Aging Through Cascaded Caches: Performance Issues in the Distribution of Web Content

Edith Cohen
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932, USA
edith@research.att.com

Haim Kaplan
School of Computer Science
Tel-Aviv University
Tel Aviv 69978, Israel
haimk@math.tau.ac.il

ABSTRACT

The Web is a distributed system, where data is stored and disseminated from both *origin* servers and *caches*. Origin servers provide the most up-to-date copy whereas caches store and serve copies that had been cached for a while. Origin servers do not maintain per-client state, and weak-consistency of cached copies is maintained by the origin server attaching to each copy an expiration time. Typically, the lifetime-duration of an object is fixed, and as a result, a copy fetched directly from its origin server has maximum time-to-live (TTL) whereas a copy obtained through a cache has a shorter TTL since its *age* (elapsed time since fetched from the origin) is deducted from its lifetime duration. Thus, a cache that is served from a cache would incur a higher miss-rate than a cache served from origin servers. Similarly, a high-level cache would receive more requests from the same client population than an origin server would have received. As Web caches are often served from other caches (e.g., proxy and reverse-proxy caches), age emerges as a performance factor. Guided by a formal model and analysis, we use different inter-request time distributions and trace-based simulations to explore the effect of age for different cache settings and configurations. We also evaluate the effectiveness of frequent pre-term refreshes by higher-level caches as a means to decrease client misses. Beyond Web content distribution, our conclusions generally apply to systems of caches deploying expiration-based consistency.

1. INTRODUCTION

Web objects are typically associated with one authority that can originate and modify them (their *authoritative* server), but can be cached and further distributed from multiple *replicating* servers. Indeed, caching and replication are widely deployed for reducing Web-servers load, network load, and user-perceived latency. Replicating servers are located in different points in the network and include reverse proxies, proxy caches, and browser caches.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

Replication necessitates some form of consistency, and to this end, the sole widely-supported protocol provides weak consistency in a client-driven expiration-based fashion: each cached copy has an expiration time, beyond which it must be validated or discarded. We briefly mention that proprietary strong consistency mechanism are deployed by Web hosting services and mirroring applications [1, 14], and various server-driven mechanisms, where servers notify subscribed clients when objects are modified, had been proposed. These protocols, however, are not widely supported or used.

The distribution of Web content is governed by the HTTP protocol [12]. Each object is identified by a URL which specifies its "location" and authoritative server. The object is requested by sending an HTTP request and a copy of the content is sent back on a corresponding HTTP response. The response includes headers with important information, including cache directives. The headers specify if the copy can be cached, and may provide an explicit expiration time or information that can be used to compute one. When an object is requested from the cache then if the cache has a fresh (non-expired) copy, the response is generated locally. If the cached copy has expired, it must be validated by contacting a server with a fresh copy. To this end, the HTTP protocol provides conditional (`If-Modified-Since` or `E-tag` based) `GET` requests. Similarly, if there is no cached copy, the cache must obtain a fresh copy.

Requests for which the cache does not have a fresh copy are termed *cache misses*. Cache misses require the cache to communicate with an external server before the response can be sent to the client. Remote communication, in turn, constitutes a significant portion of overall cache service times.

Our focus here are the effects of age on the miss-rate of the cache. The *age* of a cached copy is the elapsed time since it was first obtained from an authoritative server. When a cache receives an aged copy, the age is effectively subtracted from the freshness lifetime value which was provided by the authoritative source. Thus, a copy obtained through a cache expires earlier than a copy obtained from an authoritative source. Therefore, even caches that always have fresh copies are somewhat less effective than origin servers. A cache which forwards requests to a cache incurs a higher miss rate than a cache that forwards requests to the respective origin. We refer to the increase in the miss rate of a client cache using a replicating rather than authoritative source as the *age penalty.*

While extensive work concerned improving cache effectiveness (including document prefetching [18, 11], prefetching validations [17, 9, 10], and proactively refreshing selected content as it becomes stale [3, 15, 8]), age-effects had been by and large overlooked. Age becomes a performance factor for caches that function as either clients or sources to other caches, that is, when there is more than one cache placed between the end user and the origin server. Thus, age is becoming increasingly relevant with the deployment of caching hierarchies [16], and placement of caches both as proxy caches in local ISPs and reverse proxies closer to Web servers [22, 13]. Content delivery networks (CDNs) also function somewhat like caches [1, 21] and can induce age-penalty on client caches [7]. Age also impacts cooperative caches, the potential benefits of which were recently explored [23]. The study observed that caching hierarchies are not effective for very unpopular objects, since for them there is no value in sharing and there is increased latency of forwarding a request through a cache. Age was not considered by [23] but constitutes a complementary performance facet that also affects popular objects. Overall, the impact of age on performance seems not to be sufficiently recognized, understood, or addressed.

Our main contribution is introducing and studying age-related performance issues, and exploring the effectiveness of mechanisms that address them. Guided by Web practices and the HTTP protocol we model different cache configurations and behaviors of low-level and high-level caches. Specifically, we compare the performance of the following three configurations: 1) client-cache that uses origin servers, 2) client-cache that uses consistently the same higher-level cache per-object, and 3) client-cache that alternates between several higher-level caches. We show that even when all higher-level caches always serve fresh copies, there are definite performance gaps between the configurations. We also study these gaps for specific distributions, including Poisson, Pareto, and fixed-frequency arrivals, and finally, using trace-based simulations.

We then explore two practices: pre-term refreshes at higher-level caches and extended freshness lifetimes at client-caches. A cache performs a pre-term refresh when a request forces it to contact an origin server, even when it has a fresh copy. Configured pre-term refreshes can also be used as a mechanism to reduce the age penalty. Surprisingly, we show that pre-term refreshes can degrade performance of client caches. On the positive side, we show how to guarantee that no client-cache suffers performance losses. Moreover, we show that well-timed pre-term refreshes can be very effective in reducing the age penalty. Again, we demonstrate, via analysis and simulations, that these phenomena are fairly universal as they apply to large families of sequences.

Extended freshness lifetimes at client caches trades decreased miss-rate at the cost of decreased coherence (serving a higher fraction of outdated content). We demonstrate, however, that in some cases extending the freshness lifetime can decrease coherence without a corresponding decrease in miss-rate. We provide guidelines for more effective choices of the *extension factor* value (the factor of increase in the lifetime).

Section 2 provides an overview of the HTTP freshness control mechanism. Our model, which includes the basic cache configurations, is presented in Section 3. Section 4 discusses our choice of specific distributions and explains the methodology and data used for our trace-based simulations. Our results are presented in Sections 5–7, each containing a separate summary: Section 5 is concerned with the relation and gaps between basic source-configurations. Section 6 explores pre-term refreshes. Section 7 studies client-caches which use longer lifetime durations than their source. We conclude in Section 8.

2. HTTP FRESHNESS CONTROL

We provide a quick overview of the freshness control mechanism specified by HTTP and supported by compliant caches. For further details see [12, 2, 22]. Caches compute for each object a *time-to-live (TTL)* value during which it is considered *fresh* and beyond which it becomes *stale*. When a request arrives for a stale object, the cache must validate it before serving it, by communication either with an entity with a fresh copy (such as another cache) or with the origin server. The cachability and TTL computation are performed using directives and values found in the object's HTTP response headers.

Upon receiving a request for an object a cache acts as follows. If the object is not in the cache, the cache forwards the request to the origin server or another cache, and the request constitutes a *content miss*. If the object is cached then the cache checks whether its cached copy is fresh or stale. If the cached copy is fresh then the cache sends it to the client and the request constitutes a *hit*. If the cached copy is stale the cache issues a conditional HTTP `GET` request to the origin server (or to another cache). If the source response is `Not-Modified` then the cached object is sent to the client and we consider the request as a *freshness miss*. Otherwise, a new copy is obtained and sent to the client and we consider the request as a *content miss*. We use the general term *miss* for all requests that can not be processed locally, that is, content or freshness misses. It is important to note at this point that HTTP requests may be specified with a `no-cache` header. When a cache received such a request it must forward it to the origin server even if it has a fresh copy. The cache uses the response to replace or refresh its older copy of the object.

HTTP/1.1 specification considers every object as cachable unless an explicit no-cache directive is present (there are few exceptions, but they are not really used in practice). The TTL calculation for a cachable object, as specified by HTTP/1.1, compares the *age* of the object with its *freshness lifetime*. If the age is smaller than the freshness lifetime the object is considered fresh and otherwise it is considered stale. The TTL is the freshness lifetime minus the age (or zero if negative).

The *age* of an object is the difference between the current time (according to the cache's own clock) and the timestamp specified by the object's DATE response header (which indicates when the response was generated at the origin). If an AGE header is present, the age is taken to be the maximum of the above and what is implied by the AGE header.

