Reducing Latency Through Page-aware Management of Web Objects by Content Delivery Networks

Shankaranarayanan Puzhavakath Narayanan
Purdue University
spuzhava@purdue.edu

Yun Seong Nam
Purdue University
nam21@purdue.edu

Ashiwan Sivakumar
Purdue University
asivakum@purdue.edu

Balakrishnan Chandrasekaran
Duke University
balac@cs.duke.edu

Bruce Maggs
Duke University & Akamai Technologies
bmm@cs.duke.edu

Sanjay Rao
Purdue University
sanjay@purdue.edu

ABSTRACT

As popular web sites turn to content delivery networks (CDNs) for full-site delivery, there is an opportunity to improve the end-user experience by optimizing the delivery of entire web pages, rather than just individual objects. In particular, this paper explores page-structure-aware strategies for placing objects in CDN cache hierarchies. The key idea is that the objects in a web page that have the largest impact on page latency should be served out of the closest or fastest caches in the hierarchy. We present schemes for identifying these objects and develop mechanisms to ensure that they are served with higher priority by the CDN, while balancing traditional CDN concerns such as optimizing the delivery of popular objects and minimizing bandwidth costs. To establish a baseline for evaluating improvements in page latencies, we collect and analyze publicly visible HTTP headers that reveal the distribution of objects among the various levels of a major CDN's cache hierarchy. Through extensive experiments on 83 real-world web pages, we show that latency reductions of over *100 ms* can be obtained for 30% of the popular pages, with even larger reductions for the less popular pages. Using anonymized server logs provided by the CDN, we show the feasibility of reducing capacity and staleness misses of critical objects by 60% with minimal increase in overall miss rates, and bandwidth overheads of under 0.02%.

1 Introduction

Reducing the latency of web pages is critical for electronic commerce as it directly impacts user engagement and revenue [16, 18, 21]. Amazon, e.g., found that 100ms of latency costs 1% in sales [21], while Google Search found that a 400 millisecond delay resulted in a 0.59% reduction in searches per user [31].

SIGMETRICS '16, June 14-18, 2016, Antibes Juan-Les-Pins, France
© 2016 ACM. ISBN 978-1-4503-4266-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2896377.2901472

The quest to reduce web-page latencies has triggered much effort in the networking community among both researchers and practitioners. On the one hand, we have seen the large-scale adoption of widely distributed Content Delivery Networks (CDNs), that involve placing caches at thousands of Internet vantage points, close to end users. On the other hand, we have seen the recent emergence of new protocols such as SPDY [19] that significantly influenced the HTTP 2.0 standard. Despite these efforts, web-page latencies remain significant, constituting 80-90% of overall application response time by some reports [20, 31].

A key challenge in reducing the latencies of web-pages (the time to get an acceptable initial rendering of the page, formally defined in §3.1) is their complexity [10, 22]. Web pages are comprised of tens to hundreds of static and dynamic objects such as images, style-sheets (CSS), and JavaScript (JS) files, which may be served from multiple domains. Web-page download process has complex dependencies [27, 36], where some objects may have more impact on web-page latencies than others. The first objects fetched during a download (e.g., HTML, CSS, and JS) may need to be parsed or executed to decide which objects to fetch subsequently. Objects needed for an initial rendering of the page (e.g., to trigger a browser load event) may be more critical to the user experience than those that refine the initial rendering.

The need to accommodate the varying impact of individual objects on overall latencies has begun to receive attention from the community [12, 19]. Specifically, SPDY allows servers to transmit objects out of order to reflect their priority in the page load process. While useful in single server settings, most web pages today are served from multiple domains, and make extensive use of CDNs. Simply enabling SPDY between clients and CDN servers addresses only part of the problem. It is also necessary to reduce the CDN retrieval time of critical objects, especially since CDNs are typically organized as a hierarchy of caches [14] with different capacities and latencies at each layer.

Our motivation arises in part from the results of a study we conducted in which we collected end-to-end measurements of clients downloading pages from a number of web sites. The data shows that (i) objects appearing on the same web page are often served from multiple layers of the CDN cache hierarchy; (ii) critical objects are not always served from the fastest caches; and (iii) delays in serving a small number of critical objects can disproportionately impact overall latency.

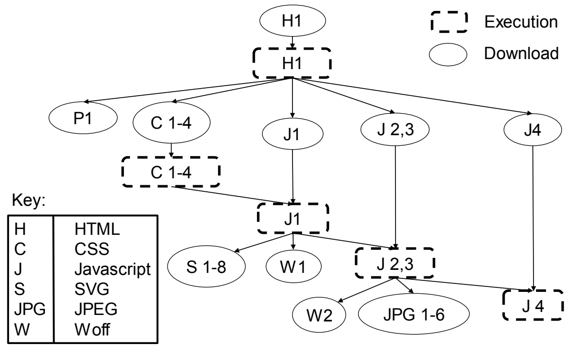
Motivated by these findings, we present a framework that allows CDNs to map objects more important for page latencies to faster cache layers. Our framework is enabled by the increasing shift of popular web-sites to CDNs for full-site delivery (e.g., for 89% of the pages in our study above, the main HTML document was served by the CDN). We consider a family of schemes for determining object priorities including a strategy based on content type, a strategy that prioritizes objects needed for an initial rendering of the page, and a scheme that explicitly takes the dependencies across objects of the page into account. We show how CDN cache placement and replacement algorithms may be redesigned to take object priorities into account, while still considering object popularity. We consider an approach where, to keep bandwidth overheads small, only objects most critical for latency are proactively refreshed to avoid staleness related misses. We present a family of schemes for proactive refreshing that differ in terms of which objects are refreshed.

We present an extensive evaluation study of our schemes using a combination of controlled experiments that emulate real web pages in hierarchical CDN settings, as well as trace data from a real CDN deployment. Our evaluations seek to understand the benefits of prioritization in CDN placement and refresh schemes, the relative benefits of different schemes for prioritization, and the sensitivity of our results to page popularity and composition.

Our evaluations with 83 real-world pages show that 30% of the most popular pages and 59% of the other pages show latency reduction larger than $100ms$, with some pages showing latency reductions as high as $500ms$. Both placement and proactive refreshing are important in achieving the benefits. For the vast majority of pages, considering content type in both placement and proactive refreshing provides most of the benefit. However, the additional benefits with other prioritization schemes can be significant in lower hit rate regimes, and when the penalty of going to the origin is higher. Finally, using trace driven simulations, we show the feasibility of the priority-based caching approach for reducing miss rates of page-critical objects in CDNs by 60% with minimal increase in overall miss rates. We also highlight the opportunity of minimizing stale misses for objects critical for latency by as much as 60% while incurring additional bandwidth costs of less than 0.02%.

2 Motivating measurement study

Web pages consist of tens to hundreds of objects of multiple content types (HTML, CSS, JS, images). A typical page load process involves significant dependencies across objects [27,36]. An initially downloaded HTML, CSS or JS (henceforth referred to as *HCJ*) object (which often embeds pointers to other objects) must be parsed (and executed) to identify further objects to download. Browser policies may dictate dependencies – e.g., execution of a JS must wait for a prior CSS to complete execution. Figure 1 shows an example dependency graph obtained using wprof [36]. Clearly, not all objects have the same impact on page latencies – e.g., $C1$ is much more important than $W1$. Further, content type need not necessarily reflect object importance – e.g., some *HCJ* objects may not be required for an initial rendering of a page most important for user experience, while Non-*HCJ* objects such as images may in fact be required. Moreover,



H	HTML
C	CSS
J	Javascript
S	SVG
JPG	JPEG
W	Woff

Figure 1: Dependency graph for a single load of www.apple.com. Each node shows download or execution of an object, and the directed arrow shows the dependency between them.

other objects may be dependent on Non-*HCJ* objects – e.g., a JS execution may wait on the arrival of a sprited image.

CDNs consist of a hierarchy of caches [14], typically consisting of clusters of servers deployed in multiple edge locations, and in parent locations. A user request that arrives at a server in an edge cluster (*First* server) could "hit" either at the memory or disk layer of that server. On a cache miss at the *First* server, requests could be directed to other servers in the CDN hierarchy (*Second* server), which could be a peer server in the same cluster or a server in a parent cluster. The latency of CDN served objects may vary widely depending on whether the object hits at the CDN and the layer that serves it.

To understand opportunities for reducing page latencies with CDNs by better mapping more important objects to faster CDN caches, we analyzed a prominent CDN that extensively provides edge caching (which we refer to as *CDN*). In the rest of this section, we discuss our measurement approach, and our findings.

