Performance Evaluation of the Distributed Object Consistency Protocol

Stephane Perret, John Dilley, Martin Arlitt
Internet Systems and Applications Laboratory
HP Laboratories Palo Alto
HPL-1999-108
September, 1999

World Wide Web, proxy cache, cache replacement policy, distributed object consistency protocol, performance analysis, trace-driven simulation

This report analyzes the performance of two methods for maintaining object consistency in World Wide Web proxy cache servers: the widely used Alex adaptive TTL protocol and a proposed Distributed Object Consistency Protocol (DOCP). The goal of this report is to evaluate these approaches to web cache consistency using a realistic workload and trace-driven simulation. We examine the effect of cache replacement policies and consistency protocols on cache performance.

This analysis indicates that the DOCP outperforms the Alex protocol while preventing access to inconsistent objects. DOCP also improves cache response time by serving consistent objects without communicating with the origin server to check freshness. This improves fast hit rate and reduces request demand on origin servers.

Internal Accession Date Only

Performance Evaluation of the Distributed Object Consistency Protocol

Stéphane Perret, John Dilley, Martin Arlitt

HP Laboratories, Palo Alto, CA

Abstract

This report analyzes the performance of two methods for maintaining object consistency in World Wide Web proxy cache servers: the widely used Alex adaptive TTL protocol and a proposed Distributed Object Consistency Protocol (DOCP). The goal of this report is to evaluate these approaches to web cache consistency using a realistic workload and trace-driven simulation. We examine the effect of cache replacement policies and consistency protocols on cache performance.

This analysis indicates that the DOCP outperforms the Alex protocol while preventing access to inconsistent objects. DOCP also improves cache response time by serving consistent objects without communicating with the origin server to check freshness. This improves fast hit rate and reduces request demand on origin servers.

1 Introduction

To improve service to web clients web proxy cache servers have been deployed throughout the network. These cache servers store copies of objects requested by web users and subsequently serve those objects to users if requested again. By serving cached objects, the proxy reduces network and origin server demand. However, the objects they serve are not necessarily current with the origin server. This *weak consistency* leads content providers to disable caching for some objects and users to reload pages when they suspect inconsistency. This increases server load, especially during *flash crowds* when many users visit the site at the same time.

Web consistency and the Alex consistency protocol are discussed in detail in [9]. The Distributed Object Consistency Protocol (DOCP) is defined in [10]. This report presents the results of a trace-driven simulation of the DOCP and Alex consistency protocols. The simulation establishes that DOCP provides strong web consistency without adding network traffic or origin server load.

1.1 HTTP Web Cache Consistency

HTTP/1.0 provides an approach to maintain consistency of cached objects based on validation. Using this mechanism a client or proxy cache queries an origin server with an object's last known modification date using an HTTP GET request with an **`If-Modified-Since`** header. If the object hasn't changed at the origin server, the server responds with a validation of freshness. Otherwise, the updated object is returned as a regular response. The server can also add an **`Expires`** header to indicate when the object should expire from the cache and be re-validated. in practice this header is difficult to use because it is hard to predict when an object will next be modified.

HTTP/1.1 [12] adds mechanisms to allow clients, proxy caches, and origin servers to control caching policy. These headers can identify the maximum “staleness” allowed for an object (how old an object can be served from cache without validation), the minimum freshness (again, how old), the maximum age (how old); or they can force revalidation of the object with the origin server. These mechanisms are described in over 30 (of 160) pages in the HTTP protocol definition (RFC 2616 [4]). These headers are complex and difficult for users and origin servers to use effectively. The DOCP aims to replace many of these complex mechanisms with a simpler, more robust one.

The most common consistency mechanism currently in use in the web uses an adaptation of the heuristic developed in the Alex protocol [6], in which each object is assigned a time-to-live (TTL) in the cache as a percentage of the age of the object when accessed. The age is the difference between the current time and the object's age as determined by the **`Last-Modified`** date reported in the server headers when the object was accessed. (Not all servers return this value; the cache can only apply this protocol successfully when the modification date is supplied.)

This age-based approach is effective in practice since the probability that an object will change is proportional to the age of the object: new objects tend to change sooner than old objects. More formally, the probability of object modification follows a Zipf (power-law) distribution based upon the time since last modification (t) and a constant alpha (a).

$$P(t) \sim t^{-a} \qquad \text{(EQ 1)}$$

While the Alex protocol has good synergy with Web content modification probability there is still a possibility of inconsistent access. This prevents content providers from being able to trust that objects in the cache are consistent with what they intended, and thus limits use

of caching by providers who want greater control over the distribution of their objects.

1.2 Proposed Consistency Protocol

Strong consistency in the web can be achieved by forcing a validation with the origin server on every request, effectively tunneling through the cache. This increases the request load at the origin server and the response time of user requests [10]. The time to validate an object with the origin is about the same as that of returning a 2 KB object, and is much longer than serving an object directly from the cache.

To improve user response time and object consistency in the web we designed a protocol that delivers strong consistency in the web. The Distributed Object Consistency Protocol (DOCP [9]) uses invalidation after an object is modified rather than occasional validation after an object is requested.

Section 2 explains the methodology used to build the simulator and Section 3 gives some details about its implementation. Section 4 characterizes the workload, and Section 5 describes the simulation we performed given this workload. Finally, Section 6 presents the results and our findings about this new approach of caching in the World Wide Web.