Freshness lifetime calculation is done as follows. First, if a MAX-AGE directive is present, the value from this header is taken to be the freshness lifetime. Otherwise, if EXPIRES header (indicating absolute expiration time) is present, the freshness lifetime is the difference between the time specified by the EXPIRES header and the time specified by the DATE header (zero if this difference is negative). Thus, the TTL is the difference between the value of the EXPIRES header and the current time (as specified by the cache's clock). Otherwise, no explicit freshness lifetime is provided by the origin server and a heuristic is used: The freshness lifetime is assigned to be a fraction (HTTP/1.1 mentions 10% as an example) of the time difference between the timestamp at the DATE header and the time specified by the LAST-MODIFIED header, subject to a maximum allowed value (usually 24 hours, since HTTP/1.1 requires that the cache must attach a warning if heuristic expiration is used and the object's age exceeds 24 hours).[1]

We distinguish between different *freshness control mechanisms* by the type of the headers used for calculating the freshness lifetime. There are basically four different mechanisms: (1) Using MAX-AGE header. (2) Using an EXPIRES header such that it is set in a relative way. I.e. the difference between the values of EXPIRES and DATE is fixed for the object. (3) Using an EXPIRES header set to some absolute point in the future. (4) Using a heuristic based on the DATE and LAST-MODIFIED headers.

We estimated the usage frequency of different freshness control mechanisms as follows. We downloaded a 6 day log of the UC NLANR cache [16]. For a random sample of URLs in the log we carried out a GET request and by analyzing the headers in the response we deduced which freshness control mechanism is being used. For each URL we weighted the freshness control mechanism it uses by the number of requests to the URL in the log, and we summed the weights for each freshness control mechanism. We found that 3.4% of the requests were to objects with MAX-AGE specified, 1.4% were to objects with no MAX-AGE header but with EXPIRES set in a relative way (or to a time no greater than the one specified by the DATE header), and 0.8% were with EXPIRES specified in an absolute way. The vast majority, 70% of the requests, did not have either MAX-AGE or EXPIRES specified but had a LAST-MODIFIED header which allowed for a heuristic calculation of freshness lifetime. Other requests either had neither of these 3 header fields, were explicit noncachables (3%), or corresponded to objects with HTTP response status codes other than 200 (On separate GET requests), the most common of which was 302 (HTTP redirect).

A CDF of TTL values of objects in the log weighted by respective number of client requests [7] shows a dominant (60%) TTL duration of 24 hours, which is mostly due to the heuristic calculation (using LAST-MODIFIED) with a maximum setting of 24 hours. About 25% of TTLs are 0 (due to MAX-AGE or EXPIRES directive). These statistics also show that most TTLs are in fact *fixed*, where the freshness lifetime is the same for subsequent checks with the origin.

Figure 1 plots the TTL of an object with MAX-AGE or "relative" EXPIRES freshness control mechanism when fetched either from an origin server (the flat line) or from a top-level cache. The illustrated top-level cache refreshes its copy only when the copy is stale and the figure reflects three refreshes (where the TTL goes up to the MAX-AGE value). The zero-TTL region corresponds to a time period for which the cached copy is stale and no requests are received.

A *Pre-term refresh* by a cache is a refresh of a non-expired copy of an object. A pre-term refresh can occur when the cache receives a client request with a *no-cache* request header. With pre-term refreshes, the TTL of an object as in Figure 1 would look as illustrated in Figure 2.

[1]Squid calls these two constants CONF_PERCENT and CONF_MAX [22]. The TTL is $\min\{\text{CONF\_PERCENT} * (\text{DATE} - (\text{LAST-MODIFIED})), \text{CONF\_MAX}\}$.

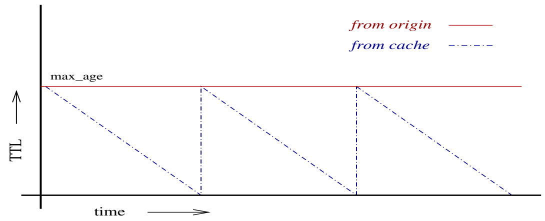


Figure 1: TTL of an object with MAX-AGE response header (i) when fetched from the origin and (ii) when fetched from a cache.

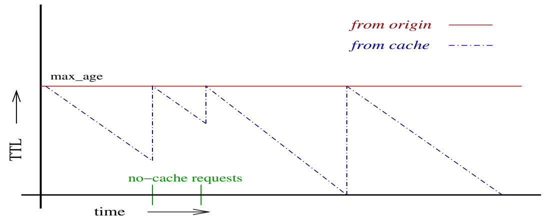


Figure 2: TTL of an object with MAX-AGE response header (i) when fetched from the origin and (ii) when fetched from a cache. Two client requests with a `no-cache` directive are illustrated.

3. MODELING BASIC TYPES OF SOURCES

We present a model for the distribution of copies of objects. Our setup is a collection of distributed servers that cache, request from each other, and distribute copies of an object. We distinguish between *origin* (authoritative) servers, that originate, continuously host the object, and may modify it over time, and *caches* (*replicating* servers) that may hold cached copies of the object. Following the HTTP protocol our model assumes a TTL based weak consistency mechanism. Each copy has a *freshness lifetime duration* which is set when the copy originates (is obtained from an authoritative server). The copy can be further distributed and used only for the length of its lifetime duration. Much of our analysis focuses on values that are fixed, that is, each time a copy originates its freshness lifetime is set to the same value which we denote by T. The *age* of a copy is the elapsed time since it originated. The TTL of a copy equals the lifetime T minus its age. If the age is larger than T, the copy is *stale*. Otherwise, the copy is *fresh*. Thus, a copy obtained from an origin server has initial age of zero and TTL of T whereas a copy obtained from a cache would generally have a positive age. A cache can distribute its copy only if it is fresh. Otherwise, it may attempt to obtain a fresh copy from an authoritative server or another cache. Thus, two caches can have a source-client relationship.

In the Web context, authoritative server can be in principle any server that returns copies of zero age (TTL that equals the freshness lifetime). This includes origin servers, mirrors, and sometimes CDN servers.

A client-cache can fill requests for objects for which it has stale or missing copies by contacting another cache, a selected cache from a set, or an origin server. In practice, the destination may be forced on a cache through a transparent configuration or may be selectable. We use the term *source* for the destination entity to which the client cache sends requests. We consider several types of sources and look at how the type of the source affects the miss rate at the client cache. The base line for comparison would be the miss rate when the source is authoritative. Let m_s be the miss rate of the client cache when it works through a source s, and m_a the miss rate of the client cache through an authoritative source. The *age penalty* of source s is defined as the relative increase in miss rate with respect to the authoritative source, i.e. the age penalty of source s is equal to $(m_s - m_a)/m_a$.

Our modeled sources are such that objects remain cached until they expire. Hence, the behavior of the system on different objects is independent and it is sufficient to analyze requests for each object separately. Since we focus on age-induced effects, our modeled sources always have a fresh cached copy of the object. Cached copies, however, may be aged.

We say that a client-cache is *synchronized* with a source if whenever the client cache contains a copy of the object which expires at some time t, then requests directed to the source at times $t + \Delta$ ($\Delta > 0$) obtain an object with age at most Δ. By definition, a client-cache is always synchronized with an authoritative source. Synchronization does not generally occur with sources consisting of caches but it does hold when the source is a single cache which never performs pre-term refreshes. As we shall see, synchronization helps performance.

3.1 Demonstrating the effects of age

We motivate some issues through an illustrative example. Consider a proxy server, `low-cache`, a heavily used reverse proxy cache, `top-cache`, and an origin server, `www.s.com`, containing the object `www.s.com/obj1`. We assume that if `low-cache` fetches `www.s.com/obj1` from `www.s.com` then the TTL of `obj1` is T. If `low-cache` fetches the object from `top-cache` we assume that the age of the object is the time passed since it was last fetched by `top-cache` (Here we neglect the transmission time from `top-cache` to `low-cache`.) As mentioned before, we assume that `obj1` is requested very frequently at `top-cache` so `top-cache` always has a fresh copy of it. We also assume that `obj1` is never evicted from `low-cache`.

Consider the case where `obj1` is requested at `low-cache` with inter-request times of $T/2$. If directing requests to the origin server `www.s.com`, `low-cache` would incur miss rate of $1/3$ (every third request would be a miss). If all requests are directed to `top-cache` then the miss rate is $1/2$. (Except for the case when `top-cache` happens to be in-sync with the stream of requests at `low-cache` and refreshes `obj1` exactly every third request from `low-cache`. We assume that this case occurs with probability 0.) Hence, the age penalty is $1/2$, or in other words, if `low-cache` always fetches `obj1` from `top-cache` there is a 50% increase in misses compare to the case where `low-cache` communicates directly with `www.s.com`. Denote the estimated response time of the parent cache by d_c and that of the origin server by d_o (we assume $d_c < d_0$). If our performance metric was the sum of induced latencies, then it is worthwhile directing requests of `www.s.com/obj1` to `top-cache` only if $1.5d_c < d_o$ (that is, if the estimated response time of `top-cache` is less than $2/3$ of that of `www.s.com`).