2.1 Measurement methodology

We conduct end-to-end experiments by downloading real web pages from web clients and for each page measure the fraction of objects served from the different *CDN* layers. To determine the layer in the CDN hierarchy from which an object is served, we leverage HTTP pragma headers supported by CDNs for debugging purposes. Specifically, *CDN* supports the following pragma headers – *CDN*-x-cache-on, *CDN*-x-remote-cache-on and *CDN*-x-get-request-id. We set these pragma headers on all HTTP requests issued from the client. If the object is served by *CDN*, then the first contacted *CDN* server appends an X-Cache header in the HTTP response, and if a second *CDN* server is involved it appends an X-Cache-Remote header. The response also contains an X-*CDN*-Request-ID header with a dot-separated list of request IDs appended by each of the contacted *CDN* servers.

The X-Cache and X-Cache-Remote response headers contain values such as TCP MEM HIT, TCP HIT, TCP MISS, TCP REFRESH MISS, which respectively indicate a hit in the memory of that server, a disk hit, a server miss and a TTL expiry of a cached object with a new version fetched from the origin. We also count the number of request IDs in the X-*CDN*-Request-ID header to obtain the total number of *CDN* servers contacted. We use the values in these three headers to determine the layer from which the object was served. For instance, a TCP MEM HIT in the X-Cache header with one ID in the X-*CDN*-Request-ID header im-

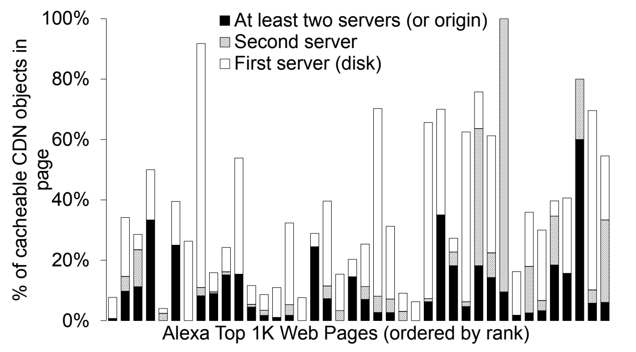


Figure 2: Breakdown of the different caching layers from which CDN-served objects of a web page are received. Fractions not shown hit at the memory layer of the first server

plies the object was served from the memory of the first *CDN* server. Like-wise a TCP MISS in the X-Cache header with a TCP MEM HIT in the X-Cache-Remote with two IDs implies the object was served from the second *CDN* server, while a MISS in both the headers means the object was fetched from origin. Note that, if we see a TCP MISS in both the headers with more than two IDs, due to limitations of the pragma headers, it is not possible to precisely tell if the object was served from the origin or another *CDN* server. But in our real runs we find these cases to be insignificant. For 90% of the pages fewer than 6 requests had more than two IDs in the X-*CDN*-Request-ID header.

We chose 100 Web pages for our measurement study across a wide range of popularity (Alexa US top sites [7]). Our measurement set has 40 pages in the Alexa rank top 1–1000 and 60 pages beyond rank 1000 which we refer to as *Top1K* and *Beyond1K* respectively in the rest of the paper. These pages were selected based on whether they had a good fraction of their objects served from *CDN*. Across all pages, at least 38% of the objects were served from *CDN* and for 25% of the pages more than 68% of the objects were served from *CDN*. Further, at least 92% of *CDN* served objects were cacheable for 90% of the pages. We also find that the main HTML for 89% of these pages were served from *CDN*.

Back-to-back downloads of the same page may artificially inflate the hit rates in subsequent runs owing to objects cached by the CDN from the earlier runs. To ensure our measurements themselves do not impact the hit rates, our entire set of measurements were spaced out across several weeks with consecutive downloads of the same page separated by 3 days – an analysis of the TTLs of objects in the pages indicated most objects would expire by this time.

2.2 Key findings

We now present key observations from our study.

• *Objects of a Web page may be served from different CDN caching layers incurring very different latencies:* Figure 2 presents the fraction of cacheable *CDN* objects served from each layer in the CDN hierarchy for a given run of the Alexa Top1K websites. Each stacked bar corresponds to a web page, and the segments in the bar show the breakdown – e.g., for the second most popular page (second stacked bar from the left), going from top to bottom, 19% of the cacheable *CDN* objects are served from the disk of the first server, 5% from the second server, 10% from at least 2 *CDN* servers (or origin) and the rest (66% – not shown) are served from the memory of the first server.

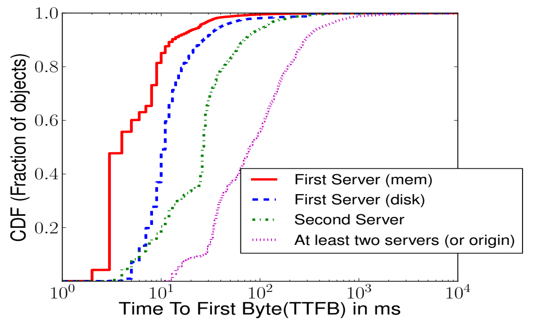


Figure 3: TTFB of objects served at different CDN layers

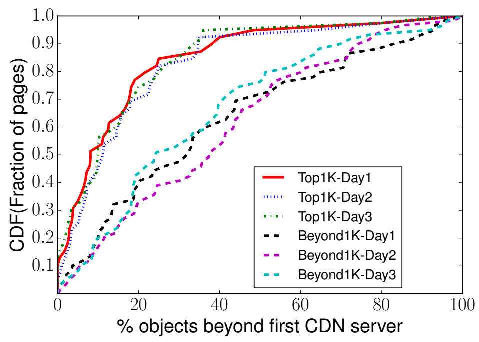


Figure 4: Fraction of cacheable CDN objects that are served from beyond the first CDN server across pages of two popularity classes for 3 different days

Figure 3 shows a distribution of the *Time To First Byte (TTFB)* of objects across all pages categorized by the layer from which it was served. The TTFB for an object is the time elapsed from when the request was sent from the client until the first byte of the response was received at the client, including the network time and retrieval time at the cache (or server). As expected, the TTFB observed across the different CDN layers vary substantially.

Figure 4 shows the fraction of cacheable CDN objects that are served from beyond the first contacted *CDN* server for the Top1K and Beyond1K classes for three different days. The figure shows that a significant fraction of objects are served from beyond the first *CDN* server even for the Top1K pages – e.g., the 50th(90th) %ile of objects served from beyond the first server were more than 11%(34%). Further, for the Beyond1K pages more objects are served from the farther layers in the CDN hierarchy – e.g., the 50th(90th) %ile of objects served from beyond the first server were more than 37%(74%). Moreover across multiple days, the hit rates remain similar for both the classes indicating the trends are consistent across many days and the hit rates are representative of the page popularity. We performed similar analysis across multiple days from different geographical locations and we find the trends to be similar. For instance, the median percentage of objects served from beyond the first server across the Top1K pages from another US location for three days were 15%, 12% and 10%, while for the Beyond1K pages they were 31%, 24% and 32%, with the 90%*ile* being higher than 38% and 71% across the days for the Top1K and Beyond1K respectively.

• *Critical objects are not always served from the fastest CDN layers:* Figure 5 shows a stacked bar graph with the number of objects served from each level in the dependency graph (as described earlier in this section) of *www.weather.com*. Each

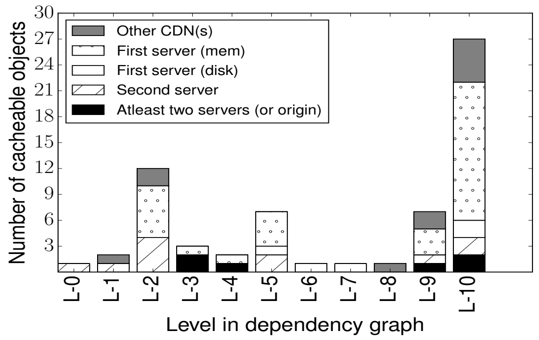


Figure 5: Number of objects from different caching layer at each level of the dependency graph for *www.weather.com*

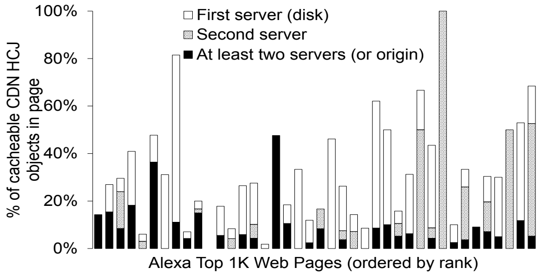


Figure 6: Breakdown of the different caching layers from which CDN-served *HCJ* objects (HTML, CSS, JS) are received.