2 Methodology

This analysis uses using trace-driven simulation to examine consistency protocol behavior Trace-driven simulation consists of replaying logs (traces) collected from real users through a simulator that is as similar to the protocol and underlying network as is feasible.

2.1 Trace-Driven Simulation

We chose trace-driven simulation because the variability in Internet traffic makes it difficult to model accurately. Using a trace of actual activity guarantees a valid model of user activity. A disadvantage of trace-driven simulation is that as real workloads change, the trace can not capture that effect. For example, a move to higher speed access networks may affect user demand and therefore the cache workload. Furthermore, many traces openly available in the Internet community are for a short duration or small user base. To be realistic a trace needs to be able to describe real user workload over a sufficiently diverse set of requests. This is particularly important when studying caching, as repeated access to objects over a time period is essential to measure actual cache behavior.

Fortunately, we had access to a large data set from a busy proxy server [3]. Data were gathered by tracing every request made by a population of thousands of home users connected to the web via cable modem technology over a five-month period (for a total of 117 million requests). This user demand should be similar to that of other high-speed home users. In this study we augmented the simulator previously used to explore the effect of various cache replacement policies [2] to simulate the proposed DOCP consistency protocol.

One difficulty with such a simulation is modification information is not available in proxy logs nor in origin server logs. The origin server logs can provide an estimate of the modification rate for popular objects, but can not identify when an object changes more than once between accesses. We examined origin server logs and related research [11] to gain an understanding of the modification profile for web objects. In Section 3.2 we present this analysis in detail.

Using the modification profile we created a statistical model to simulate object modification. The model distributes object modifications across the set of objects observed in the trace. We adjusted the modification rate to test the sensitivity of the protocol to the assumptions about object modification rate.

2.2 Proxy Simulation Model

FIGURE 1. Simulation Model

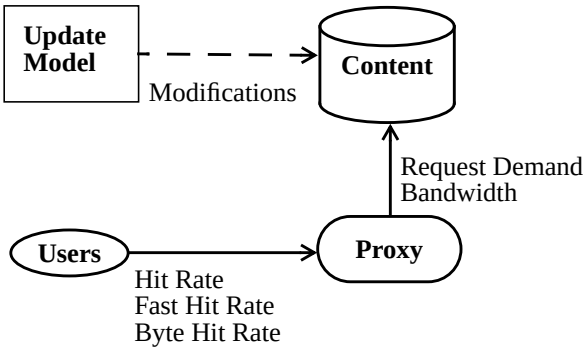


Figure 1 describes the simulation model and lists the metrics collected during this study.

- *Hit rate*: the percent of all object GET requests that were served with data from the cache (including those that required validation with the origin server).
- *Fast hit rate*: the percent of GET requests that were served directly from the cache without validation from the origin server (i.e., without external communication).
- *Slow hit rate*: the percent of GET requests that were served by the cache after contacting the origin server to validate object consistency.
- *Byte hit rate*: the percent of all bytes served to clients that came from the cache (including those that were served following object validation with the origin server).
- *Request demand*: the number of requests from the proxy cache to all external origin servers.
- *Bandwidth demand*: the number of outgoing and incoming bytes between the proxy cache and all external origin servers.

Separation of cache hit rate into *fast* and *slow* hits allows us to estimate cache response time based upon the cache consistency model.

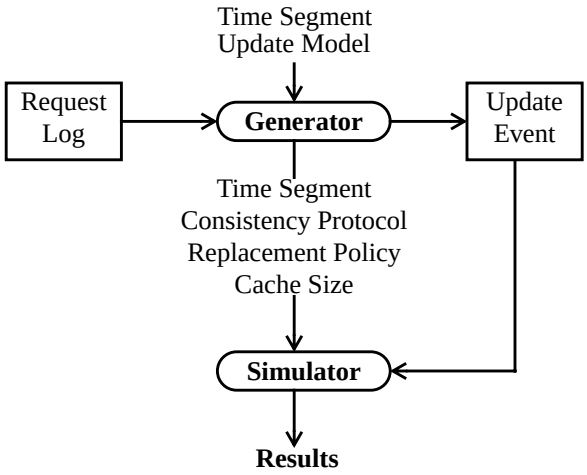
3 Implementation

The simulated system includes origin servers served by DOCP masters and one DOCP slave cache serving a population of users. Origin servers maintain web objects and handle object requests and modifications. DOCP slave caches maintain copies of requested objects, handle object requests from users and invalidation requests from DOCP masters. The system simulates consistency protocols and cache replacement policies. It does not attempt to simulate wide area network delays or errors.

3.1 Overview

The content used for the simulation is the set of objects requested within a given time period from a *request log*. An update model assigned modifications to the objects within the simulated time period. The simulator replays the requests and updates and records the results. These steps are depicted in Figure 2 and explained below.

FIGURE 2. Simulation Platform



The **`generator`** produces the update events according to the update model and the indicated time segment to be simulated from a proxy cache request log. The **`generator`** builds a table of all unique objects in the request log and distributes updates among them. The updates are sorted by timestamp and stored in the *update event* file. This gives consistent, repeatable modification behavior.

The **`simulator`** replays both the *request log* and *update event* files and simulates protocol operation. The **`simulator`** collects simulation results after a configurable warm up period.

We faced several issues in the development of this simulation platform. The following sections summarize the issues and how we resolved them. The most important was the assignment of modifications to objects.