We now demonstrate how diminished synchronization can exacerbate the age penalty effect. Assume now that `low-cache` forwards requests to different independent parent caches each equivalent to `top-cache`. As with a single parent also in this case a hit at `low-cache` is always followed by a miss. However a miss is not necessarily followed by a hit. The age and therefore also the TTL of `obj1` which is fetched as a result of a miss is distributed uniformly between 0 and T. Therefore a miss is followed by a hit only with probability $1/2$. Let X be a random variable that counts the number of consecutive misses prior to a hit. It is easy to see that $X - 1$ is distributed geometrically with $p = q = 1/2$, so the expectation of X is 1. We obtain that on average there are two misses prior to a hit so the miss rate is $2/3$. Thus, in this example, lack of synchronization amount to a $1/3$ increase in the miss rate compared to the case of using just one `top-cache`. (from $1/2$ to $2/3$), and the age penalty is 1 (increase from $1/3$ to $2/3$ compared to an authoritative source).

3.2 Source types

To further explore these issues, we formally define the three types of sources illustrated in the example above (see Figure 3). The TTL value obtained through each source as

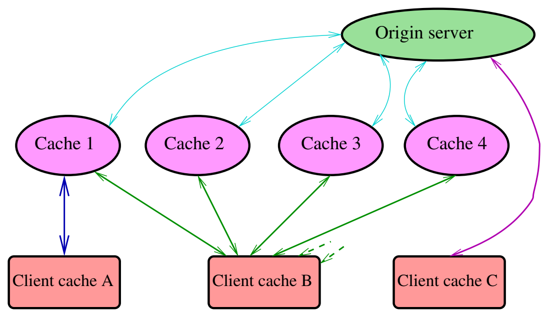


Figure 3: Different types of sources: Cache A uses cache 1 as an EXC source. Cache B uses caches 1,2,... as an IND source. Cache C uses an AUTH source.

a function of time is illustrated in Figure 4.

1. Authoritative (AUTH): Always provides a copy with zero age (TTL that equals the freshness lifetime T).

2. Exclusive replicating server (EXC): The client-cache consistently sends requests to the **same** replicating server. We assume that the replicating server always has a fresh cached copy which is refreshed each time that it expires. The replicating server refreshes the copy by contacting an AUTH source. Therefore, when a client cache sends a request at time t the TTL of the copy it obtains is $T - (t + \alpha) \bmod T^2$, where T is the freshness lifetime of the object. The parameter α represents the displacement between the request sequence at the client cache and the "refresh cycle" of the particular replicating server. Since we are interested in modeling the situation where the request sequence at the client cache is independent of the "refresh cycle" of the particular replicating server we assume that $\alpha \in U[0, T]$ (selected once for the whole sequence from the uniform distribution on $[0, T]$). Thus, when analyzing performance through an EXC source we consider the expected miss-rate over this random choice of α.

[2] "mod" is a generalized modulo operation to arbitrary non-negative numbers $a \bmod b = a - b * \lfloor a/b \rfloor$. If $b = 0$ then we define $a \bmod b \equiv 0$.

3. Independent replicating servers (IND): Upon each miss, the client cache forwards the request to a randomly-selected replicating server that always keeps a fresh copy as described above. The replicating servers for different requests are chosen independently so the displacement α in this case is chosen uniformly at random per request (rather than once per the whole sequence). Equivalent we can say that the age (and TTL) which the client cache obtains upon a miss through an IND source is independently drawn from $U[0, T]$.

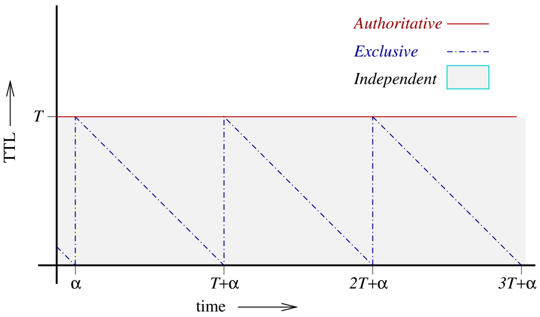


Figure 4: TTL obtained through different types of sources. AUTH source always provides zero age and TTL of T. EXC source (shown with displacement α) provides age that cycles from 0 to T and thus TTL which cycles from T to 0. IND source provides at each point in time age drawn independently from $U[0, T]$ and thus TTL drawn from $U[0, T]$.

The EXC and IND sources are basic models which abstract different performance factors of actual caches. The Exclusive model captures a prototypical scenario of Web caches directing requests to the same source cache (e.g., a transparent proxy, a configured proxy, or a reverse proxy). Actual Web caches, however, perform sporadic pre-term refreshes as a result of requests with `no-cache` request headers (see Section 2). Pre-term refreshes are not captured by the pure Exclusive model and break synchronization between source and client caches. The effect of pre-term refreshes is studied in Section 6 which extends the pure Exclusive model. The Independent model captures the use of several different caches for load balancing purposes. Real sources seem to be hybrids of IND and EXC. Which of our models is better for a particular real source depends on the number of higher-level caches that are used for the same object, and the way one of these caches is selected when the client cache incurs a miss. Last, we remark that high-level Web caches suffer cache misses, which are propagated to an origin server. These well-explored misses are a central performance metric for the effectiveness of top-level caches, and occur in the "channel" between high-level caches and origin servers. Our focus here is performance in the "channel" between high- and low-level caches, and thus, our models factor out propagated misses by assuming constant availability of fresh copies at high-level caches.

4. DATA AND METHODOLOGY

We discuss some qualitative phenomena which apply to all or large families of request sequences. We then characterize the extent and presence of these patterns for specific distributions and using trace-based simulations.

4.1 Speci£c distributions

We analyze sequences generated by a random process where inter-request times are independent and drawn from some distribution. We consider Poisson, Pareto, and fixed-frequency arrivals. Poisson and Pareto sequences have *reference locality*, which means that the next request is more likely to occur sooner. More formally, locality means that the probability density function of the inter-request duration is non-increasing.

Poisson arrivals. Poisson arrivals constitute a natural model for independent arrivals. The request sequence is defined such that the density function of inter-request durations is $f(x) = \lambda \exp(-\lambda x)$ and the respective CDF (likelihood for an inter-request duration not to exceed x) is $F(x) = 1 - \exp(-\lambda x)$. The parameter λ is the *rate* that is, the average number of requests in a time duration of length 1 (the inverse of the average inter-request duration). For our purposes, it is convenient to fix the lifetime duration to $T = 1$ and vary λ (it is symmetric to fix λ and vary T). That way, λ is also the *request-rate*. The request-rate is defined to be the expected number of requests in a lifetime duration, and it equals the lifetime duration divided by expected inter-request duration. The Poisson distribution was amenable for analysis and we obtained closed-form expressions for the miss-rate as a function of the sources and λ.

Pareto-distributed inter-arrival times. Since the Web's infancy, the *Pareto* family of distributions had been a widely-used model for inter-arrival times [19, 20]. For a power $\alpha > 0$ and scale parameter $k > 0$, inter-request time durations have density function $f(x) = \alpha k^{\alpha}(x+k)^{-\alpha-1}(x \geq 0)$, and the respective CDF is $F(x) = 1 - (k/(x+k))^{\alpha}$. The distribution is heavy-tailed and has unbounded variance when $\alpha < 2$. The parameter k is inversely proportional to the rate (average number of request per time unit), which is equal to $(\alpha - 1)/k$ for $\alpha > 1$. For $\alpha \leq 1$ the expectation of an inter-request duration is unbounded, and thus the rate is 0.[3] We consider Pareto distributions with different values of α. For any given α, we vary "request-rates" by fixing $T = 1$ and sweeping k (it is symmetric to fix $k = 1$ and vary $1/T$).

Fixed-frequency arrivals. In practice, some objects are accessed mostly by robots or have access-likelihood which highly depends on time-of-day. For such objects, inter-request durations do not exhibit reference locality and instead, the histogram "peak" in various "round" values (such as one hour or a day) [6]. The actual inter-request time histogram is then a mix of fixed-frequency arrivals with another distribution which posses reference-locality. Thus, it is interesting to consider fixed-frequency arrivals, and indeed, analysis and simulations reveal different patterns than seen for Poisson and Pareto arrivals.

[3] In quantile sense, the parameter $1/k$ for Pareto serves as an analogous metric to the λ parameter of the Poisson distribution. Even though the expectation is unbounded for $\alpha \leq 1$, for any fixed p and α, the pth quantile duration is bounded, and is proportional to k. (Note that quantiles are proportional to $1/\lambda$ with Poisson.) This explains why behavior patterns we observed with Pareto $\alpha \leq 1$ arrivals by varying $1/k$ resemble those obtained when varying request-rate for Poisson arrivals.

4.2 Trace-based simulations

The request sequence received by our simulated client-cache was obtained from traces of busy proxy caches. We then simulated performance under different types of sources. Our simulations required header values of HTTP responses to calculate freshness-lifetime information for the requested URLs. This information is usually not collected by large scale caches over extended periods because of its volume. To get this information we separately performed GET requests to URLs after downloading the trace.