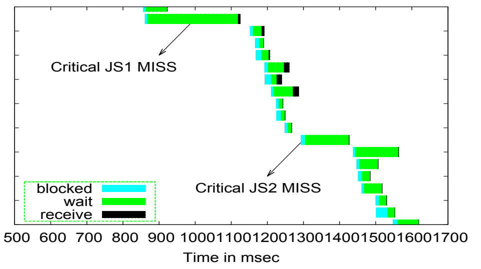


Figure 7: Serving delays in *HCJ* objects disproportionately impacts latency

bar corresponds to a level in the graph and the stacks in the bar show the number of objects served from the corresponding CDN layer. The figure shows that many critical objects (generally the internal nodes of the dependency graph) are served from beyond the first server indicating the potential to reduce page latencies by considering object priorities in CDN mechanisms.

Figure 6 shows a breakdown of the CDN layers from which the *HCJ* objects are served for the Alexa Top1K pages. While *HCJ* objects do not exactly correspond with objects important for a page load, we consider them here for simplicity. Note that a significant fraction of the *CDN* served cacheable objects are *HCJ* objects – e.g., 42% of *CDN* served cacheable objects are *HCJ* for 50% of the pages. The figure shows that in general across all pages a significant fraction of *HCJ* objects are served from different layers of the CDN hierarchy incurring vastly different latencies – e.g., for 50% of the Top1K pages more than 10% of the *CDN* served cacheable *HCJ* objects are served from higher layers with this fraction being greater than 48% for 10% of the pages.

• *Delays in serving a few critical objects can disproportionately impact page latency:* Figure 7 shows a section of the waterfall diagram which depicts how the objects arrive at a client during the download of an actual web page. The X-Axis is the time since the start of the download (only the relevant time segment is shown). Each bar corresponds to an object, and extends from when a request to that object was made, to when the object was fully downloaded at the client. Further, each bar shows the breakdown of the time spent waiting for a connection to the server (blocked), time spent waiting for the first byte of the response (wait) and time spent in receiving the object (receive). From the figure, we see that many objects were delayed because they depended on two JS objects, which got delayed (267ms, 135ms respectively with the time dominated by wait time). Further investigation showed both JS objects were cacheable and served from *CDN*, but were served beyond the first two servers in the hierarchy. Interestingly, many of the dependent objects hit in the CDN. Avoiding delays of these two JS objects would potentially reduce page load times by over 400ms (a 19% reduction).

• *Both true misses and stale misses contribute to objects being served from beyond the first* CDN *server:* We conducted a deeper analysis on the causes for the first *CDN* server misses. We found that even though true misses contribute greatly to the first server misses, we also find significant fractions of staleness related misses – e.g., the fraction of first server misses that were staleness related misses is more than 29% and 45% at the median and 75%*ile* respectively, with the rest being true misses.

3 Enabling page-awareness in CDNs

In §2, we have shown that there is opportunity to reducing web-page latency by mapping objects in a page most critical for latency to the fastest caches in the CDN hierarchy. In this section, we revisit CDN design to exploit this opportunity. In doing so, a number of issues must be tackled including (i) determining which objects to prioritize; (ii) reconciling the need to prioritize objects more critical for latency with the traditional CDN goals of placing more popular objects at the edge to save bandwidth, by appropriately tailoring cache placement and replacement policies; and (iii) avoiding staleness misses for important objects in addition to capacity misses. We discuss our schemes for each of these issues in the following sections.

3.1 Schemes for prioritization

We consider a range of schemes for assigning priorities to objects, which involve different trade-offs between the complexity of the priority-marking scheme, and the potential latency benefits as we discuss below:

• **Prioritizing objects based on content type: (*Type*)** Our first scheme is content-type based priority assignment, where objects are accorded priorities based on content type – specifically, HTML objects receive higher priority, followed by JS and CSS, and finally images and others. This, in fact, conforms to the best-practices for prioritization with the SPDY protocol, and is implemented by Chrome today [4].

• **Prioritizing objects needed for initial page rendering (*OLType*):** Content type may not accurately reflect the importance of an object to page latency. Objects needed for an initial acceptable rendering of a page are more critical to user experience than other objects. While images may be important for such an initial rendering, some *HCJ* objects may not be required. A commonly used indicator to identify such an initial version of the page is an Onload event triggered by the browser. The time to generate a browser Onload event, which we refer to as *Onload Time (OLT)*, is a commonly used metric to measure web page load performance [30]. This motivates our *OLType* strategy which prioritizes objects prior to the Onload event, and among such objects, prioritizes objects based on content type. More generically, this strategy could be refined to consider other indicators of an initial page load such as "above the fold" content, content related to most critical visual progress [2], or content with the highest utility to users [11].

• **Prioritization based on page dependency graph (*OLDep*):** While *Type* and *OLType* are fairly coarse-grained strategies, a more fine grained prioritization scheme is to consider the actual dependency graph associated with the page, and assign prioritizes based on the graph – e.g., in Figure 1, J1 would be accorded higher priority than J4. This motivates the *OLDep* algorithm. Like *OLType*, *OLDep* prioritizes objects needed to trigger the Onload event over other objects. However, among objects needed for the Onload event, it prioritizes objects based on their depth in the dependency graph, with *HCJ* objects preferred among those at the same depth. Likewise, objects after the Onload event are also prioritized based first on their depth in the dependency graph, and then their content-type.

3.2 Balancing popularity and priority in cache placement and replacement

CDNs have two potentially conflicting goals that they must consider in deciding whether to cache objects at a given edge location: (i) cost savings; and (ii) minimizing user latency. For cost savings, it is desirable to cache the most popular objects at the edge. However, for user latency savings, it is desirable to cache high priority objects at the edge. Since more popular objects might not be the highest priority and vice versa (e.g., page logos vs product related images), it is important to carefully reconcile these considerations.

A naive approach to tackling these issues is to use multiple LRU queues with one queue per priority level. When an eviction is required, incoming objects evict the least recently used objects in lower priority queues, before evicting the least recently used objects in the queue having the same priority as the incoming object. A key limitation of this approach is that objects with higher priority tend to remain in the cache even if they are no longer accessed, creating cache starvation for the popular, but low priority objects.

Instead, our approach is inspired by the notable Greedy-Dual-Size algorithm [13] which considers how to balance locality of access patterns with object size and the variable costs associated with fetching objects. We adapt this algorithm to balance object priority and popularity by assigning a utility to each object based on its priority (P_i), and implement our cache as a priority queue ordered by the utility of the objects. When an eviction is required, objects with the lowest utility value (which are located at the tail of the queue) are evicted first. To prevent high priority objects from residing permanently at the head of the queue, we gradually decrement the utility value of the objects in the queue that are no longer accessed. This may be achieved in a computationally efficient manner by maintaining a monotonically increasing global *clock* for the cache, which is added to the utility value of the object ($U(i)$) as follows:

$$U(i) = clock + 1 + (R-1) * \frac{P_{min} - P_i}{P_{min} - 1} \tag{1}$$

Here, P_{min} is the lowest assignable priority and is higher than P_{max}, the highest assignable priority. For simplicity, we fix $P_{max} = 1$ in our formulation and hence P_i varies between 1 and P_{min}. The parameter R is the ratio of the lowest and highest assignable priorities (P_{min}/P_{max}). A linear interpolation is used to assign the initial utility to objects with any priority. R is a knob that the CDN could tune to decide how much to favor hits to higher priority objects over lower priority objects, and we evaluate the impact of R with real traces in §6.1. The utility value of the object is updated using the above equation when the object is accessed from the cache. The monotonicity of the clock is maintained by incrementing the clock on an eviction to the utility value of the evicted object. Therefore, objects that are accessed more frequently will have a higher utility value than objects in the cache that do not see any further accesses. This ensures that high priority objects that are no longer accessed, eventually get evicted from the cache. Finally, note that an item is placed in the cache only if its utility exceeds the utility of the lowest utility object in the queue.

3.3 Priority based proactive refreshing

Staleness related misses could be avoided by proactively refreshing objects that are currently in the cache, but are about to expire in the near future, at the cost of some bandwidth related to unnecessary refreshes. Given the tradeoff between reducing staleness misses and the bandwidth penalty, it is desirable to only proactively refresh those objects most important for page latency. We consider a family of strategies which primarily differ in terms of which objects are proactively refreshed:

• ***HCJ*,** which only proactively refreshes *HCJ* objects. The primary advantage of the scheme is the simplicity in identifying objects to refresh.

• ***BO*,** which only proactively refreshes all objects required for the page Onload event. While the strategy has the potential to better mirror objects most important for latency, it is more involved to identify these objects.

• ***HCJ BO*,** which only proactively refreshes *HCJ* objects needed for the page load event. This strategy has the potential to reduce the bandwidth overheads compared to *HCJ* while matching its latency benefits.