3.2 Object modification

Arlitt et al developed a method to detect possible modifications from a log [3]. This method assumes that a file modification will result in a relatively small change in reported file size. By contrast an aborted transfer will generally result in a much smaller transfer size. This heuristic allows estimation of the number of aborted transfers and object modifications.

This method was not adequate to assign modifications to object for this simulation because a precise modification time is required. The modification time should not be the same as the access time, since this is known not to be consistent with actual object modifications. Furthermore multiple modifications may occur between accesses, which may influence the behavior of a consistency protocol. Instead, we used a statistical model for updates.

To create the model we analyzed logs from the **`www.hp.com`** web server to detect possible object modifications. The detection method described above is most accurate when considering popular objects on busy servers: higher request frequency gives a better estimate of object modifications by reducing the window between successive requests. This validation considered the top 15% of cacheable objects on the site, which received 90% of the requests. The analysis used one week data.

TABLE 1. Cacheable Object Modification Rate

	HTML	Other	Total
Requests	6,067,086	47,174,278	53,241,364
Objects	9,898	4,381	14,279
Modified	5,299	4,381	9,680
Updates	230,844	47,241	278,085

Table 1 shows that HTML objects are modified more frequently than images and other objects, as others have observed [11]. This indicated that updates to HTML objects should be modeled independently from other objects. The other types were modeled together.

Most objects are not modified frequently, but a few were updated very frequently. Earlier studies [8][1][5] have observed a Zipf (power-law) distribution for object modification and object popularity. Under a Zipf distribution, popular objects are very popular, but the distribution has a heavy tail: as popularity decreases there is still a significant probability that the object is accessed. By comparison, an exponential distribution decays much more rapidly, so unpopular objects under an exponential distribution would receive few to no requests. Looked at another way, the area under an exponential curve is finite, but the area under a power-law curve is infinite. Based on the data from **`www.hp.com`** and the prior research we chose a Zipf distribution to model modifications.

On this server the average frequency of modifications for popular objects is about 4 orders of magnitude smaller than for the requests (9 hours versus 3 seconds). While this does not have bearing on the object modifica-

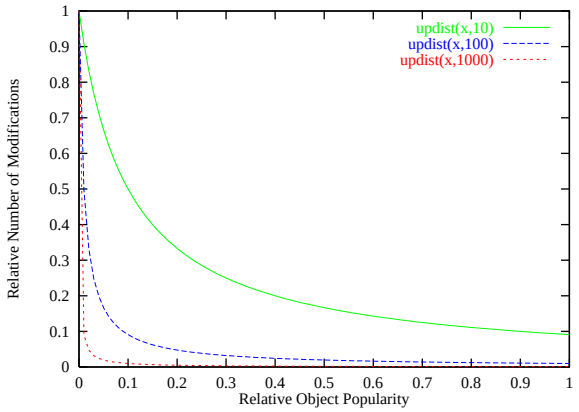
tion distribution it is encouraging, and illustrates significant potential for performance improvement from consistent caching.

To test the sensitivity of the simulation to assumptions about update frequency, we generated a family of heavy-tailed distributions. The following function is a distribution generator for Zipf-like distributions.

$$updist(x,a) = 1/(ax+1) \qquad \text{(EQ 2)}$$

where *a* is the shape parameter and *x* a uniform random variable in [0,1). Figure 3 illustrates three such distributions from the family.

FIGURE 3. Update Frequency Distribution



We implemented the **`generator`** to produce update events for each of the two classes of objects: HTML and others as follows.

- First, it assigned the total number of modifications during the interval to all objects using the **`updist`** function multiplied by a scale factor. The scale factor determines the how often an object can possibly change. This factor is based upon estimated update frequency from the logs. At the conclusion of this step, each object has some number of modifications assigned to it.
- Second, it assigned modification times for each object. This assignment used a Poisson process to determine the interval between modifications. This process used the exponential function **`exp`**, described in EQ. 3 to distribute the updates within the simulation interval. In EQ. 3 *mu* is the object update frequency from the previous step and *x* a uniform random variable in [0,1).

$$exp(x,mu) = -mu*ln(x) \qquad \text{(EQ 3)}$$

- Finally, it sorted all the update events by timestamp.

3.3 Cache validation

We had to distinguish cache validations requests from regular requests in order to replay a realistic situation. Most browsers use a private cache and send out validations when the user requests an object more than once. It is important to take into account this hierarchical behavior when simulating a cache consistency protocol.

Unfortunately, the logs do not include client request headers. The HTTP status code indicates whether a response was a validation, but not whether the request was for a validation. When a server receives a conditional request (with an **`If-Modified-Since`** header) it responds with a **`304 Not Modified`** status code if the requested object was not updated since the last retrieval. If the object was modified the server responds with a **`200 OK`** status code. The number of **`304 Not Modified`** responses therefore provides a lower bound on the number of validation requests. 15% of responses in the logs used in this simulation resulted in **`304 Not Modified`** responses.

3.4 Simulated Consistency Protocols

The goal of this study was to analyze the behavior of DOCP as compared with the following approaches to handle cache consistency in the World Wide Web.

3.4.1 Polling for Strong Consistency

Object validation with the origin server is the only alternative for strong consistency using today's HTTP protocol. This mechanism results in relative high response time and server load when compared with the other alternative, weak consistency. When used judiciously this mechanism can allow a content provider to deliver their content consistently and measure user access.