We calculated freshness-lifetime values from the HTTP-response headers by the Squid object freshness model (HTTP/1.1 compliant), using a CONF_PERCENT value of 10%, a CONF_MAX value of a day, and a CONF_MIN value of 0 for all URLs. As indicated in Section 2 freshness-lifetime values are fairly static so the values we obtained should be a fairly good approximation of actual values at the time requests were logged. In various simulations, we changed the age of returned copies (and TTL values) to emulate access through different types of sources.

Since we could not GET all URLs in a reasonable time frame without adversely affecting our environment, we used random sampling, weighted by number of requests, and factored it out for the final results. In total we fetched about 224K distinct URLs.

4.3 Data

We used two 6 days logs from NLANR caches [16] collected January 20th until January 25th, 2000. We used both the complete original traces and sub-traces obtained by considering only requests issued by a randomly-sampled fraction of the clients (IP addresses). (We actually sampled a different set of clients per day of the trace since the anonymization process of the NLANR logs scrambles client IP addresses differently each day.)

Table 1 lists the number of requests in each log, and the fractions of requests on which our separate GET request issued to the URL obtained (i) an HTTP response code other than 200 (OK), (ii) 200 code with a non-cachable response, and (iii) a 200 code with a cachable response.

As we shall see, the analysis suggests that performance and its dependence on source type and parameters could highly vary with the request-rate of the object at the client-cache. In the context of traces, we define the request rate as the number of requests issued for the object in the duration of the trace divided by $(t_f - t_0)/T$ (the number of freshness-lifetime durations contained in the duration of the trace). To facilitate performance measurements as a function of request-rate we partitioned objects in the trace accordingly. The partition included only objects for which request-rate is defined, namely, cachable objects. The separate consideration of each range of request-rates also allows for extrapolating results to caches with different distributions of request-rates.

We use the notation UCx$[a–b]$ and SDx$[a–b]$ for all requests made to objects with rate in $[a, b)$ in the respective trace (UC or SD) and an x fraction of the clients. Figure 5 shows that only 2% of hits and 45% of requests were on URLs with rate less than 0.2. The 0.2–2 range includes 66%–74% of hits and 40% of requests. We observed that most no-cache requests occurred on the 5–∞ range. Whereas our analysis covers all rates, these statistics indicate which rate ranges are more significant for the performance of these two

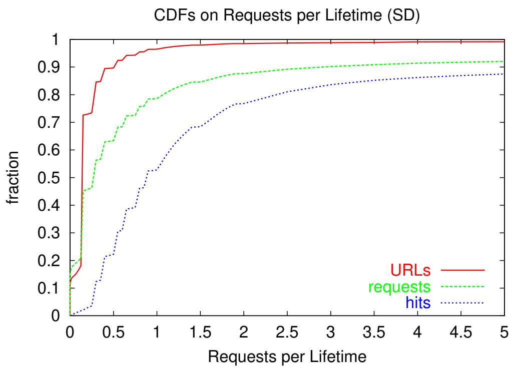


Figure 5: Cumulative fraction of URLs, requests, and hits on requests made to objects with request rate at most x.

NLANR caches. Even though higher request rates correspond to higher hit rates, since fewer requests involve very high rates, most hits are obtained in the range 0.2–2.

4.4 Performance metric

We use the *miss rate* as our performance metric. The miss rate is the ratio of *misses* (request on which the cache did not have a fresh cached copy) to "total" requests. In "total requests" we excluded some requests that would be cache misses under all source types. In particular, we did not include requests made to explicit noncachable objects. We also did not count cold-start misses (first logged request for each URL). Coupled with the miss-rate, we also consider the age-penalty.

We simulated a cache with infinite storage capacity. Thereby separating aging-issues from the well-studied performance effects of the choice of replacement policy and available storage. Our assumption of infinite storage is justified by the common scenario for high-volume well-tuned caches [16] where the freshness lifetime duration (typically 24 hours) of most objects is smaller than the duration for which an object remains cached after its most recent request (typically few days). In such caches a copy would typically become stale before it is evicted by the replacement policy.

4.5 Simulated sources

We simulated the three basic sources defined in Section 3.2. For sources other than AUTH we consider two versions according to the action on client requests that are labeled in the log as issued with a no-cache request header. For a source type x, the version x^- assumes that these requests get forwarded to an authoritative source and thus a copy with zero age is obtained. The source version x^+ provides the same effective TTL value for no-cache requests as for "regular" cache misses. Our motivation for considering x^+ is that it mimics a source that seems authoritative but is not the true origin as occur for example with CDNs. Another reason for studying these two versions is to asses the effect of no-cache requests.

5. RELATIONS BETWEEN SOURCES

Analysis reveals [4, 5] that for any sequence of requests the miss-rate of a client cache when it uses an AUTH source is never larger than the miss-rate through an EXC source.

log	# requests	HTTP RC $\neq$ 200	Not cachable	Cachable
UC1 [0–∞]	5.6M	0.93M (16.6%)	0.21M (3.7%)	4.5M (79.7%)
SD1 [0–∞]	7.5M	0.97M (12.8%)	0.4M (5.3%)	6.1M (80.6%)

Table 1: Classification of the requests in the NLANR logs.

In turn, the miss-rate through an EXC source is never larger than through an IND source. It follows that in terms of age penalty, authoritative servers are the most effective source. For replicating sources, it is more effective to use the same cache (per client-object pair) rather than alternate between several caches. So when configuring top-level Web content caches one would like to partition the workload such that each client-object pair is consistently mapped to the same cache.

It is now interesting to ask what are the performance gaps between the different sources. Worst-case analysis shows [4, 5] that the age-penalty of an EXC source can be 1 (double the miss-rate incurred with an AUTH source) whereas an IND source can have age-penalty of $e - 1 = 1.718\ldots$ [4, 5]. Here we explore the performance gaps for request sequences drawn from specific distributions and actual proxy logs.

5.1 Fixed-frequency arrivals

We extend the example of Section 3.1 to arbitrary fixed frequency arrivals and compute the miss-rate of different sources as a function of the request-rate.

Consider a client-cache that get requests for an object in regular intervals of $f * T$ $(0 < f)$ where T is the freshness lifetime of the object (the request-rate is $\lambda = 1/f$). If the client-cache uses an AUTH source then the TTL of the copy obtained after a miss would be T and thus the $\lfloor 1/f \rfloor$ subsequent requests must be hits so the miss rate of the client-cache is $1/(\lfloor 1/f \rfloor + 1)$. Through an EXC source, the miss rate is $\min\{f, 1\}$, since each request is a miss in a fraction of $\min\{f, 1\}$ of the possible source displacements $\alpha \in [0, T]$ (this is the likelihood that a T-interval started between the previous and current requests). Therefore, the age penalty of EXC is $\min\{f, 1\} * (\lfloor 1/f \rfloor + 1) - 1$. Hence, for fixed frequency arrivals AUTH is more effective than EXC if $f \leq 1$.

We next consider an IND source. Let X be a random variable that counts the number of hits at the client-cache that follow a miss. The variable X takes the value i if the TTL drawn was between $i * f * T$ and $(i + 1) * f * T$ for every $1 \leq i \leq \lfloor 1/f \rfloor - 1$. Variable X can also take the value $\lfloor 1/f \rfloor$ with probability (1 mod f). So we obtain that the expectation of X is

$$E(X) = (\lfloor 1/f \rfloor)\left((\lfloor 1/f \rfloor - 1)f/2 + (1 \bmod f)\right) .$$

Hence, the miss rate is 1 if $f \geq 1$ and otherwise it is

$$1/\left(1 + (\lfloor 1/f \rfloor)\left((\lfloor 1/f \rfloor - 1)f/2 + (1 \bmod f)\right)\right) ,$$

which can be simplified to $2/(1/f + 1)$ for integral values of $1/f$. The behavior is more easily viewed through Figure 6, which shows the miss-rate and age-penalty as a function of f. The miss-rate decreases with request-rate. The age-penalty, however, exhibits non-monotonic dependence on request-rate, where integral values of $\lambda = 1/f$ incur considerably lower penalty than slightly lower values. For high request-rates, the age-penalty through IND approaches 1 and through EXC approaches 0. For request-rates lower than 1 $(f \geq 1)$ the age-penalty is 0.

trace	AUTH	EXC	IND
UC1 [0–∞]	42%	50%	52%
UC1 [5–∞]	3.0%	3.8%	5.0%
UC1 [2–5]	18%	25%	29%
UC1 [0.2–2]	42%	55%	57%
UC1 [0^+–0.2]	82%	85%	86%
UC0.1 [0–∞]	45%	51%	52%
UC0.1 [2–∞]	23%	32%	33%
SD1 [0–∞]	47%	56%	57%
SD1 [5–∞]	3.3%	4.0%	5.3%

Table 2: Simulated miss-rate for different logs, sources, and request-rates

5.2 Poisson and Pareto arrivals

We now consider the performance of a client-cache which receives a stream of requests which is generated by Poisson process with rate λ.