For all schemes, a refresh is triggered only when a request for the object is received and the following conditions are satisfied: (i) the object is unlikely to receive further accesses until it expires; and (ii) the estimated number of accesses in its lifetime is sufficiently high to warrant a proactive refresh. Specifically, we require that

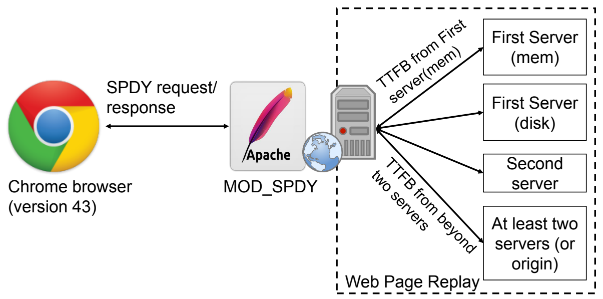
$$A_i * e_i \leq T_{P_i} \tag{2}$$

$$A_i * l_i \geq K_{P_i} \tag{3}$$

Here, A_i, l_i, e_i, and P_i are respectively the average request rate, lifetime, time left to expiry and priority of the object. A_i is computed by tracking the number of accesses seen by the object since it entered the cache, and l_i and e_i are obtained from the cache-TTL or expiry-time of the object. T_{P_i} is ideally kept smaller (close to 1) to trigger just-in-time refreshes. Note that larger T_{P_i} and smaller K_{P_i} support more aggressive refreshing. We evaluate the impact of these parameters with real traces in §6.2. These thresholds may be set differently across priority classes to support more aggressive or conservative refreshing for each class.

4 Evaluation Methodology

Our evaluations have two primary goals. First, we seek to understand the potential latency benefits of our various



SPDY request/ response
Chrome browser (version 43)
Apache
MOD_SPDY
TTFB from First server(mem)
TTFB from beyond two servers
First Server (mem)
First Server (disk)
Second server
At least two servers (or origin)
Web Page Replay

Figure 8: Experimental setup for evaluating latency benefits.

Placement schemes	Proactive refresh schemes
OBS	None
Type	*HCJ*
OLType	*BO*
OLDep	*HCJ BO*
	All

Table 1: Placement and refresh schemes studied in our evaluation.

schemes for priority-based placement and proactive refreshing, as well as their sensitivity to factors such as page popularity, CDN hit rates, page composition, and the relative latencies of various CDN cache layers. Second, we also seek to understand the impact of prioritization on CDN cache hit rates, and the bandwidth costs associated with the proactive refresh schemes.

We achieve the first goal by conducting a detailed emulation study of our schemes for the real-world pages analyzed in §2). Our emulations allow us to compare schemes in a fair manner while capturing the heterogeneity in latency (and object fetch times) across the CDN hierarchy, and realistic factors such as client execution times. We tackle the second goal by conducting a detailed analysis of traces from a real CDN. We present our experimental setup for latency comparisons in the rest of the section, latency benefits results in §5.1, and our trace-driven analysis in §6.

4.1 Methodology for latency comparisons

We use *Onload Time (OLT)* (defined in §3.1) to quantify the page load latency. We focus on OLT since it is objective, easy to determine, and widely used as a measure of page latency, while other indicators [2] are inherently more subjective. We do not consider the time to download the last byte for the page since many of the pages we evaluated tend to request objects indefinitely, and the time for an initial rendering is more important in practice.

We next discuss factors impacting our comparisons. A first factor is the CDN hit rates in terms of how many objects are served from each layer of the hierarchy. Of particular importance is the edge hit rate (*EHR,*) which we define as the fraction of objects served from the first CDN server (edge). A second factor is the composition of pages. Specifically, the fraction of *HCJ* and *BO* objects as well as the complexity of the dependency graph, can impact our comparisons. We compare our schemes with 83 real-world web pages analyzed in §2, which exhibit a wide range of diversity in terms of popularity and page composition. We highlight the characteristics of a page which impact the relative performance of schemes when appropriate. A third factor is the relative ratio of latency to the various layers of the CDN hierarchy, which we vary based on real measurements.

4.2 Schemes compared

Our schemes (Table 1) include:
Baseline for comparison (*OBS*): The *OBS* scheme corresponds to the placement observed when the page was loaded in the real-world measurements (§ 2). All objects served by *CDN* are fixed to the same layer from which it was served during the real page-load. All cacheable objects not served through *CDN* are split across the caching layers according to the hit-rates for cacheable objects observed in the real page-load.
Placement schemes: Our CDN placement schemes differ in their algorithms for assigning objects to the cache layers according to the fractions described above. We consider the *Type*, *OLType* and *OLDep* schemes are as described in § 3.1.
Proactive refresh strategies: This includes the *HCJ*, *BO* and *HCJ BO* schemes described in §3.3 which primarily differ in terms of which objects are proactively refreshed. We also consider the *None* and *All* strategies as baselines for comparison which indicate none or all of the objects are proactively refreshed.

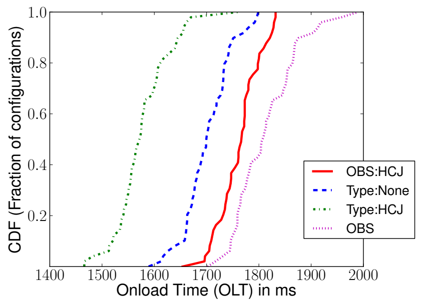
To ensure fair comparisons with *OBS*, for all schemes, all non-cacheable objects were always pinned to the farthest layer (origin). All objects that observed a staleness miss with *OBS* were served from the same layer as they were with *OBS* if they were not proactively refreshed. For example, with no proactive refresh all objects that saw refresh misses in *OBS* were pinned, while with the *HCJ* strategy non-*HCJ* objects with refresh misses were pinned. All remaining objects were assigned to the CDN cache layers as per the placement strategy. In doing so, the total number of objects served from each CDN layer was ensured to be (i) the same as *OBS* in the absence of proactive refreshing; and (ii) the same as *OBS* augmented with that proactive refreshing strategy otherwise.

If only a subset of objects of a given priority class (e.g., a subset of *HCJ* objects with *Type*) can be accommodated at a given CDN layer to satisfy the constraints on the number of objects that could be served from each layer, our schemes pick objects from within the priority class randomly. To ensure a robust comparison, we generate 50 different placement configurations with each scheme. We load each web-page with each scheme for each of its 50 configurations, alternating across schemes. We clear the browser cache between runs to eliminate the impact of local browser caching. We usually summarize the performance of a scheme for a given page by presenting the median OLT across the 50 configurations, but we also report on higher percentiles.

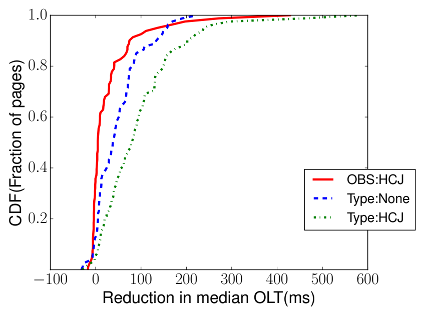
Many of our schemes require knowledge of objects needed for Onload and their dependencies. Rather than detailed activity dependence graphs [36] that may vary across runs, we obtain more static object level dependencies [27, 37]. Through multiple controlled experiments that each delay an object in a page, we determine the objects needed for Onload based on whether the Onload event is delayed. Likewise, dependent objects may be determined based on delays observed in their download times. We determine object importance based on its depth in the dependency graph rather than consider critical paths which may vary across runs.

4.3 Experimental setup

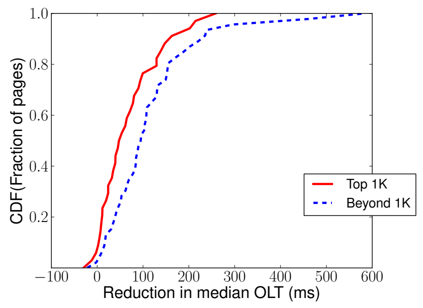
Figure 8 presents our experimental setup. Web pages are hosted on a web server (corresponds to an edge server in a CDN cluster), where TTFBs to the different caching layers



(a) OLT for *www.mercurynews.com* across schemes



(b) Median OLT reduction relative to *OBS* across all pages



(c) Median OLT reduction with *Type*:*HCJ* over *OBS* for Alexa Top1K and Beyond1K

Figure 9: Latency benefits of prioritized placement and proactive refresh strategies in isolation and combination.

are emulated and the page-load latency from an actual web browser is measured.

