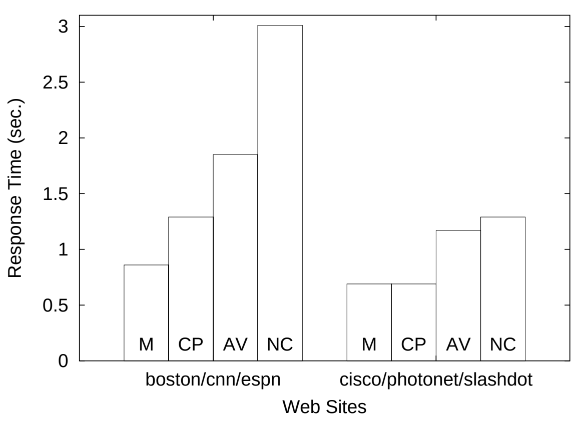
5.4 Response Time Implications for Different Policies

This study allows us to determine the performance of different policies in terms of the requests and byte traffic between caches and servers. It is less clear how this performance impacts the end user. As a means to study this issue we use performance data that was gathered as part of another work [5].

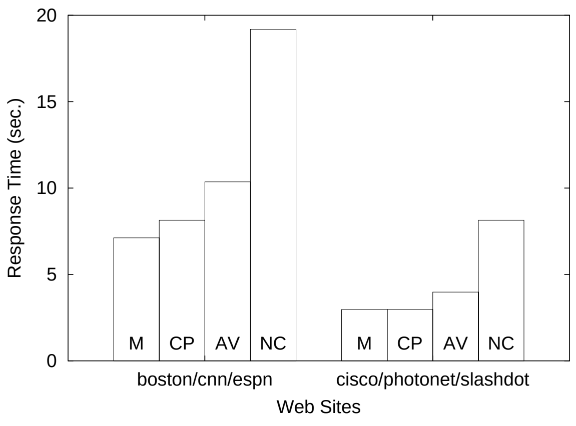
The authors in [5] characterized pages based on the amount of content on a page, which is defined as the number of bytes in the container object, the number of embedded objects and the total number of bytes for the embedded objects. Using proxy logs of a large manufacturing company, popular URLs containing one or more embedded objects were successfully retrieved and the 33% and 67% percentile values were used to create a small, medium and large value range for each characteristic. Using these three ranges for each of the three characteristics defines a total of 27 "buckets" for the classification of an individual page. The cutoffs for container bytes in small, medium and large were less than 12K, less than 30K bytes, and more than 30K bytes respectively. Similarly, for embedded objects it was less than 7, 22, and more than 22 and for embedded bytes 20K, 55K, and more than 55K bytes. The authors identified test pages that spanned the space of these characteristics and created a test site of content. They installed the test site on unloaded servers on both coasts of the U.S. and used *httperf* [8] from six other client sites to make automated retrievals to each test server for each test page.

In this work, we use the results from [5] as benchmark response time performance measures for different types of clients and amounts of content. We focus on results from retrievals using up to four parallel TCP connections and HTTP/1.0 requests. Persistent TCP connections with pipelining are expected to produce better results, but pipelining is not commonly used by real clients and proxies. Persistent connections with serialized HTTP requests have been shown to perform no better than four parallel connections [5]. Our methodology is to map the requests and amount of content served under each policy in this work to a corresponding bucket from [5]. We then use the benchmark performance of different clients tested in [5] as an estimate of the relative performance of the various policies.

The buckets in [5] are only coarse classifications and, not surprisingly, in many cases policy traffic performance maps to the same bucket. For example, the Opt, MONARCH and OVL policies invariably map to the same bucket across different Web sites and retrieval patterns. This convergence is realistic as only significant differences between policies for the number of objects or number of bytes is going to translate into significant response time differences between the policies. The heuristic policies sometimes map to the same bucket as the Opt, MONARCH and OVL policies and in other cases map to the same bucket as CP. The AV and NC policies generally map to distinct buckets. Given these observations,



a. from Commercial Client



b. from Modem Client

Figure 8: Estimated Response Time Performance for Different Policies for Web Site Pages Using Para-1.0 Results in [5]

we show results for the MONARCH, CP, AV and NC policies for a commercial and modem client.