The miss rate through AUTH is $1/(1 + \beta)$, when β is the expected number of requests in a T-interval following a cache miss. For a memoryless process, β is simply the expected number of requests in a T-interval following any request. For a Poisson process, $\beta = \lambda$ (expected λ requests in a T-interval), and thus, the miss rate is $1/(1 + \lambda)$.

To obtain the miss rate through an EXC source, consider some fixed source displacement α and the corresponding partition of time into T-intervals. A request constitutes a miss if and only if it is the first request in a T-interval. Therefore if $\bar{r}$ is the average number of requests in a nonempty T-interval, then the miss rate is $1/\bar{r}$. For a Poisson process with rate λ, the likelihood of a given T-interval to be empty is $\exp(-\lambda)$, and thus, the expected number of requests in a non-empty interval is $\lambda/(1 - \exp(-\lambda))$, thus the miss rate is $(1 - \exp(-\lambda))/\lambda$.

The miss rate through IND is $1/(1 + \gamma)$, where γ is the expected number of requests occurring after a miss in an interval drawn from $U[0, T]$. For a Poisson process, the number of requests is proportional to interval length and thus, $\gamma = \lambda/2$, yielding a miss rate of $1/(1 + \lambda/2)$.

Figure 6 shows the miss-rate and age-penalty for different source types, as a function of $1/\lambda$ for Poisson arrivals, and as a function of k for Pareto arrivals with $\alpha = 1, 2$. We observed similar patterns in Pareto arrivals simulations with values of α ranging between 0.5 and 10.

5.3 Trace-based simulations

Table 2 lists the miss-rates for different request-rate ranges under different (simulated) basic sources. The age-penalty through EXC or IND sources amounts to an overall 15%-25% increase in misses. The penalty is slightly larger for IND source, and is larger for higher request-rates. These numbers correspond to the simulated source treating requests with a no-cache header as regular GET requests. The gap between EXC and IND is slightly smaller otherwise.

5.4 Lessons and implications

At the extremes, for low or high request-rates, the age-

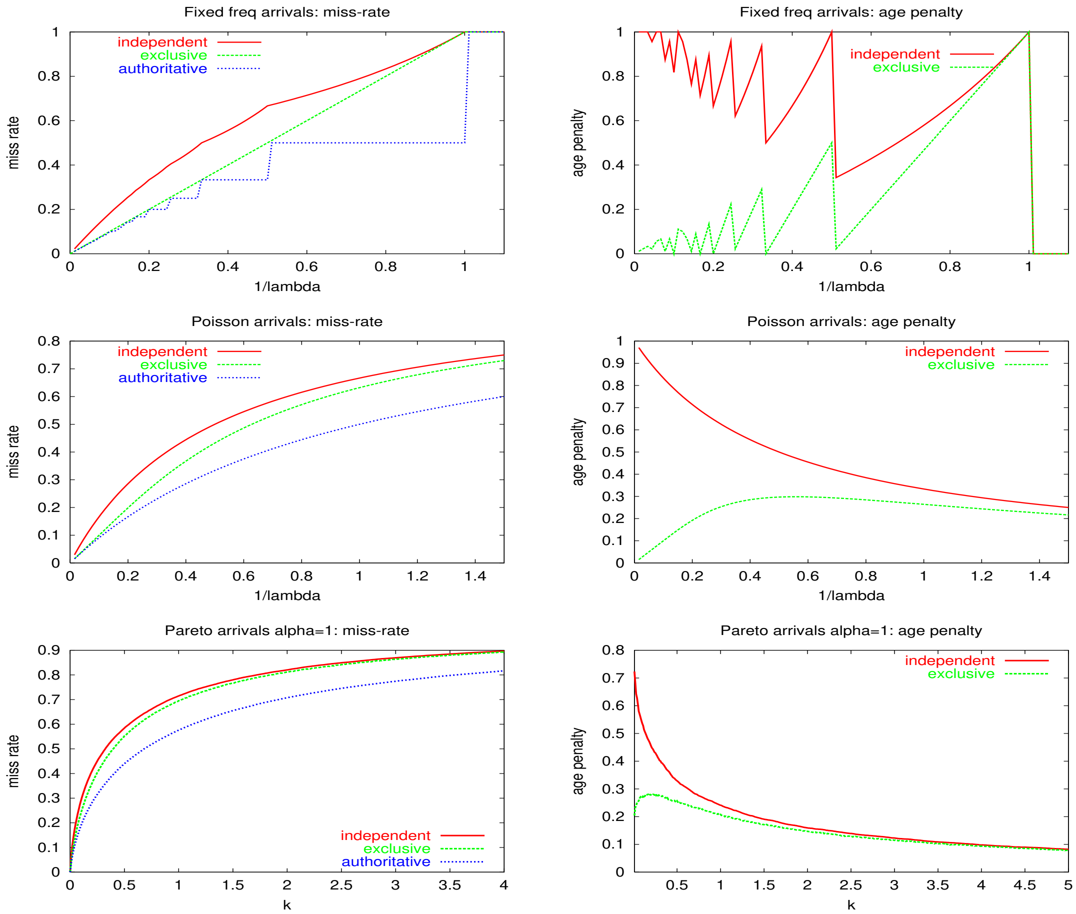


Figure 6: Fixed frequency (top), Poisson (middle), and Pareto $\alpha = 1$ arrivals (bottom). Miss-rate and age penalty dependence on (inverse) request-rate.

penalty of EXC approaches 0 whereas the age-penalty of IND approaches 1 for high rates and 0 for low rates. The miss-rate curves of IND and AUTH typically remain well-separated across requests-rates, with EXC being closer to AUTH with higher rates and closer to IND with lower rates.

Both Poisson and Pareto arrivals exhibit similar patterns with the age-penalty of IND source being monotonic increasing with rate and the age-penalty of EXC being bitonic, peaking at an intermediate request-rate value (which varies by distribution).

This shows that the magnitude of the age-penalty highly depends on the mix of request-rates at the client cache. Our trace-based simulations showed a gap of 15%-20% in miss-rate between AUTH and EXC sources, and a smaller gap of 2%-4% between EXC and IND.

Interestingly, the (very different) Poisson and Pareto distributions exhibit similar patterns, and differ from the choppier patterns for fixed-frequency arrivals. As we shall see, this observation prevails in more-involved scenarios explored later on and we attribute it to the reference-locality property of these distributions.

6. REJUVENATION

We now analyze the effect of pre-term refreshes. A *Pre-term refresh* by a cache is a refresh (validation request sent to an origin server) of a non-expired copy of an object. As a result, the cache obtains a copy with zero age. A pre-term refresh can occur when the cache receives a client request with a *no-cache* request header. We also consider top-level caches that are configured to do periodic pre-term refreshes of objects as a proactive mechanism for reducing the age penalty. We call this mechanism *Rejuvenation*, and we call a cache which is configured to use it a *rejuvenating cache*.

We consider specific types of rejuvenating sources defined as follows. The source EXC$_v$ (respectively, IND$_v$) is an EXC (resp., IND) source that refreshes its copy of the object when its age exceeds v fraction of the lifetime duration. The period of EXC$_v$ is of length vT and with displacement $\alpha \in [0, vT]$. So at time t the TTL provided by EXC$_v$ is $T - (t + \alpha) \mathrm{mod}(v * T)$. As with the basic EXC source, when considering the performance of EXC$_v$ we average over uniformly chosen displacement $\alpha \in [0, vT]$. The source IND$_v$ provides copies with age drawn from $U[0, vT]$, and thus, TTL drawn from $U[(1-v)T, T]$. For both IND$_v$ and EXC$_v$ sources, a rejuvenation interval of $v = 1$ corresponds to the respective basic source: EXC$_1$ ≡EXC and IND$_1$ ≡IND. A rejuvenation interval of $v = 0$ corresponds to the basic AUTH source. That is, EXC$_0$ ≡AUTH and IND$_0$ ≡AUTH. Figure 7 illustrates the TTL for rejuvenation with $v = 0.5$.

At first glance, it seems that since rejuvenation reduces

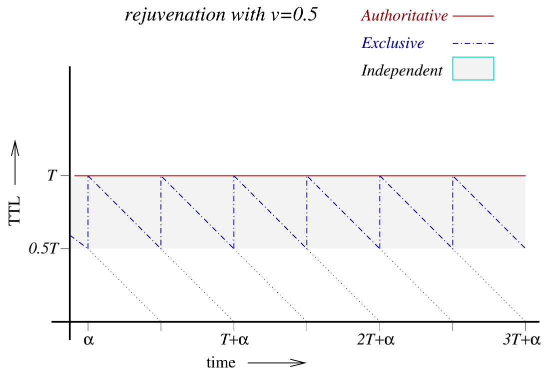


Figure 7: TTL as a function of time for rejuvenating sources with $v = 0.5$. $\text{EXC}_{0.5}$ has a period of $0.5T$ where the age cycles from $T/2$ to T and thus, the TTL cycles from T to $T/2$. For $\text{IND}_{0.5}$ the age at any point in time is independently drawn from $U[T/2, T]$ and thus the TTL is drawn from $U[T/2, T]$. The dotted lines correspond to expiration times of previously-served copies by $\text{EXC}_{0.5}$. The expiration times are at $iT/2 + \alpha$ for integral values of i.

the average age of cached items, it can only improve performance of client-caches. Furthermore, one might expect a monotonic improvement in miss rate at the client cache as v decreases from $v = 1$ to $v = 0$. This behavior indeed occurs for a rejuvenating IND_v source [4, 5]. In contrast, EXC_v sources exhibit more involved patterns where for some values of $v < 1$, for high request rates, the miss-rate with EXC_v can be strictly worse than through basic EXC.