Web pages exhibit significant variability in the number of objects and aggregate download size for a given web page, even over short intervals of time. To ensure fair comparisons, we used an open source tool called web-page-replay [38]. Entire web pages including all constituent objects were first recorded through downloads from the actual web server(s). Then, the same recording was replayed for all schemes in later experiments. Some web pages still showed variability as they had JS that requested different URLs (e.g., using a random number or date) over different runs. We modified the web-page-replay code to replace such occurrences with constant values to ensure the same objects were requested for all schemes.

We focus our evaluations on settings where SPDY is enabled between the client and the edge server, in order to highlight that our benefits are complementary to SPDY. We also note that our schemes show similar benefits in the presence of traditional HTTP as well. We use apache mod_spdy server co-located with WPR, and Chrome browser (version 43.0) running SPDY (version 3.0) as the client in all our experiments. The client uses SPDY to forward object requests to the mod_spdy server, which in turn proxies the requests (and responses) to (and from) WPR. We modify the local DNS resolver configuration file in Linux to resolve all domains to localhost so that the requests are issued to the apache proxy during the replay experiments and no requests are served over the Internet. In all our experiments, the client uses the default priorities set by the SPDY implementation of Chrome when issuing requests to the server.

In order to minimize the impact of browser variability on page-load times, we disable all extensions, background and sync activities in the browser using Chrome command line flags [1]. We set the browser cache and user profile directories to RAMDisk [3] to minimize the impact of disk read/write variability on page-load times. We also clear the RAM disk across runs to ensure clean-slate page-loads where all objects are fetched from the emulated CDN layer.

5 Results

We begin by evaluating the potential benefits of our placement and proactive refresh schemes, focusing on content type prioritization (§5.1). We next compare all our placement strategies in the absence of proactive refresh (§5.2), and all our proactive refresh schemes in the absence of priority-based placements (§5.3), with a view to understanding the potential benefits of prioritizing objects based on factors other than content type. Finally, §5.4 evaluates the benefits when such richer prioritization is used as part of both placement and proactive refresh.

To emulate heterogeneous latencies associated with CDN cache hierarchies, by default, we use the median TTFB observed across all objects fetched from each layer in our measurement study (Figure 3), but we present a sensitivity study to our latency settings in §5.4.

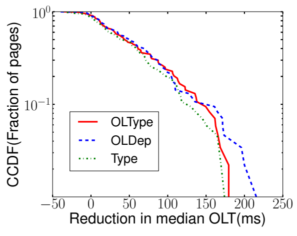
We compare the performance of the schemes with 83 real-world web pages (out of the 100 pages analyzed in §2). The remaining 17 pages either did not consistently trigger an Onload, or had 100% of its objects being served from the edge server(mem), in which case all placement schemes are equivalent. In comparisons where both placement and proactive refresh strategies vary, we use names such as *Type*:*HCJ* (indicating placement using the *Type* strategy and proactive refreshing of *HCJ* objects). When schemes compared vary in only in their placement (or refresh) strategy, we abbreviate by only using names of the placement (or refresh) schemes.

5.1 Latency benefits of prioritization

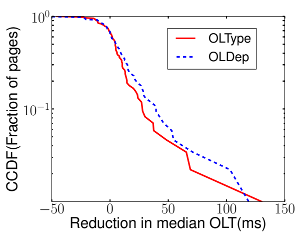
In this section, we evaluate the potential benefits of prioritized placement and proactive refresh, in isolation and in combination. We focus on schemes that primarily distinguish objects based on their content type (*Type* and *HCJ*).

Figure 9(a) shows a CDF of the OLTs observed (across 50 placement configurations as described in §4) with *OBS*, *OBS*:*HCJ* indicating *OBS* placement with *HCJ* refresh strategy, *Type*:*None*, and *Type*:*HCJ* for a popular web-page *www.mercurynews.com* (Alexa Rank:1245). The breakdown of objects served from different layers in the real download (*OBS*) for this page was 54% at the edge server (34%mem, 20%disk), 26%remote, and 20%origin. The figure shows that both priority based placement and proactive refresh independently help reduce the OLT when compared to *OBS*. However, the combination of the two provides significantly higher benefits, reducing the OLT by more than 200ms.

Figure 9(b) shows the reduction in median OLT achieved by the three schemes relative to *OBS* for all pages. The figure shows that *Type*:*HCJ* provides significant reductions in OLT over *OBS*, despite the high EHRs for some of these pages. For instance, we see a median OLT reduction of more than 100ms for 40% of the pages and more than 200ms for 10% of the pages. Our results also show the importance of priority based placement only (*Type*:*None*), which reduces the median OLT by more than 50ms for 30% of the pages. Interestingly, we also find *OBS*:*HCJ* strategy providing sig-



(a) Reduction relative to *OBS*.



(b) Reduction relative to *Type*.

Figure 10: CCDF of median OLT reduction with *Type*, *OLType* and *OLDep* for all pages with Y-Axis in log-scale. Proactive refresh is disabled for all schemes.

nificant latency reductions when compared to *OBS* by avoiding refresh misses for some *HCJ* objects. For example, for 15% of the pages *OBS*:*HCJ* provides a latency reduction of more than 68ms. These correspond to pages where $> 17\%$ of all objects were *HCJ* and saw refresh misses. We also found similar trends for the reduction in 90%*ile* latency for all the schemes, though the benefits were marginally higher.

To understand the impact of prioritization for pages with different popularities, in Figure 9(c) we show the OLT reduction with *Type*:*HCJ* over *OBS* split by the Alexa Top1K and Beyond1K classes (§ 2). The figure shows that though the benefits are more pronounced for the Beyond1K pages, prioritization provides significant reduction in Median OLT even for the Top1K pages. For example, though *Type*:*HCJ* provides 93ms (225ms) reduction in median OLT for 50%(10%) of the Beyond1K pages, we see benefits of over 157ms for 10% of the Top1K pages also. Overall, our results clearly emphasize the importance of prioritization through better placement and proactive refresh strategies.

To ensure statistical significance, we conducted a Mann-Whitney-Wilcoxon (MWW) test [28] (a non-parametric test to compare two populations) to reject the null hypothesis that the OLT distributions observed with the two schemes being compared are identical. Typically, the MWW test confirms significance (at a significance level of 0.05) when our schemes provide latency reduction over 10 ms. However the null hypothesis cannot be rejected when our schemes show more marginal reductions, or marginal increases. These cases usually correspond to scenarios where prioritization schemes provide no benefit – e.g., 33% of the pages see no *HCJ* refresh misses at all, and hence the *OBS*:*HCJ* scheme provides no intrinsic benefits over *OBS* for these pages. Interestingly, the latency increase over *OBS* is significant for 1 page with *Type*:*HCJ*, and for 2 pages with *Type*:*None*. On further examination, we found that the increase could be attributed to limitations of *Type*-based prioritization. These pages had some important Non-*HCJ* objects required for the initial page rendering (Onload), which were not placed in the edge by *Type*, but interestingly were served from the edge in the *OBS* run. For such cases, more sophisticated prioritization schemes like *OLType* or *OLDep* could potentially provide benefits.

5.2 Comparing placement strategies

We next evaluate the benefits of *OLType* and *OLDep* which consider factors besides content type in placement decisions. Since our focus is on placement schemes, our comparisons are done without proactive refresh.

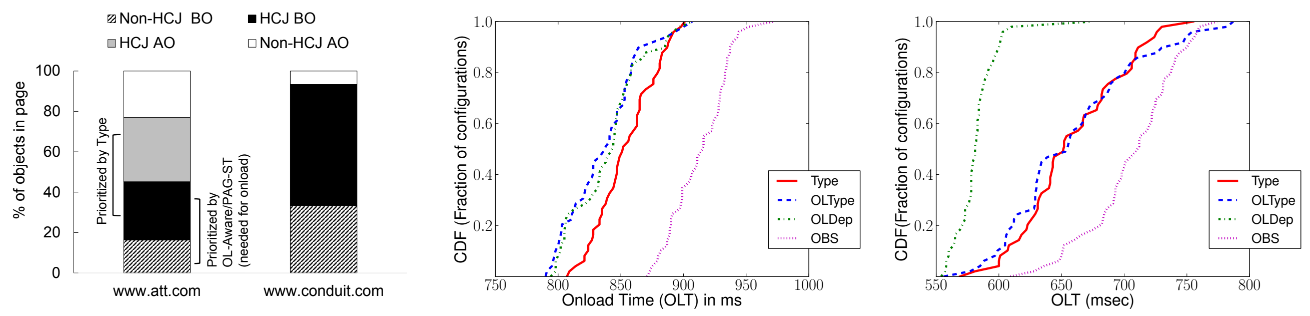
Figure 10(a) presents a CCDF of the median OLT reduction relative to *OBS* for all the placement schemes, and all the pages in our experiment set. While all schemes show significant reductions compared to *OBS*, *OLType* and *OLDep* provide only slightly higher benefits than *Type*. The benefits are marginal for most pages, however somewhat more significant at the tail. Note that the CCDF is shown with Y-Axis in log scale since the schemes show more prominent difference in the tail. Figure 10(b) shows the median OLT reduction of *OLType* and *OLDep* relative to *Type*. Across all pages, *OLDep* achieves a median OLT reduction higher than 35ms for 12% of the pages, while for 6% of the tail pages both schemes achieve median OLT reduction higher than 50ms relative to *Type*. Though we find higher benefits for the Beyond1K pages, we also see latency savings of more than 35ms for 14% of the Top1K pages as well, with as much as 71ms for the tail page. We have also verified the statistical significance of our results using the MWW test.