For the simulation we model the strong policy by counting each request as a slow hit or a miss, depending on if the object was in the cache or not.

3.4.2 Alex Consistency Protocol

The Alex consistency protocol [6] is used by the Squid open source cache server and other Harvest [7] derived works. It is probably the most widely deployed web cache consistency protocol.

The Alex protocol assigns each object a time to live (TTL) in the cache. The TTL value is computed using three values: the percentage of the object's age, a minimum limit, and a maximum limit on the TTL. See [10] for a detailed description of this protocol.

We developed a simulation module for the Alex protocol as it is implemented in Squid using these parameters. We also identified a trick for handling cache validations. While reading Squid source we noticed that it does not always propagate user validation requests to the origin server. Squid permits the cache administrator to change any **`Pragma: no-cache`** request (specified by a client to request not to use a cache) into a cache validation. This can negatively impact object consistency if the cache is in a hierarchy: At the first level, the "no-cache" request becomes a validation request, which is validated by the second level cache directly, rather than with the origin server. We analyzed the behavior of the Alex protocol with and without this feature, which is referred to as the *weak validations* feature.

To summarize, the parameters for the Alex protocol are:

- *Min*: minimum TTL in the cache,
- *Pct*: percentage of the age used to set the TTL,
- *Max*: maximum TTL in the cache,
- *Ims*: weak validation feature switch on or off.

3.4.3 Distributed Object Consistency Protocol

The simulation also examined the performance of the proposed Distributed Object Consistency Protocol, DOCP [9]. With this protocol a DOCP slave proxy cache guarantees object consistency to the client. The DOCP cache implements strong consistency using validation until an object becomes sufficiently popular. When an object becomes popular, the slave cache asks for a subscription to that object with a DOCP master agent for the origin server. If a subscription is granted the slave receives a lease interval for the object. If a subscribed object changes during the lease interval the DOCP master's notification agent sends an invalidation notification message to each subscribed slave proxy cache. The lease represents a time-bounded agreement to receive an invalidation message if the original copy is updated or deleted. After the lease expires the cache may renew the lease on a subsequent client request; otherwise it uses polling to achieve strong consistency.

The simulator assisted during protocol development to help define what is "sufficiently popular", and to establish an appropriate lease interval. Popularity is the frequency of access: the number of requests within a given time interval. An invalidation has about the same cost as a validation, so the protocol strives to subscribe early. If an object has been requested in the previous 24 hours a subscription will be requested. Other values for the popularity threshold were simulated, which confirmed our assumption. The threshold was made a protocol constant based upon early simulation, rather than a parameter.

We also explored setting the lease interval based upon object modification profile observed by the DOCP master. We used the modification history to estimate the most likely time of next update and configured the lease to end prior to that. From simulation we learned that it is difficult to accurately determine both the next update and access times. This method led to leases that were too short, causing extra network traffic. We concluded that a simple constant lease interval was more efficient than a complicated lease function based upon metadata whose ability to predict the future cannot be assured.

We do not present a further analysis of the popularity function or the dynamic lease algorithm. The only the parameter we considered for the DOCP was:

- *Lease*: the maximum period of subscription for an object

4 Workload

This section presents details about the *request log* and the *update event* files used for the simulations.

4.1 Time Segment

The full log period covers five months with 117 millions requests. This simulation extracted the first 10 weeks (6048000s) and analyzed only cacheable HTTP traffic (the dominant traffic component). 92% of HTTP requests had cacheable responses, which accounted for 95% of the total HTTP content transferred. The following tables show the breakdown for the HTTP traffic.

TABLE 2. Simulation Trace Characteristics

Total Time Segment	10 weeks
Total HTTP Requests	46,263,216
Cacheable	42,733,169
Not Cacheable	3,530,047
Total Content Transferred	460.61 GB
Cacheable	442.06 GB
Not Cacheable	28.55 GB

This workload consists of 42.7 million requests for 7.7 million unique cacheable objects. These objects represent 104 GB of content, which would require 442 GB of traffic if none of it was cached and 104 GB of external traffic if it was all cached.

TABLE 3. HTTP Request Characteristics

	HTML	Other	Total
Requests (million)	6.546	36.186	42.733
Unique Objects (million)	1.617	6.082	7.699
Unique Content Size (GB)	8.81	95.26	104.07
Total Content Transferred (GB)	29.59	412.47	442.06

4.2 Update Model

Given the cacheable objects identified from the log, we built two update models using the heavy tail distribution as discussed previously. The goal was to build a workload to compare protocols without introducing side effects that can favor one of them. We created two update models, one for HTML objects and one for all others. The models have an average modification period of one hour (3600s) for HTML objects and one day (86400s) for other objects.

We produced a normal modification workload ($a=10^4$) for the first distribution and a heavy workload ($a=10^3$) for the second (see Figure 3). We simulated the proto-

cols using each of the models. The following tables summarize the distribution of modifications.

TABLE 4. Update Distribution: normal

	HTML	Other
Period	1 hour	1 day
Modified Objects	543,693	83,163
Total Updates	2,270,084	217,509

For the normal distribution, 8.14% of the objects were modified during the period of 10 weeks with a global average update frequency of one object modified every 2.43 seconds.

TABLE 5. Update Distribution: heavy

	HTML	Other
Period	1 hour	1 day
Modified Objects	1,617,877	834,489
Total Updates	18,812,729	2,185,555

For the heavy distribution, 31.84% of the objects were modified during the period with a global average update frequency of one object modified every 0.32 seconds. Note that all HTML objects were modified at least once.