Figure 8a shows results for two sets of sites in our study. The response time results, obtained in [5], are from a commercial client on the East Coast of the U.S. to the West Coast server. Results for the `boston`, `cnn` and `espn` sites are mapped to the same buckets and are shown together. Response time for the MONARCH policy is improved relative to current practice and is much better than no cache, although the absolute differences are smaller because the client is well connected. Figure 8a also shows that for pages on Web sites with less content, there is less difference between the performance of different policies. The AV policy generally yields worse response time than current practice, although pipelining of responses can reduce the difference.

We also used results from a modem client on the East Coast to the East Coast server. These results for the various policies and Web site pages are shown in Figure 8b. Due to the reduced bandwidth of the client, the absolute differences between the policies is greater, particularly for Web site pages with more content.

The results above are averages for all pages at each Web site. We also examined the relative response time differences when retrieving just the home page at multiple times each day. For this analysis, four of the sites showed some response time difference between the MONARCH and CP policies. Overall, the results show that better cache consistency policies can improve expected response time relative to current practice for larger, more dynamic, pages.

6. RELATED WORK

The issue of maintaining strong cache consistency on the Web has been explored by a number of researchers. Gwertzman and Seltzer compared heuristic approaches to Web cache consistency with server invalidation [3]. They showed that heuristic approaches have substantially lower cost than server invalidation and concluded that weak consistency approaches are the most suitable for the Web. They did acknowledge, however, that invalidation protocols are required for perfect cache consistency. Cao and Liu also compared heuristic approaches with polling every time and invalidation [6]. Their work showed that the cost of strong consistency achieved with server invalidation can be comparable to the cost of the heuristic approaches. Subsequent work by Yin et al. introduced volume leases as a means of reducing the cost of server invalidation [10]. They showed that the introduction of volumes reduces traffic at the server by 40% and can reduce peak server load when popular objects are modified. Yin et al. further explored scalability aspects of volume leases and proposed to extend volume leases to cache consistency hierarchies [11]. Yin et al. also explored engineering techniques for improving scalability of server-driven invalidation [12].

Our work differs from the server-driven approaches to invalidation in two main aspects. First, MONARCH does not require servers to maintain per-client state and does not send object updates to multiple clients in bursts. Invalidation in MONARCH is driven by client requests. Second, MONARCH does not require servers to delay object updates until all clients that hold valid leases are notified, which eliminates the issue of waiting for slow or unreachable clients. MONARCH can be deployed using the HTTP protocol and without any extra support from the network, such as multicast.

Cohen et al. investigated approaches for grouping related Web objects into *volumes* and reducing volume size [1]. They explored directory structure and client access patterns as indicators of relationships between objects. In our work we construct volumes based on known relationships between objects instead of using heuristics to infer relationships.

Shi et al. also download pages from a set of Web sites at regular intervals. They use the collected data to analyze page structure and derive models that characterize dynamically generated content [9].

7. SUMMARY

This work examines a new policy, MONARCH, for strong cache consistency on the Web and uses a novel evaluation method to compare the performance of this policy with current and proposed policies. One contribution of the work is the evaluation of MONARCH, which combines object relationships with object change characteristics to manage Web objects with strong consistency and no per-client state at the server. Results of the evaluation show MONARCH generates little more request traffic than an optimal cache coherency policy.

The evaluation methodology used in this work is also a contribution. Our methodology is to actively gather selected content from sites of interest. This content is then used as input to a simulator to evaluate a wide range of cache consistency policies over a range of access patterns. Traditional use of proxy and server logs for such evaluation has its limitations. While proxy logs contain real client request patterns, they do not contain the complete request stream to a particular site and provide no indication of when resources change. Logs from popular server sites are not generally available to the research community and do not contain a record of object modification events. Our content collection methodology is a step towards filling these gaps and obtaining data for any site of interest that is not otherwise available for study.