To better understand these results, and when different schemes are most helpful, we analyze the page composition into objects in 4 categories as follows – (i)*HCJ* and Non-*HCJ* objects; and (ii) required for Onload (*BO*) or not (*AO*). Figure 11(a) shows the composition of objects in these 4 categories for two example pages where *OLDep* and *OLType* show benefits over *Type*. In general, the benefits with these schemes depend both on the EHR, and the composition of the page, among other factors.

When do *OLType* and *OLDep* perform better than *Type*? Both *OLType* and *OLDep* prioritize objects before Onload. For *www.att.com*, the EHR is sufficiently high that all *BO* objects may be placed in the edge, hence both *OLType* and *OLDep* prioritize all *BO* objects to the edge. However, *Type* prioritizes *HCJ* objects (agnostic of whether they were required for Onload) over Non-*HCJ BO* objects, and is unable to accommodate all Non-*HCJ BO* objects at the edge. Indeed, Figure 11(b) confirms that for this page, *OLType* and *OLDep* perform better than *Type*. Note that the two schemes themselves perform comparably - this makes sense due to the high EHR both schemes are able to place all *BO* objects in the edge.

Interestingly, we observed many other pages where *Type* performs comparably to *OLType* and *OLDep* even though it is not able to place all Non-*HCJ BO* objects in the edge. On further analysis, we found that the Non-*HCJ* objects were leaves in the dependency graph for these pages – consequently, the performance with *Type* was relatively unaffected even though these objects were not placed in the edge. In contrast, for pages like *www.att.com* some of the Non-*HCJ* objects were internal nodes in the dependency graph, in the sense that a lot of object fetches were dependent on these objects (delaying the internal Non-*HCJ* objects delays a lot of other objects being fetched). For example, in *www.att.com* some of the Non-*HCJ* internal nodes were *sprited images* and the execution of JS code waits on these images. As a result, delaying these images delays all the objects to be fetched after executing the JS code. Thus careful placement of the internal Non-*HCJ* objects was particularly important to reduce the page latencies.

When does *OLDep* perform better than *OLType*? For *www.conduit.com*, 93% of the objects are required before Onload with more *HCJ BO* objects (60%) than can fit in the edge. While both *OLDep* and *OLType* prioritize *HCJ BO* objects, *OLDep* makes finer-grained distinctions, and prioritizes objects at the highest levels of the dependency graph, which are more critical for latency. Indeed, Figure 11(c) confirms that *OLDep* performs better than *OLType* for this



(a) Percentage of objects in each category (b) OLTs for *www.att.com* (EHR 84%) (c) OLTs for *www.conduit.com* (EHR 20%)

Figure 11: Impact of page composition on the relative performance of schemes

page. *OLType* and *Type* perform similar - this is because there are only a few *HCJ* AO objects and hence *OLType* and *Type* are choosing from relatively the same set of objects to place in the edge. More generally, *OLDep* also provides benefits over *OLType* for pages where all *HCJ BO* but not all *BO* objects fit in the edge, and some Non-*HCJ BO* objects are internal nodes in the dependency graph.

Finally, we found rare cases where *OLDep* performs worse than *Type*, when objects deeper in the dependency graph have more impact on the OLT than the objects closer to the root in the graph. For example, in *www.comcast.com*, a JS object (appearing as a leaf in the dependency graph) had a larger impact on the OLT owing to its higher execution times than other internal Non-*HCJ* objects, while *Type* placed all *HCJ* objects at the edge. While prioritizing objects occurring consistently on the critical path across runs (and clients) may help, determining such objects is not trivial, and we did not explore such a technique in depth given the small number of occurrences.

5.3 Comparing proactive refresh strategies

In this section, we fix the placement scheme as *OBS*, and compare the performance of various proactive refresh strategies described in §4. Since proactive refresh strategies differ only in their handling of refresh misses, we confine this study to only those pages(61 pages) that saw at least one stale access in the real page-load. Figure 12(a) shows that all strategies give significant latency reductions relative to *OBS*. For the vast majority of pages, the schemes perform similarly, though there are a small number of pages where *BO* and *All* perform better. Interestingly, we also find that the *HCJ BO* scheme performs similar to *HCJ* while incurring lesser bandwidth costs.

When does *BO* perform better than *HCJ*? We illustrate this using one of the pages *www.mercurynews.com*, where *BO* performs better than *HCJ*. Figure 12(b) shows the CDF of OLTs observed with each of the proactive refresh strategies. Clearly, all refresh schemes perform better than None, while *BO* performs even better than *HCJ* (and *HCJ BO*). We note that the page observed multiple refresh misses for Non-*HCJ BO* objects in the higher levels (L 9-14) of the dependency graph as shown in Figure 12(c). These objects have further dependent objects (in L15), and impacts the critical path of the page-load. Therefore, the *BO* scheme, which proactively refreshes all objects needed for Onload (including Non-*HCJ BO* objects), provides significant latency reductions.

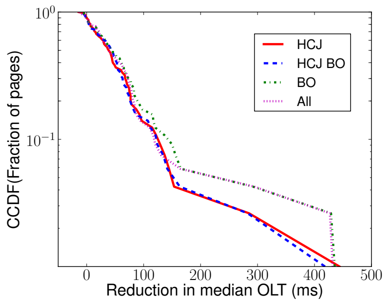
Overall our results show that (i) *HCJ* suffices in most cases, though *BO* can provide further latency reduction for some pages; (ii) *BO* itself gives all the benefits of *ALL* with lower bandwidth costs; and (iii) *HCJ BO* provides comparable benefits to *HCJ* with lower bandwidth costs.

5.4 Sensitivity to origin TTFB

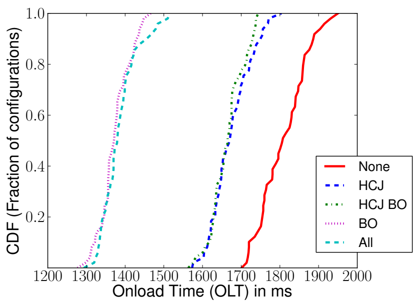
In this section, we achieve two goals. First, we evaluate the benefits achievable by considering factors other than content type in both placement and proactive refresh strategies together. Second, we study the sensitivity of the observed latency reductions with our schemes to heterogeneity in TTFBs for fetching objects. We focus on the TTFB to origin servers since they show the highest variability and have the highest impact on the page-load latency. Therefore, we conduct the sensitivity study by varying the TTFB to the highest layer retaining the same values for the other cache layers (used in §5). We compare *OLDep*:*BO* with *OBS* for three different ratios of *CDN* edge to origin server TTFBs viz. $1:4$, $1:8$ and $1:16$ (rounded-off) representing the 25th, median and 75th percentiles respectively from the real downloads (see §2). Note that the ratio used in all our prior experiments (§5) is $1:8$.

Figure 13 shows the reduction in median OLT with *OLDep*:*BO* compared to *OBS* split by pages in Alexa Top1K and Beyond1K classes. Clearly, *OLDep*:*BO* which combines both placement and proactive refresh provides significant benefits – with $1:8$, the median latency reduction is $> 100ms$ for 30% of the Top 1K pages and $> 100ms$ for 59% of the Beyond 1K pages. As expected, the benefits with prioritization increases with larger origin TTFB. Interestingly, the benefits are higher for both the Top1K and Beyond1K pages when the origin TTFB is high (ratio $1:16$). For instance, the median OLT reduction for 50% of the Beyond1K pages is about $2X$ higher with $1:16$ than $1:8$, while for 50% of the Top1K pages we see almost $3X$ higher reduction with $1:16$ than $1:8$. Though not shown, the reduction in median OLT with *OLDep*:*BO* was as much as $> 1s$ for $1:16$ (and $586ms$ for $1:8$) at the tail.