5 Simulation

The **`simulator`** collects statistics after a warm up period. During the warm up period many cache misses occur because the cache is empty. We warmed up the cache using the first week of the 10 weeks (10% of all requests).

This gave the **`simulator`** about two months of busy traffic during which to collect statistics. Table 6 shows a breakdown of the requests, validations and modifications during the simulation period.

TABLE 6. Simulation Results Workload

Statistics Period	9 weeks
Total HTTP Requests	42,227,728
Completed	42,224,590
Cacheable	38,971,702
Validation (304)	7,194,224
Total Content Transferred	417.26 GB
Modifications (normal)	4,488,916
Modifications (heavy)	42,515,677

5.1 Replacement Policy

We simulated the least recently used (LRU) replacement policy used by most of the current cache products, and two new policies described in detail in [2]. GDS-Frequency (GDSF) is a variant of the Greedy Dual-Size policy optimized for popular, smaller objects. LFU with Dynamic Aging (LFUDA) is a variant of LFU that uses a dynamic aging policy to accommodate shifts in the set of popular objects. We ran the simulation with the following cache sizes: 1, 2, 4, 8, 32, and 128 GB. A cache of 128 GB will hold all cachable objects (104 GB) and models an infinite cache for the simulation period. We ran the simulator with all combinations of replacement policy, consistency protocol, and cache size. We also simulated some special cases as described below.

5.2 Alex Protocol

The simulation of the Alex protocol used Squid default parameters (min=0, pct=20%, max=3days, ims=off). The simulator includes a sensor to measure inconsistent responses delivered from the cache (fast hits for objects that had been modified at the origin server).

We conducted an additional experiment to measure the impact of weaker consistency policies in the Alex/LRU protocol on consistency. In addition to the Squid default policies we simulated the following.

- Strong consistency (validate every time),
- Weaker consistency using parameters (min=1hour, pct=100%, max=1week, ims=off). We have seen Squid proxy caches configured this way.
- Weak consistency with the *weak validations* feature turned on (min=1hour, pct=100%, max=1week, ims=on).

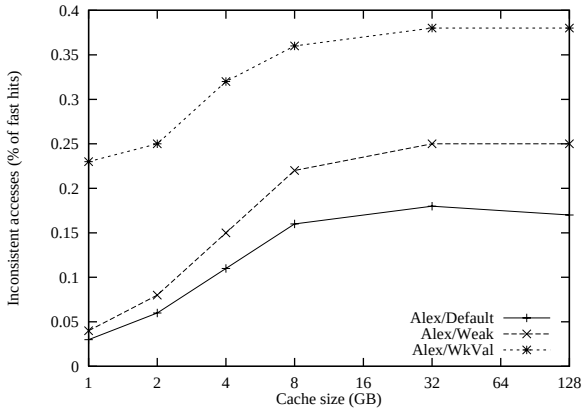
Table 7 shows the fast hit rate, bandwidth demand, and inconsistent hit rate when using an infinite cache, the normal workload, and each of the consistency levels.

TABLE 7. Summary Results by Refresh Policy

	Strong	Default	Weak	WkVal
Fast hits	0.00%	29.31%	30.05%	46.44%
Bandwidth (GB)	185.891	185.850	185.835	185.840
Requests (million)	42.224	29.850	29.537	22.617
Stale hits (% of fast hits)	0.00%	0.17%	0.25%	0.38%

Figure 4 shows the percent of fast hits that return inconsistent object data under the four consistency policies using the normal workload.

FIGURE 4. Inconsistency, normal workload



Relaxing the consistency level does not affect network bandwidth demand; it alters the ratio of fast and slow hits. Increasing the ratio of fast hits reduces the number of external requests made by the cache to validate object freshness and therefore improves average response time.

Inconsistent accesses under the normal workload occurred well less than 1% of the total fast hits. With the heavy workload inconsistent accesses occur about five times as often.

The remaining simulations use the default policy to achieve a good level of consistency to compare to the DOCP approach.

5.3 Distributed Object Consistency Protocol

For the simulation of the DOCP we explored how to configure and tune the subscription mechanism. A subscription has costs to the DOCP master to maintain state and send out the invalidation in case of modification. These costs can be prohibitive if the subscription mechanism is not well tuned. For example, a subscription becomes useless when the object is evicted by the DOCP slave's cache replacement policy.

To study this effect we developed sensors to measure subscription efficiency. The invalidation miss rate is the ratio of invalidations for objects that are no longer in the cache to total invalidations. We conducted experiments to understand the impact of various lease intervals on the invalidation and subscription miss rate.

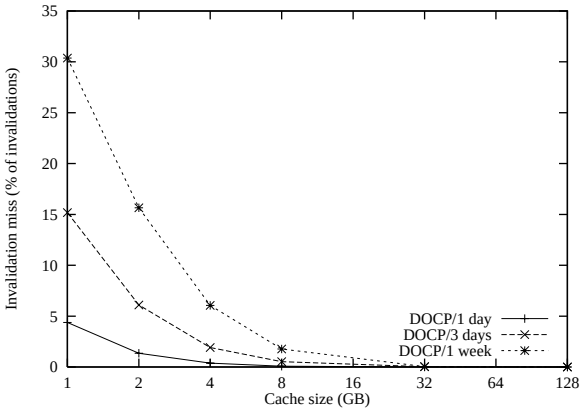
Table 8 shows the results of a simulation using the normal workload, a 4 GB cache, and the GDSF policy.