LEMMA 6.1. *Consider the source EXC_v with $v > 1/2$ and a request sequence with inter-request times of at most $T(2v-1)$ time units. Let m_v be the miss-rate through EXC_v. Then $m_v = m_1/v$.*

PROOF. A cached copy at client-cache expires $(1 - v)T$ time units after the source most-recent refresh. Hence, if a request arrives at client-cache shortly after the TTL expired, it would get a TTL of at most vT. If objects are requested at client-cache every $T(2v - 1)$ time units, then it incurs a miss every vT time units whereas without rejuvenation a miss would be incurred once every T time units. □

Moreover, it can be shown through examples that EXC_v $(7/8 > v > 3/4)$ can induce miss-rate even higher than through the basic IND source. These claims suggest that EXC_v sources are more effective when $1/v$ is integral (i.e., $v = 1/2, 1/3, \ldots$). Thus, since sporadic pre-term refreshes are better captured by non-integral $1/v$ values, they can degrade performance at client caches.

6.1 Poisson and Pareto arrivals

We now consider the performance of a client-cache that receives requests generated by a Poisson process with rate λ (per T-interval) that upon a miss contacts a rejuvenating source. Following the derivation in [5]: the miss rate as a function of v and λ is

$$\frac{1}{1 + (2 - v)\lambda/2}$$

through an IND_v source, and

$$\frac{1}{\lambda v\left(\lfloor 1/v \rfloor + \exp(\lambda v(1/v - \lfloor 1/v \rfloor))/(\exp(-v\lambda) - 1)\right)}$$

through an EXC_v source.

Figure 8 illustrates dependence of miss-rate on v, for 3 representative request rates, for a cache using EXC_v and IND_v sources.

Simulations that we performed for Pareto arrivals when varying the scale parameter k (for a fixed power α) revealed patterns similar to those obtained for Poisson arrivals when varying $1/\lambda$. Figure 9 shows simulation results for some representative "request-rates" and values of the power α.

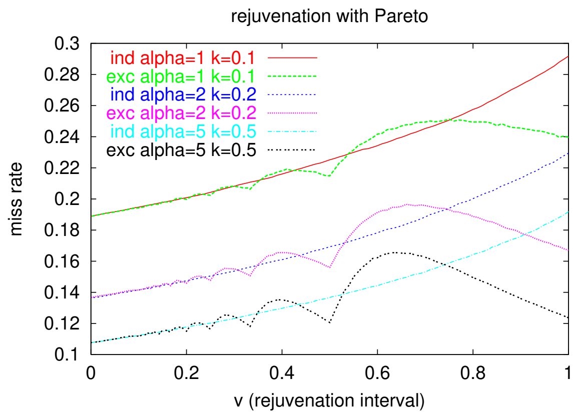


Figure 9: Rejuvenation with Pareto arrivals

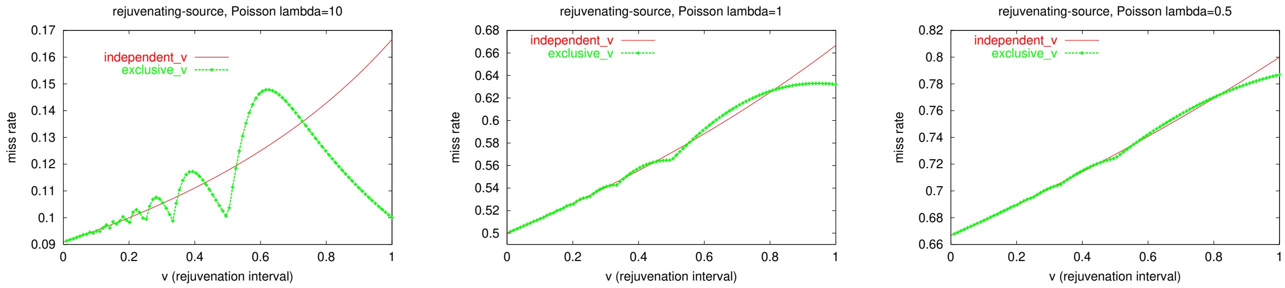
Elementary derivations show the following for Poisson arrivals across all rates. Simulations using Pareto arrivals and traces suggest that these patterns are present for these sequences as well.

- The miss rate of the client cache through IND_v and EXC_v converges to its miss rate through AUTH as v approaches 0, and to its miss rate through IND and EXC respectively as v approaches 1.

- The miss-rate through IND_v is monotone increasing with v and convex.

- The miss rate of EXC_v has local minima for v's such that $1/v$ is integral. For these values of v, EXC_v outperforms IND_v. EXC_v restricted to these points is a convex monotone increasing function of v. Between each pair of local minima EXC_v is a concave function of v, and has a local maxima around which it performs worse than IND_v. This is more pronounced for high rates ($\lambda \gg 1$). The ratio is less than 1 for integral values of $1/v$ and is improving with increasing $1/v$. For non-integral values and higher request rates the ratio is larger than 1.

- For cost-benefit considerations, we consider the miss-rate as a function of $1/v$ (*rejuvenation cost*). With IND_v, the function is convex and monotone. With EXC_v, the function is monotone and concave function between integral values of $1/v$. Restricted to integral values of $1/v$ the miss rate through EXC_v is convex and monotone decreasing function of $1/v$. Note that



$\lambda = 10$ $\lambda = 1$ $\lambda = 0.5$

Figure 8: Miss rate dependence on rejuvenation interval for different rates.

the tradeoffs generated by integral values of $1/v$ define the lower envelope of the set of tradeoffs for all possible values of $1/v$. Therefore to get the smallest miss rate for a particular cost, say x, the source has to alternate between rejuvenating with parameter v_1 and rejuvenating with parameter v_2 where $1/v_1 = \lfloor x \rfloor$ and $1/v_2 = \lceil x \rceil$.[4]

Note that although we observed similar patterns for Poisson arrivals, Pareto arrivals and in our traces (see Section 6.2) these patterns are not universally true for all sequences. For example, it is possible to construct sequences on which the miss-rate with $\text{EXC}_{1/2}$ is strictly lower than with $\text{EXC}_{1/3}$.

6.2 Trace-based simulations

Figure 10 shows the miss rate as a function of the rejuvenation interval $v \in [0, 1]$ for the sources IND_v and EXC_v. The results shown are for the UC trace. Results for the SD trace were similar and are not shown. The simulations show the presence of patterns obtained in our Poisson analysis. We observed that treatment of no-cache requests in the simulation did not seem to affect general patterns, but the patterns were somewhat smeared for the IND_v^- and EXC_v^- sources, due to some loss of synchronization (see definition in Section 3). The patterns were evident in the sampled traces UC0.1 and SD0.1 (not shown), but to a much lesser extent due to considerably lower request-rates.

6.3 Lessons and implications

In this section we discuss implications of our results to tuning of higher-level caches that perform pre-term refreshes. We learned that under a wide range of circumstances, sporadic pre-term refreshes (caused by requests with a `no-cache` request header) or rejuvenations that are not well-timed can result in incomparable, often inferior, performance of a client-cache relative to its performance under the basic EXC source.

Synchronization (see definition in Section 3) of a client with the source cache would guarantee that the performance of the client is no worse than its performance through the basic EXC source. An EXC_v source is guaranteed to remain synchronized with all clients only if $1/v$ is integral. With sporadic pre-term refreshes, a source cache is only guaranteed to remain synchronized with the one client which invoked the most-recent pre-term refresh (delivered the `no-cache` request) and with clients with already-expired copies.

A source that would like to serve sporadic `no-cache` requests without loosing synchronization with its other clients can do one of the followings.
1) It can serve the `no-cache` request by contacting an origin server but refrain from updating the expiration time on its cached copy.
2) It can update the expiration time of its copy but perform another *follow-up* pre-term refresh of the object at its original expiration time.

Rejuvenation policies and follow-up refreshes increase traffic in the *upstream channel* between the high-level cache and origin servers while potentially reducing user-perceived latency and traffic in the *downstream channel* between the high-level cache and its clients (see Figure 11). This tradeoff should guide the selection of rejuvenation interval or the follow-up action on a sporadic pre-term refresh.