Results of the simulations with real content, obtained in this work, agree with the limited results from the simulations with synthetic content, obtained in our previous work [7]. While we do not discount the usefulness of the simulations with synthetically generated content, we see substantial value in using real content to drive simulations. First, downloading content from sites of interest allows us to obtain HTTP cache directives used by these sites, and not only compare proposed policies with each other, but to compare them with current practice. The results show that for some sites the current practice policy yields close to optimal cache performance, but for larger, more dynamic, sites it generates more request and more byte traffic than is necessary. Second, using real content provides more realistic composition of objects and change characteristics, including more realistic sharing of objects across pages, than synthetic content. This better understanding of page composition can lead to better synthetic content generators, and steps are already being taken in that direction [9].

The simulation also allows us to compare the relative overhead of strongly consistent cache policies, although each policy evaluated incurs different types of overhead. The Always Validate policy can generate a large number of requests, the Object and Volume Lease policy requires the server to maintain a potentially large number of per-client leases and the MONARCH policy must track volume membership changes. Results for the OVL policy show that for many of the sites studied the number of per-client object leases is in the hundreds even though few of the objects actually change. The amount of state for the MONARCH policy is up to a couple hundred volume membership revisions per day.

The last part of this work uses results from previous work to study how traffic variations between the policies affect the response time that might be expected by a user. The results show that for all sites, current caching practice yields significantly better response time than if no caching is used. For some sites and pages, proposed policies provide no improvement in terms of estimated response time. However, for Web site pages that require many validation requests, proposed policies, such as MONARCH and OVL, do show response time improvement over current practice.

As part of our ongoing work we are interested in further exploring our methodology of taking snapshots of a site. A better understanding of the appropriate size and frequency of these snapshots is needed. In terms of the MONARCH approach, we are interested in examining the use of alternate object relationships such as common object dependencies. Finally, we are interested in investigating potential performance improvements if Web sites expose the internal components of large container pages to caches. With the weak consistency model of current caching, servers have limited control over when cached objects are validated. With the strong cache consistency approach of MONARCH, Web servers would be able to reliably invalidate such cached objects if they change.

8. ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for the detailed and thoughtful comments that helped us improve the presentation of the paper.

9. REFERENCES

[1] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *ACM SIGCOMM'98 Conference*, September 1998.

[2] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol—HTTP/1.1. RFC 2616, June 1999.

[3] J. Gwertzman and M. Seltzer. World-Wide Web cache consistency. In *Proceedings of the USENIX Technical Conference*, pages 141–152. USENIX Association, Jan. 1996.

[4] B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherency. In *Proceedings of the Seventh International World Wide Web Conference*, pages 185–193, Brisbane, Australia, Apr. 1998.

[5] B. Krishnamurthy, C. E. Wills, and Y. Zhang. Preliminary Measurements on the Effect of Server Adaptation for Web Content Delivery. In *Proceedings of the Internet Measurement Workshop, Short abstract*, Marseille, France, Nov. 2002.

[6] C. Liu and P. 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] M. Mikhailov and C. E. Wills. Exploiting Object Relationships for Deterministic Web Object Management. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, Boulder, CO, Aug. 2002.

[8] D. Mosberger and T. Jin. httperf – a tool for measuring web server performance. In *Workshop on Internet Server Performance*, Madison, Wisconsin USA, June 1998.

[9] W. Shi, E. Collins, and V. Karamcheti. Modeling Object Characteristics of Dynamic Web Content. In *Proceedings of the IEEE Globecom*, Taipei, Taiwan, Nov. 2002.

[10] J. Yin, L. Alvisi, M. Dahlin, and C. 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.

[11] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Hierarchical Cache Consistency in a WAN. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, USA, Oct. 1999. USENIX Association.

[12] J. Yin, M. Dahlin, L. Alvisi, C. Lin, and A. Iyengar. Engineering server driven consistency for large scale dynamic web services. In *Proceedings of the Tenth International World Wide Web Conference*, Hong Kong, May 2001.