TABLE 8. Impact of Lease Interval on Invalidation Miss Rate

Lease	1 day	3 days	1 week
Fast hits	36.78%	41.78%	44.79%
Bandwidth (GB)	283.337	283.354	283.719
Requests (million)	26.696	24.582	23.312
Invalidation miss (% invalidations)	0.38%	1.92%	6.05%

Figure 5 shows the percentage of invalidation misses on subscribed objects for the three lease intervals using the normal workload, 4 GB cache, and the GDSF policy.

FIGURE 5. Invalidation Miss Rate by Lease



Extending the lease interval does not affect network bandwidth demand because renewals and validations do not carry much content relative to object data transfer. It alters the length of time a popular object will be served during a subscription. Increasing the duration allows the cache to serve more fast hits, and also increases the chance of eviction of a subscribed object. Sending invalidations increases DOCP master workload, but since they occur asynchronously to user requests they do not directly affect response time. A long lease also has an impact on server state since it increases the size of the subscriber list, but we did not study this in our simulation.

With an infinite cache the fast hit rates are slightly higher (about 1 percent each) and the invalidation miss rate is zero, since no subscribed objects are evicted.

The lease parameter does not significantly alter the number of subscribed objects, which depends primarily on object popularity. It does not significantly affect the network bandwidth demand.

We chose a value of three days for the lease period for the remainder of our simulation. This is the same value as the Alex **`max`** parameter, but there is a difference between the interpretation of the two values. The Alex refresh policy is set up by the cache administrator to optimize the client resources. The DOCP lease period is determined by the DOCP master administrator to optimize its resources. Both control an object's maximum stay in the cache without validation.

6 Results

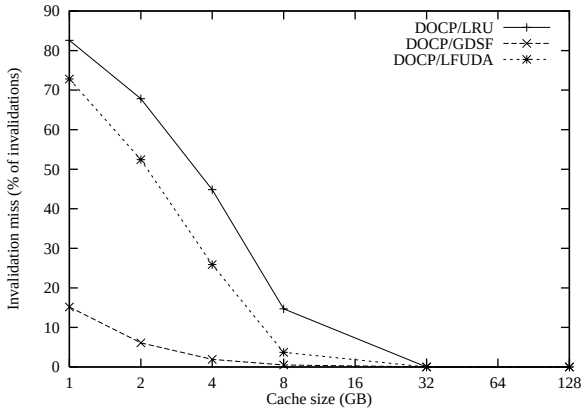
This section presents the overall results for the normal workload and examines the impact on the protocols when using the heavy modification workload.

6.1 Subscription Efficiency

In the previous section the invalidation miss rate was used under a single replacement policy to determine the

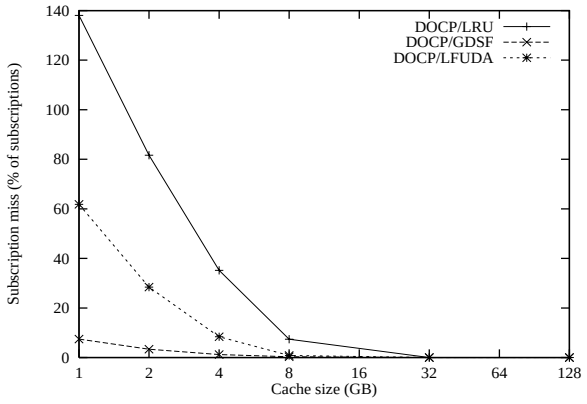
DOCP lease interval to be used for the remainder of the simulation. Figure 6 presents the subscription miss rate for all the replacement policies studied.

FIGURE 6. Invalidation Miss Rate



Another measure of subscription efficiency is the number of subscription requests a DOCP slave makes to a DOCP master for objects that the master considers the slave already to be subscribed. This is the subscription miss rate. Figure 7 presents the subscription miss rate under the normal workload and the three replacement policies.

FIGURE 7. Subscription Miss Rate



The cache replacement policy determines the objects that are left in cache when more space is needed. The GDSF policy attempts to maximize cache hit rate by keeping more objects in cache. Therefore it evicts large or unpopular objects. The LFUDA policy attempts to maximize cache byte hit rate by keeping more popular bytes in cache; it does not consider object size when making replacement decisions, only popularity. The LRU policy keeps recently referenced objects regardless of popularity. Since the subscription decision is based on object popularity the frequency-based policies achieve better invalidation and subscription miss rates.

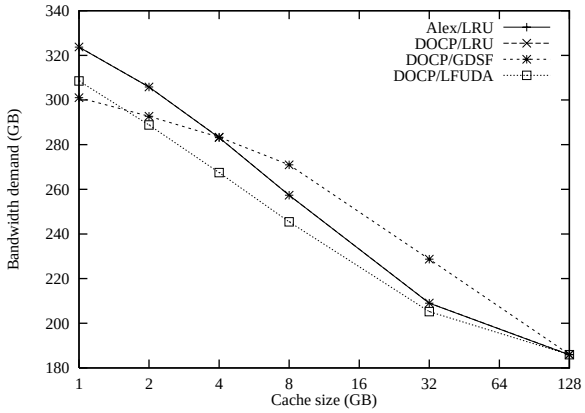
The following sections present the results of simulation of the DOCP protocol and the Alex protocol under three replacement policies and a range of cache sizes.