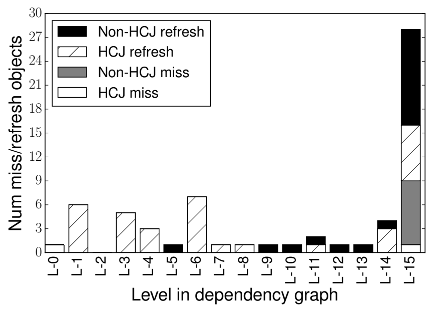
We now study the relative benefits of *OLDep*:*BO* over *Type*:*HCJ*, with the various edge to origin ratios. Figure 14 shows a CCDF of the reduction in median and $90\%ile$ OLTs with *OLDep*:*BO* over *Type*:*HCJ* for all pages. The figure shows that the latency benefits of *OLDep*:*BO* over *Type*:*HCJ* increases with higher ratios. The trends hold for reductions in both the median and $90\%ile$ OLTs, with reductions in the median OLTs as high as $700ms$ for *www.mercurynews.com* which had many Non-*HCJ* refresh misses. Overall our results show that *OLDep*:*BO* gives higher benefits over *Type*:*HCJ* for pages at the tail, especially when the TTFB to origin is high.



(a) CCDF of the median reduction in OLT relative to *OBS* with Y-Axis in log scale.



(b) OLTs for *www.mercurynews.com* with different refresh strategies.



(c) Composition of misses at each level of the dependency graph for *www.mercurynews.com*

Figure 12: Latency reduction with proactive refresh. All schemes use the *OBS* placement.

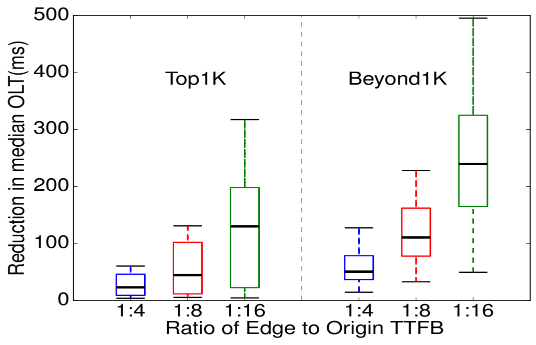


Figure 13: Median OLT reduction with *OLDep*:*BO* over *OBS* with three different edge to origin TTFB ratios split by Alexa Top1K and Beyond1K. The boxes show the 25th, 50th and 75th and the whiskers showing the 10th and 90th percentile pages.

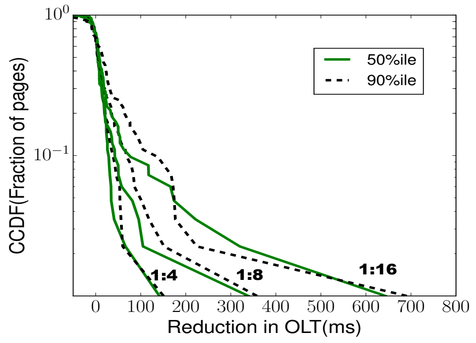


Figure 14: CCDF of the reduction in 50%*ile* and 90%*ile* OLTs for all pages with *OLDep*:*BO* over *Type*:*HCJ* for the three edge to origin TTFB ratios

6 Trace-driven evaluation

In this section, we conduct trace-driven simulations for evaluating the feasibility of priority-based caching and proactive refresh in CDNs. The request traces for this study were obtained from the edge cluster of a real CDN deployment, which serves a wide class of web traffic, and consists of 162 million requests for about 13.5 million distinct objects. The week long trace is non-sampled and consists of all client requests observed at each of the 18 servers in the edge cluster. For all simulations in this section, we set the cache capacity to those seen in the real deployment. Since the page structure (dependencies) and *Onload* information is not deducible from the trace, we use content-type based prioritization, and focus on miss-rate reduction (and bandwidth overhead) for all experiments in this section.

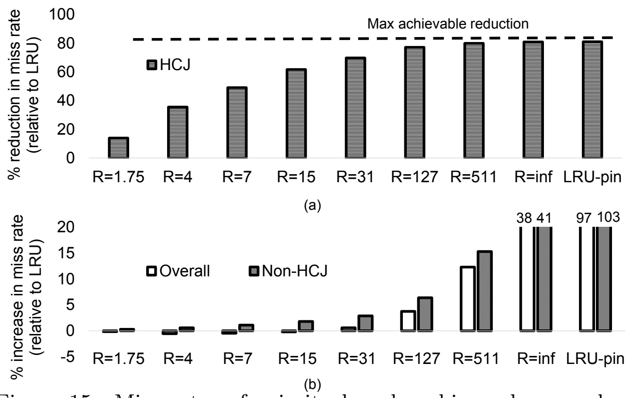


Figure 15: Miss rates of priority based caching schemes when compared to LRU(Size), and LRU-Pin.

6.1 Feasibility of priority based caching policy

We first show the feasibility of our approach in reducing the miss rate for the critical objects without significantly affecting the overall hit rates of the caches. We emulate the cache using our week-long trace for two caching algorithms - our priority-based caching described in §3.2, and (ii) LRU(size) - LRU with a size threshold, that is commonly employed by CDNs today. We evaluate our algorithm by varying the relative importance of the *HCJ* and Non-*HCJ* objects, which is captured by the parameter *R*, the ratio of the priority of *HCJ* objects to the priority of Non-*HCJ* objects. We also compare our algorithm with a variant of LRU which preferentially pins the *HCJ* objects to the cache and ensures that they are never evicted by a Non-*HCJ* object. However, *HCJ* objects may evict Non-*HCJ* objects and other *HCJ* objects similar to LRU. We use the same cache size and object-size threshold used by the LRU across all the schemes.

Figure 15(a) shows the reduction in miss rates for the *HCJ* objects for the different schemes relative to the miss rates observed with the LRU(size). The horizontal line at the top of the graph shows the maximum achievable reduction in miss rates, where the rest of the misses are compulsory misses in our trace. Figure 15(b) shows the corresponding increase in the overall miss rate as well as in the miss rate for Non-*HCJ* objects. From the figure, we see that as *R* increases, the reduction in miss rate for *HCJ* objects ramps up quickly for smaller *R* (flattens out at higher *R*), while the increase in miss rate for Non-*HCJ* objects increases gradually for smaller *R* and more rapidly for higher *R*. This shows the opportunity for the CDN to tune *R*, such that, it reduces

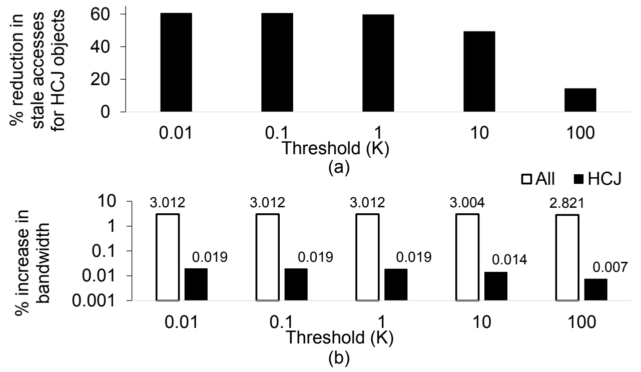


Figure 16: Impact of the *HCJ* and All proactive refresh strategies. Note that both schemes reduce stale accesses for HCJ objects by identical amounts.

the miss rate for *HCJ* objects without increasing the overall miss rate of the cache. In our trace, this occurs close to $R = 15$, which reduces the miss rate for *HCJ* objects by 61% without increasing the overall miss rate. We also see that while LRU-pin performs the best for *HCJ* objects, it drastically affects the overall miss rates. Our results show that our priority based caching scheme is able to significantly reduce the miss rates for *HCJ* objects, while incurring only a modest increase in the overall miss rate.

6.2 Bandwidth impact of proactive refresh schemes

We now show the benefits of prioritization in reducing the additional bandwidth costs associated with proactive refreshing. Since our traces do not have *Onload* information for objects, we focus our evaluation in this section on the *HCJ* and *All* proactive refresh schemes. We augment our priority-based caching algorithm with proactive refreshing as described in §3.3 and emulate the cache with our week long traces. In all our experiments, we set the threshold $T = 2$ (for just-in-time refreshes), but vary the parameter K for the *HCJ* objects to illustrate the bandwidth-cost and performance trade-off with conservative and aggressive proactive refreshing. Figure 16 compares the percentage reduction in stale accesses for *HCJ* objects, and the corresponding increase in bandwidth incurred with both the schemes. Note that the bandwidth costs estimated here are an upper bound since entire objects need not be fetched again if they are not modified at the origin, but that information is not available to us in the trace. The figure shows that both All and *HCJ* schemes reduce stale accesses for *HCJ* objects by 60%, while incurring an overall bandwidth increase of 3% and 0.02% respectively. Note that a smaller K (aggressive refreshing) results in fewer stale accesses, while a larger K (conservative refreshing) lowers the bandwidth costs of proactive refreshing. Overall, our results highlight the opportunity for priority-based proactive refresh in significantly reducing staleness for *HCJ* objects, while incurring only modest bandwidth penalties.