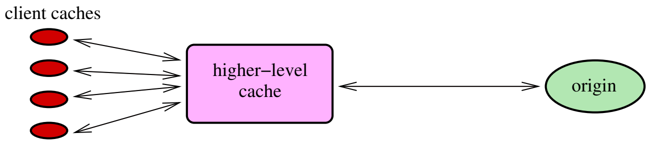


Figure 11: Rejuvenating source, clients, and origin.

Consider the simplified metric where the cost is the number of unsolicited refresh requests issued by the high-level cache and the benefit is the reduction in the number of misses incurred at client caches. Whereas the cost is independent of client activity and rather straightforward to estimate (for rejuvenation it is proportional to $1/v$), estimating the benefit, which is aggregated across all client caches, is a more involved task. The objective is then to maximize the benefit (minimize the total number of misses at client-caches), given some bound on the cost.

An interesting challenge left for future work is to efficiently estimate the benefit, possibly on-line, with small book-keeping of per-client history, for example, by tracking a sample of the clients. We do establish a general guideline that rejuvenation frequency with integral $1/v$ dominate all others. That is, the average benefit of mixing two rejuvenations intervals such that $1/v_1$ and $1/v_2$ are consecutive integral values dominate (have equal or higher benefit) all other choices of v with the same or lesser cost. The claim

[4]The source should behave as EXC_{v_2} for $x - \lfloor x \rfloor$ fraction of the time and as EXC_{v_1} for the rest of the time. As each change of the rejuvenation frequency may break synchronization, alternations should be very infrequent, or avoided altogether by balancing cost through several objects (each having a fixed rejuvenation frequency).

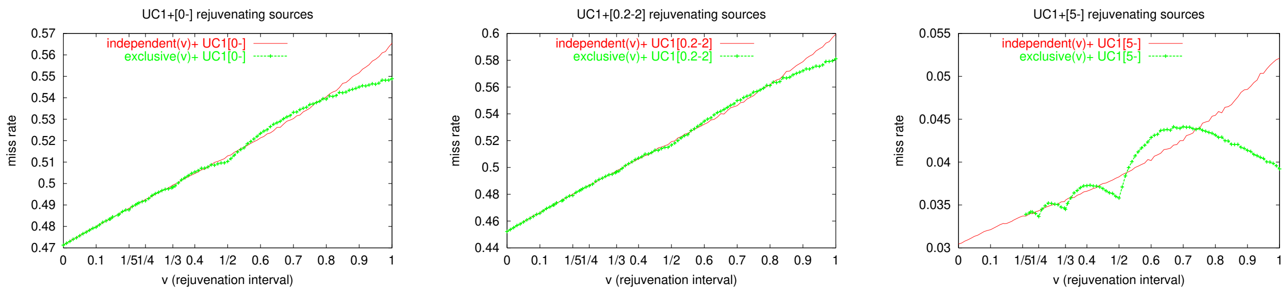


Figure 10: Miss rate dependence on rejuvenation interval for UC trace, UC1$[0.2–2]$, and UC1$[5–\infty]$.

follows from our Poisson analysis and the fact that it is sufficient to establish it for a single client-cache. Although not universally true for all request sequences, our trace-based simulations and Pareto simulations suggest that this claim applies to these sequences as well. The extent to which this guideline is useful depends on the gap between non-integral values and the lower-envelope constituting of integral values, which increases with request-rate.

7. EXTENDED LIFETIME

In some situations, client caches use longer freshness lifetime durations than their source, either as a result of different configurations or as an intentional attempt of the client cache to reduce miss-rate and outgoing traffic. In the Web caching context, most HTTP requests are issued to objects without explicit expiration to which caches apply a heuristic freshness lifetime calculation (see Section 2). Thus, longer lifetime durations can stem from different URL filters, or setting larger values for the parameters CONF_PERCENT or CONF_MAX [22]. Lifetime extension trades improved miss-rate with decreased coherence.

To facilitate the analysis of extension, we consider client caches which apply an extended freshness lifetime value of $r * T$ for some $r > 1$ (T is the freshness lifetime value used by the source). We refer to r as the *extension factor*. We use the notation AUTH(r) (respectively, IND(r), EXC(r)) for a source of type AUTH (respectively, IND, EXC) when the client-cache applies extension factor of r. Qualitatively, extension resembles rejuvenation, as there is effectively a fixed ratio between lifetime durations at the source and at the client cache. The extension factor, however, is controlled by the client cache, and for a given request sequence, varying the extension factor is not equivalent to varying rejuvenation interval. Figure 12 illustrates TTL durations for different source types with extension factor of 1.5.

As intuitively expected, we prove in [4, 5] that through any of our three basic source models (AUTH, EXC or IND) the miss rate can not increase when lifetimes are extended. Extended lifetimes, however, also decrease coherence. We study the tradeoff curve of the extension factor vs. the miss-rate for different source types and show which extension factor values achieve the best balance.

The following Lemma reveals a surprising step-like form of the tradeoff curve for high request rates.

LEMMA 7.1. *Consider an* EXC(r) *source and a request sequence such that the object is requested at least once every* $(\lceil r\rceil - r)T$ *time units. Then, the miss rate is the same as with* EXC($\lfloor r\rfloor$).

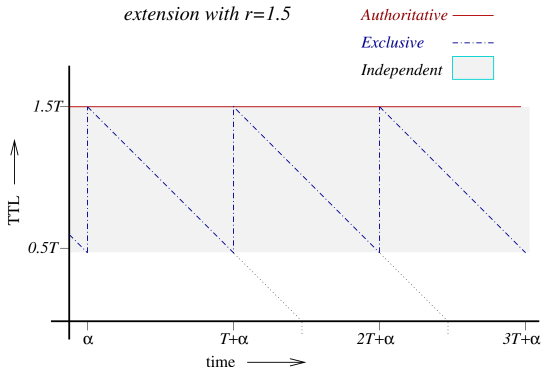


Figure 12: TTL obtained through different types of sources when the client cache uses extension factor of $r = 1.5$. AUTH source provides copies with zero age and thus TTL of rT $(1.5T)$. EXC source (shown with displacement α) provides copies with age that cycles from 0 to T and thus a TTL that cycles from rT to $(r-1)T$ $(1.5T$ to $0.5T)$. IND source provides at each point in time a copy with age independently drawn from $U[0,T]$ and thus TTL drawn from $U[(r-1)T, rT]$ $(U[0.5T, 1.5T])$. The dotted lines correspond to expiration times of previous copies at the client cache. Copies expire at times $(i+0.5)T+\alpha$ for integral values of i.

PROOF. When a copy expires at the client-cache, the copy at the source has age $(r - \lfloor r\rfloor)T$. The object is requested before the source refreshes its own copy. Thus, a miss at the client cache is incurred once every $\lfloor r\rfloor T$ time units. ☐

Note that the lemma does not hold for lower request rates. For example, for fixed-frequency arrivals with $f = 1.2$, the miss rate is 1 with $r \leq 1.2$ and is 0.8 with $r = 1.5$.

Observe that a client-cache is synchronized with its EXC(r) source if and only if r is integral. Figure 12 illustrates the lack of synchronization with $r = 1.5$: when previously-served copies expire at the client cache, the source provides a copy with age $0.5T$ and thus a TTL of T.

7.1 Poisson and Pareto arrivals

We consider a client cache that uses extension factor of r and receives requests generated by a Poisson process with rate λ. For this sequence it is possible to calculate the miss-rate of the client cache as a function of r and λ through the different source types [5]. Through an AUTH(r) source the miss rate is $\frac{1}{1+\lambda r}$, through IND(r) the miss rate is $\frac{1}{1+\lambda(r-1/2)}$, and through an EXC($r$) source the miss rate is

$$\frac{1}{\lambda\left(\lfloor r\rfloor + \exp(\lambda(r-\lfloor r\rfloor))/(\exp(\lambda)-1)\right)} .$$

For high rates (large values of λ), the miss rate of EXC(r) is $\approx 1/(\lambda\lfloor r\rfloor)$, which is a step-like function with drops at integral values of r. Derivations establish that the tradeoff curve of miss rate vs. extension is concave between any two integral values of r, but is convex and monotone decreasing when restricted to integral values of r. Figure 13 shows the miss-rate as a function of the extension factor for different source types and request rates. It illustrates, in particular, the monotone convex dependence for AUTH and IND sources. The figure also illustrates that across all request rates, EXC(r) outperforms IND(r) for integral r's but is outperformed by IND(r) on some intermediate values. At the extremes, for low request rates, EXC(r) and IND(r) converge, and are separated from AUTH(r). For high request rates EXC(r) becomes more step-like: its performance on integral values of r converges to that of AUTH(r) but for non-integral values it can be strongly outperformed by IND(r).

As we shall see, these patterns are also present for Pareto and our trace-based simulations. We note, however, that they are not universal for arbitrary inter-request distributions. In particular, some distributions with no reference-locality do not conform to these patterns.