The consistency protocol does affect the hit rate or byte hit rate of a cache. Those factors are controlled by the replacement policy. The curves for Alex with the GDSF and LFUDA policies are left off the curves for readability. They line up exactly with the corresponding DOCP curve.

6.2 Hit Rate and External Bandwidth Demand

Figure 8 shows the external network bandwidth demand under simulated consistency protocol and replacement policies. External network bandwidth demand is the total number of bytes transferred between the cache and origin servers including the headers and the object data of all HTTP requests between the cache and origin servers. Validations and invalidations contain only headers.

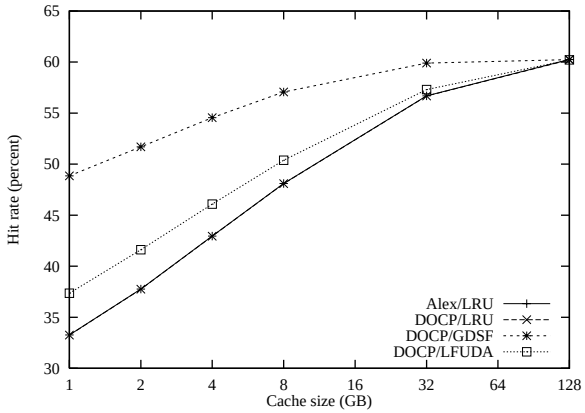
FIGURE 8. Bandwidth Demand



The total byte traffic without a cache would have been 417 GB. The curves above illustrate the number of bytes saved by the different configurations of consistency protocol and replacement policy.

Figure 9 shows the cache hit rate under the simulated consistency protocol and replacement policies.

FIGURE 9. Hit Rate



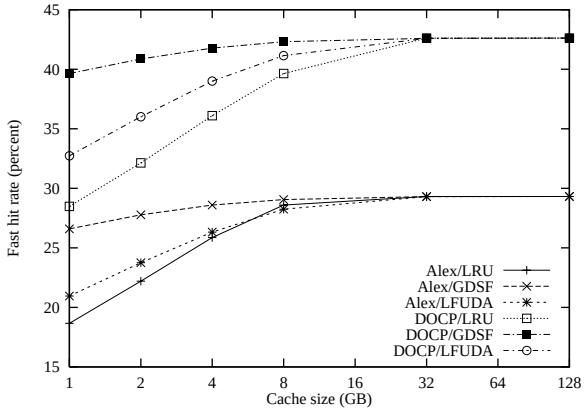
The consistency protocol does not influence network bandwidth demand nor cache hit rate. They are determined by the cache replacement policy. The consistency protocol adds validations and invalidations which do not

generate a significant amount of traffic compared to the data carried by regular requests. More detailed conclusions can be found in the study about these replacement policies [2].

6.3 Fast Hit Rate and External Requests

Figure 10 shows the fast hit rate at the DOCP slave cache; this is also the total number of external requests handled by origin servers. Those requests include object retrievals and validations made by the cache and DOCP invalidations sent by the DOCP master to the DOCP slave cache upon updates.

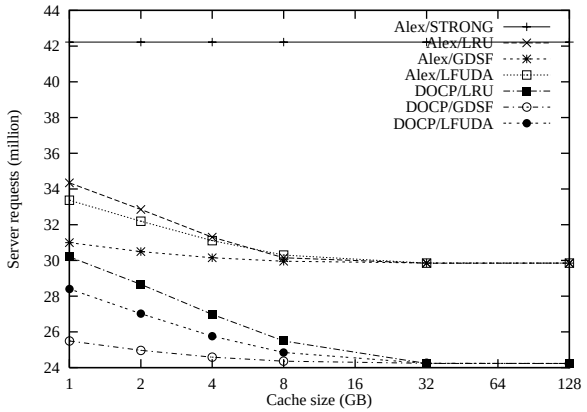
FIGURE 10. Fast Hit Rate



The DOCP improves the fast hit rate and reduces the number of total requests. Recall that when using polling to achieve strong consistency the fast hit rate is zero.

Figure 11 shows the aggregate server request demand. It includes the strong policy, which illustrates the benefit of strong consistency via invalidation.

FIGURE 11. Request Demand



The DOCP protocol outperforms the Alex protocol, and the GDSF policy outperforms the other policies. This is due in part to the synergy between the frequency based replacement policy and subscription mechanism.

6.4 Consistency

The response time (fast hit rate) and resource utilization (bandwidth, requests) between the normal and heavy workload were similar for both protocols. The Alex protocol implements weak consistency, which leads to potential access to a inconsistent objects. DOCP provides stronger consistency using invalidations when the objects are updated.

Table 9 presents a comparison of these results for the Alex and DOCP protocols with an infinite cache and the LRU policy.

TABLE 9. Consistency Results

Consistency	Normal	Heavy
Modifications	4,488,916	42,515,677
Inconsistent (Alex)	21,652	82,054
Percent of Fast Hits	0.17%	0.666%
Invalidations (DOCP)	44,247	164,264
Percent of Fast Hits	0.246%	0.918%

The heavy workload produced ten times more object updates than the normal workload. With DOCP, the cache received four times more invalidations on subscribed objects. For Alex the cache delivered four times more inconsistent objects.