7 Related work

While SPDY [19] allows resource prioritization, it supports only priority based processing (and transmission) of objects from the server to a client [5]. Recent work [12] looks at re-prioritizing delivery of objects in a web page when they are pushed from a server to a mobile client. In contrast, our focus is on an orthogonal problem – enabling priority awareness within the CDN infrastructure. All our experiments including the *OBS* baseline are run with SPDY enabled, and our benefits are complementary to SPDY. Recent research has shown that SPDY is not always beneficial [17, 37]. Our proposals in this paper do not rely on SPDY - incorporating priority awareness in CDNs has benefits even with HTTP.

Our work builds on the rich literature on caching algorithms for web caches, proxy caches and CDNs (e.g., [6, 8, 9, 13, 24, 35, 39]). We adapt the well known Greedy-Dual-Size algorithm [13] which considers how to balance locality of access patterns with object size and the variable costs associated with fetching objects on a miss, given some network paths could be more expensive than others. Others have extended the algorithm to more explicitly bias it towards more popular objects [8, 24]). In contrast to all these works, our focus is on determining the importance of an object within a page for lowering page latency, and balancing object popularity and priority.

Prefetching to reduce web latencies has been extensively studied since the earliest days of the web (e.g., [29]). Many of the early works focused on client-side prefetching (e.g., [29]) in which clients initiate prefetching guided by predictions on which files are likely to be accessed soon (e.g., based on models that indicate which hyper-links a client is likely to click when on a given page [29]). Others [23, 25, 34, 40] have investigated prefetching in CDNs and proxy servers by using global access patterns to identify which objects should be proactively replicated to caches. While we leverage these techniques, we consider the more limited goal of avoiding refresh misses on objects already in the cache by proactively refreshing them. Further, we seek to proactively refresh objects that are more important for reducing page latencies, given refresh misses are a key component of overall miss rates for popular pages.

Researchers have explored how objects must be placed in a hierarchical caching system [15, 26, 32, 33] so that the average latencies are minimized given constraints on cache capacities [26] or bandwidth [33]. [32] propose mechanisms to improve end user response times by tracking the data location and minimizing the number of hops on hits and misses within the CDN hierarchy. In contrast, our focus is on placement of objects taking priority into account - specifically, objects that are not as popular may be placed lower in the hierarchy since they may be critical for page-load.

8 Conclusions

In this paper, we have made two contributions. First, we have shown that there is significant potential to reduce webpage latencies through page-structure-aware strategies for placing objects in CDN cache hierarchies. Second, we have presented several strategies to this end which differ in their degree of page-awareness, and conducted a detailed evaluation study of their benefits. Our evaluations with more than 80 real-world web pages show that for popular pages, more than 30% of pages see median OLT reductions higher than $100ms$, while for less popular pages, the median OLT reduction is more than $100ms$ for more than 59% of the pages, with some pages showing latency reductions as high as $500ms$. Both placement and proactive refreshing are important in achieving the benefits, though each can help in isolation. For the vast majority of pages, the *Type*:*HCJ* scheme provides most of the benefit. However, *OLDep*:*BO* can provide significant additional benefits for some pages, especially in lower hit rate regimes, when there are Non-

HCJ internal nodes in the dependency graph, and when the penalty of going to the origin is higher. Finally, using trace driven simulations, we show the feasibility of priority-based caching approach to reduce miss rates of page-critical objects in CDNs by 60% with minimal increase in overall miss rates. We also highlight the opportunity of minimizing staleness related misses for objects critical for latency by as much as 60% while incurring additional bandwidth costs of less than 0.02%.

9 Acknowledgements

We thank the anonymous reviewers for their constructive feedback and comments. We also thank Mangesh Kasbekar for his valuable inputs during discussions. This work was supported in part by the National Science Foundation (NSF) under Career Award No. 0953622, Award No. CCF-1535972, NSF Award Numbers 1162333, 1345284, and USAF Award FA8750-14-2-0150. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or USAF.

10 References

[1] Chrome command-line switches. http://goo.gl/7t5nk5.

[2] Google Speed-index. https://goo.gl/TKMJTw.

[3] RAMDisk for faster browsing. http://goo.gl/qtbDTe.

[4] SPDY best practices. http://goo.gl/fPczq3.

[5] SPDY Protocol(draft3)-stream-priority. http://goo.gl/5PH7UH.

[6] M. Abrams et al. Caching proxies: Limitations and potentials. 1995.

[7] Alexa. Available at http://www.alexa.com/topsites.

[8] M. Arlitt et al. Performance evaluation of web proxy cache replacement policies. Technical report, 1998.

[9] J.-C. Bolot et al. Performance engineering of the world wide web: Application to dimensioning and cache design. *Computer Networks and ISDN Systems*, 1996.

[10] M. Butkiewicz et al. Understanding website complexity: measurements, metrics, and implications. In *Proc. of the ACM IMC*, 2011.

[11] M. Butkiewicz et al. Enabling the transition to the mobile web with websieve. In *Proc. of the ACM HotMobile*, 2013.

[12] M. Butkiewicz et al. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *Proc. of the USENIX NSDI*, 2015.

[13] P. Cao et al. Cost-aware www proxy caching algorithms. In *Proc. of the USENIX USITS*, 1997.

[14] A. Chankhunthod et al. A hierarchical internet object cache. In *Proc. of the USENIX ATC*, 1996.

[15] H. Che et al. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 2002.

[16] J. Dean et al. The tail at scale. *Communications of the ACM*, 2013.

[17] J. Erman et al. Towards a SPDY'ier mobile web? In *Proc. of the ACM CoNEXT*, Dec 2013.

[18] B. Forrest. Bing and google agree: Slow pages lose users. http://goo.gl/BNjh3G, 2009.

[19] Google. SPDY: An experimental protocol for a faster web. http://goo.gl/vy63I4.

[20] J. Hamilton. The cost of latency. http://goo.gl/26j6S6, 2009.

[21] T. Hoff. Latency is everywhere and it costs you sales-how to crush it. http://goo.gl/T4vqjZ, 2009.

[22] S. Ihm et al. Towards understanding modern web traffic. In *Proc. of the ACM IMC*, 2011.

[23] Y. Jiang et al. Web prefetching: Costs, benefits and performance. In *Proc. of the 7th international WCW workshop. Boulder, Colorado*, 2002.

[24] S. Jin et al. Popularity-aware greedy dual-size web proxy caching algorithms. In *Proc. of the IEEE ICDCS*, 2000.

[25] R. Kokku et al. A non-interfering deployable web prefetching system. In *Proc. of the USENIX USITS*, 2002.

[26] M. R. Korupolu et al. Coordinated placement and replacement for large-scale distributed caches. *IEEE TKDE*, 2002.

[27] Z. Li et al. Webprophet: Automating performance prediction for web services. In *Proc. of the USENIX NSDI*, 2010.

[28] H. B. Mann et al. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 1947.

[29] V. N. Padmanabhan et al. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM CCR*, 1996.

[30] S. Souders. Onload event and post-onload requests. http://www.stevesouders.com/blog/2012/10/30/qa-nav-timing-and-post-onload-requests.

[31] S. Souders. Velocity and the bottom line. http://goo.gl/SaaVvv, 2009.

[32] R. Tewari et al. Design considerations for distributed caching on the internet. In *Proc. of the IEEE ICDCS*, 1999.

[33] A. Venkataramani et al. Bandwidth constrained placement in a wan. In *Proc. of the ACM Symposium on PODC*, 2001.

[34] A. Venkataramani et al. The potential costs and benefits of long-term prefetching for content distribution. *Computer Communications*, 2002.

[35] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM CCR*, 1999.

[36] X. S. Wang et al. Demystify page load performance with wprof. In *Proc. of the USENIX NSDI*, 2013.

[37] X. S. Wang et al. How Speedy is SPDY? In *Proc. of the USENIX NSDI*, April 2014.

[38] web-page replay. Record and play back web pages with simulated network conditions. https://www.code.google.com/p/web-page-replay/.

[39] R. P. Wooster et al. Proxy caching that estimates page load delays. *Computer Networks and ISDN Systems*, 1997.

[40] B. Wu et al. Objective-optimal algorithms for long-term web prefetching. *IEEE Transactions on Computers*, 2006.