Figure 13 shows simulation results of Pareto arrivals with extension for representative values of the power α and request-rates. The simulations show the general patterns that were analytically revealed for Poisson arrivals. As with Poisson arrivals, for each fixed α, the pattern was more pronounced for higher request-rates.

7.2 Trace-based simulations

We simulate the performance of our cache with different extension factors and under the different source types. The source simulations are as outlined in Section 4. To incorporate extension factor of r, our simulated cache computes the freshness lifetime to be r times its calculated value for the respective source type.

Figure 13 shows the miss rate for different request-rates as a function of the extension factor for the various source-types. Very similar results were obtained for the SD trace (not shown). As predicted by the analysis, the AUTH(r) and IND(r) performance curves are smooth whereas the EXC(r) performance curve has sharper drops at integral values of r. This behavior is more pronounced for higher request rates. It is also evident that for integral values of r, EXC(r) outperforms IND(r) and for some intermediate values IND(r) outperforms EXC(r).

7.3 Lessons and implications

The patterns derived from the Poisson analysis and the Pareto and trace-based simulations provide guidelines for an effective use of *extension*.

Extension factor selection can vary with object and with time, as to obtain a better balance of the *degree of incoherence* and the *cost of cache misses*. Let us assume that for a particular object, the degree of incoherence is the average value of the extension factor r, and the miss-cost is proportional to the number of misses. We now ask, given maximum allowed incoherence, how to select extension factors as to minimize the miss-cost. Interestingly, for this objective, we obtain opposite guidelines when using an EXC source than when using IND or AUTH sources.

- The form of the miss rate vs. extension tradeoff curve of EXC(r) implies that across all request rates, integral values of r provide the best tradeoff of incoherence and miss-rate. In particular, a mix of consecutive integral values of the extension factor incurs lower miss-cost than a non-integral extension factor with the same degree of incoherence. (A mix of EXC(1) and EXC(2) with the same incoherence as EXC(1.5) has lower cost.) The benefit of using integral values depends on the performance gaps, which are more pronounced for higher request rates.

- With IND and AUTH sources, the dependence of miss-rate on the extension factor is convex, and monotone decreasing as r increases. Thus, a fixed value of r would result in lower miss-rate than a mix of values with the same average degree of incoherence.

8. CONCLUSION

We explored *age* as a new facet of cache performance which affects cascaded caches. Age is complementary to other cache-performance issues: it affects frequently-requested objects, and prevails even with infinite storage. Surprisingly, age-related performance issues had been by and large overlooked. We used analysis and simulations to evaluate performance aspects of age under various cache configurations and behaviors. In particular, we studied extended object-lifetime durations at client caches, sporadic pre-term refreshes, and *rejuvenation* at high-level caches.

To focus on age issues, our analysis factored-out other well-studied and better-understood performance facets, such as misses incurred at higher-level caches. For future work, it would be interesting to study tradeoffs between various performance facets and their combined behavior.

Acknowledgment. We would like to thank the SIGCOMM referees for insightful comments that helped improve this presentation.

9. REFERENCES

[1] Akamai. `http://www.akamai.com`.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0, May 1996.

[3] IBM WebSphere Cache Manager. `http://www.software.ibm.com/webservers/cacheman`.

[4] E. Cohen, E. Halperin, and H. Kaplan. Performance aspects of distributed caches using TTL-based consistency. In *Proc. 28th ICALP*. Springer Verlag, LNCS , 2001.

[5] E. Cohen, E. Halperin, and H. Kaplan. Performance aspects of distributed caches using TTL-based consistency. Manuscript (full version), 2001.

[6] E. Cohen and H. Kaplan. Exploiting regularities in Web traffic patterns for cache replacement. In *Proc. 31st Annual ACM Symposium on Theory of Computing*. ACM, 1999.

[7] E. Cohen and H. Kaplan. The age penalty and its effect on cache performance. In *Proceedings of the 3rd*

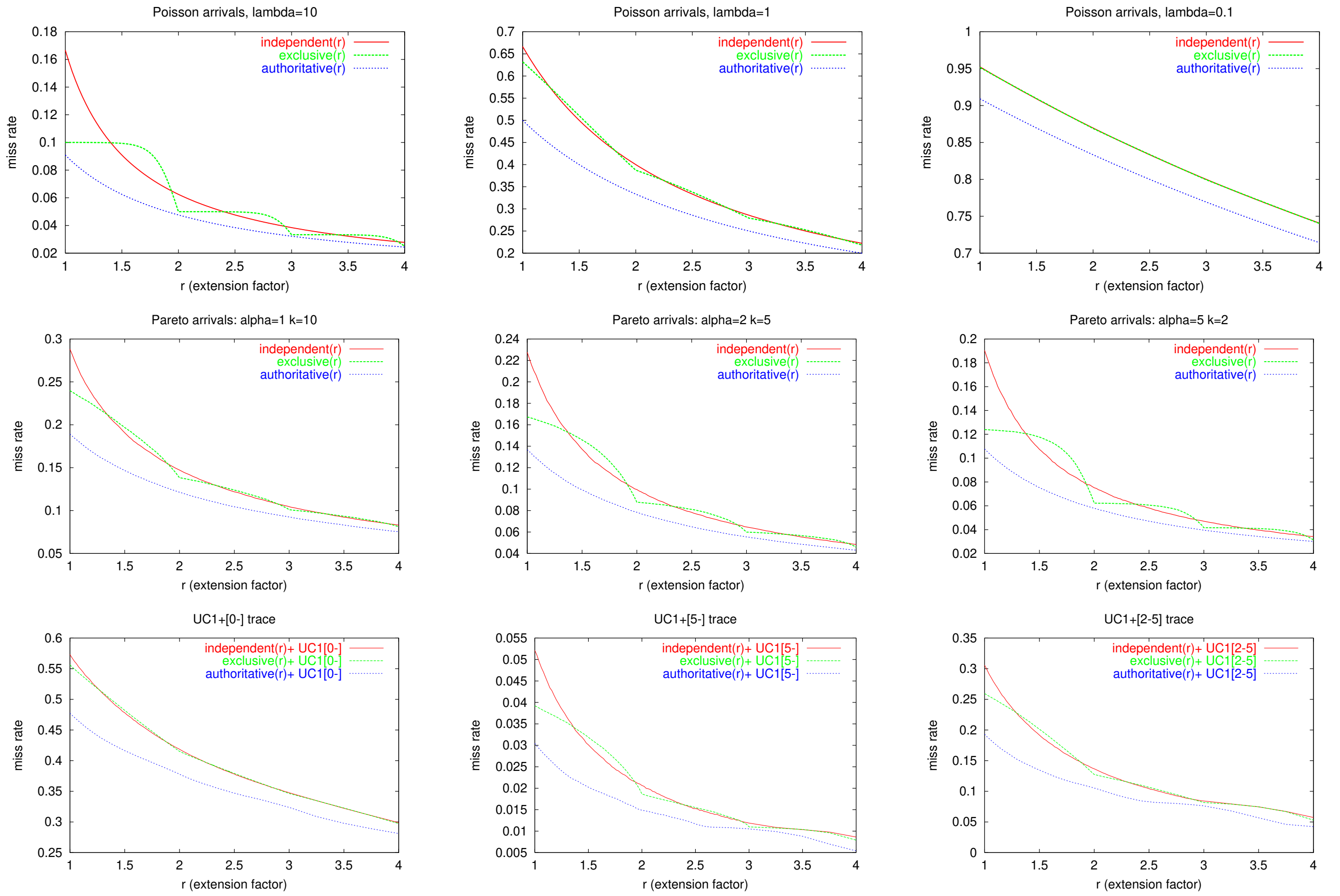


Figure 13: Miss-rate dependence on extension factor: Poisson arrivals (top), Pareto arrivals (middle), and UC trace (bottom)

USENIX Symposium on Internet Technologies and Systems, 2001.

[8] E. Cohen and H. Kaplan. Refreshment policies for Web content caches. In *Proceedings of the IEEE INFOCOM'01 Conference*, 2001.

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

[10] E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to Web servers. In *Proceedings of the IEEE INFOCOM'99 Conference*, 1999.

[11] D. Duchamp. Prefetching hyperlinks. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, 1999.

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

[13] Inktomi Traffic Server. `http://www.inktomi.com`.

[14] Inktomi Content Delivery Suite. `http://www.inktomi.com`.

[15] Lucent IPWorX. `http://www.lucentipworx.com`.

[16] A Distributed Testbed for National Information Provisioning. `http://www.ircache.net`.

[17] Balachander Krishnamurthy and Craig E. Wills. Study of piggyback cache validation for proxy caches in the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.

[18] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 13–22, December 1997.

[19] W. E. Leland, M. S. Taqq, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. In *Proc. of ACM SIGCOMM '93*, pages 183–193, 1993.

[20] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[21] Digital Island (Sandpiper). `http://www.sandpiper.com`.

[22] Squid internet object cache. `http://squid.nlanr.net/Squid`.

[23] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. on the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM symposium on Operating Systems Principles*, 1999.