7 Summary

Using invalidation for popular objects improves response time by increasing the number of fast hits. Increasing fast hits also reduces the number of external requests and the demand on origin web servers without sacrificing consistency. We simulated current technology, the Alex protocol with the LRU replacement policy, and configured it to achieve a good level of consistency. We simulated our new approaches DOCP/GDSF and DOCP/LFUDA, which provides assured consistency. We compared these technologies to the strong validation approach used to guarantee consistency. Table 10 summarizes observed results using a cache size of 4GB, which holds 4% of the cachable objects in our study.

TABLE 10. Result Summary - 4 GB Cache

	Strong/ LRU	Alex/ LRU	DOCP/ LFUDA	DOCP/ GDSF
Hit rate	42.95%	42.95%	46.09%	54.55%
Fast hit rate	0.00%	25.87%	39.00%	41.78%
Bandwidth (GB)	305.35	267.5	254.4	242.9
Requests (million)	42.224	31.301	25.757	24.583

We observed that the DOCP performs better than current protocols, while bringing object consistency to the Web. Furthermore, we point out a significant overall improvement due to the combination of the new consistency protocol and the new replacement policies.

7.1 Limitations

We present in this subsection some limitations of our simulation approach.

- Server subscriber list. We did not model the subscriber list of DOCP in our approach. This simulation focused on one proxy cache. The implication of this on protocol performance is negligible: the data sent over the wire is unchanged. The amount of server state required is important to quantify though.
- Lower bound of validations. The logs did not identify all validation requests to the proxy cache, only those with positive (slow) validation responses. There were some validation requests that resulted in consistency misses rather than positive validations. The simulator made fewer validations than occurred in practice, but the Alex and DOCP simulations were consistent with each other.

7.2 Suggestions for improvement

We have two suggestions to improve the simulation platform to better observe the behavior of DOCP.

- Optimize the generation of update events. We observed that the *update event* file size becomes a limiting factor to extend the simulation period or the modification load. It is useless to produce update events for objects accessed only once. 60% of the objects in this trace were requested only once. This optimization would be easy to implement in the `generator` and would ease the size constraint.
- Model multiple proxies. The simulation platform can be enhanced to model a more complete environment that will help understand DOCP behavior. The idea is to cluster client requests and consider each cluster of users served by a different DOCP slave. This will also permit modeling the server subscriber list.

8 Conclusions

Our goal was to evaluate the impact of stronger consistency on the Web. We built a simulator to validate and quantify our proposed Distributed Object Consistency Protocol for web proxy caches. Implementing this simulation helped us to explore the performance and behavior of current technology, and to compare and contrast our proposed protocol.

We found that the cache replacement policy drives most of the benefit regarding network bandwidth usage and cache hit rate. The consistency protocol does not significantly impact the network bandwidth usage unless the consistency level is significantly relaxed (as in the Alex with weak validation approach).

We believe consistency and latency will become more important factors to differentiate services among service providers, especially as we move towards electronic commerce over the web. One of the biggest challenges with this is to build a cache infrastructure that facilitates information distribution. Up to now critical content has not benefited from the caching infrastructure because the only alternative to control object consistency has been to bypass caching. The DOCP allows a content provider to have more control of the information distribution and to facilitate cache management with a real benefit for users because:

- A change to a popular object will be propagated by the network using best effort delivery. Inconsistent objects will not remain long in proxy caches.
- Content servers and proxy caches will not have to worry about setting expiration date. DOCP will manage that automatically based upon the request and modification streams.

Finally, those benefits come at the same cost as current technology.

9 References

[1] V. Almeida, A. Bestavros, M. Crovella, A. de Oliveira, “Characterizing Reference Locality in the WWW”, In IEEE International Conference in Parallel and Distributed Information Systems, December 1996.

[2] M. Arlitt, R. Friedrich, and T. Jin, “Performance Evaluation of Web Proxy Cache Replacement Policies”, to appear in Performance Evaluation, 1999. http://www.hpl.hp.com/techreports/98/HPL-98-97.html

[3] M. Arlitt, R. Friedrich, and T. Jin, “Workload Characterization of a Web Proxy in a Cable Modem Environment”, to appear in Performance Evaluation Review, August 1999. http://www.hpl.hp.com/techreports/1999/HPL-1999-48.html

[4] T. Berners-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0”, IETF RFC 1945, May 1996. The Internet Society.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, “On the Implications of Zipf’s Law for Web Caching”.

[6] V. Cate. “Alex -- A Global Filesystem”. In Proceedings of the USENIX File System Workshop, pages 1--12, May 1992. USENIX Association.

[7] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell. “A Hierarchical Internet Object Cache”. In Proceedings of the 1996 USENIX Annual Technical Conference, January 1996.

[8] C. Cunha, A. Bestavros, M. Crovella, “Characteristics of WWW Client-Based Traces”, Technical Report TR-95-010, Boston University, April 1995.

[9] J. Dilley, M. Arlitt, S. Perret, T. Jin, “The Distributed Object Consistency Protocol”, Technical Report HPL-1999-109, HP Laboratories, September 1999.

[10] J. Dilley, “The Effect of Consistency on Cache Response Latency”, Technical Report HPL-1999-107, HP Laboratories, September 1999.

[11] F. Douglis, A. Feldmann, B. Krishnamurthy and J. Mogul, Rate of Change and other Metrics: a Live Study of the World-Wide Web, AT&T Labs Research Technical Report #97.24.2, December 1997.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, IETF RFC 2616, June 1999. The Internet Society.