Chapter 1

THE TIME-TO-LIVE BASED CONSISTENCY MECHANISM:

Understanding performance issues and their impact

Edith Cohen
AT&T Labs-Research
Florham Park, NJ, USA
edith@research.att.com

Haim Kaplan
Tel-Aviv University
Tel Aviv, Israel
haimk@post.tau.ac.il

Abstract The Web is a large distributed database were copies of objects are replicated and used in multiple places. The dominant consistency mechanism deployed for HTTP (Hyper Text Transfer Protocol) and DNS (Domain Name Service) records is Time-to-Live (TTL) based weak consistency. Each object has a lifetime-duration assigned to it by its origin server. A copy of the object fetched from its origin server is received with maximum time-to-live (TTL) that equals its lifetime duration. Cached copies have shorter TTLs since the *age* (elapsed time since fetched from the origin) is deducted from the objects lifetime duration.

A request served by a cache constitutes a *hit* if the cache has a fresh copy of the object. Otherwise, the request is considered a *miss* and is propagated to another server. With HTTP, expired cached copies need to be validated, and if they turned out to be not modified, we refer to the request as a *freshness miss*.

Cache performance is affected by TTL-based consistency. Cache misses translate into user-perceived latency. When a popular cached object expires, a cache can reduce user perceived latency by refreshing its copy proactively, before it is requested. For hierarchical caches, the number of cache misses depends in subtle ways on the age of the copies the cache receives. Thus, fresh copies obtained through another cache are less effective than fresh copies relieved from an authoritative server.

Keywords: TTL, Time To Live, Cache, Web caching, Domain Name System

1. Introduction

Distributed databases such as the Web maintain a master version of each item and a possibly large number of cached copies. Caching offloads the server and expedites delivery of the item to end users but also requires some consistency mechanism, that makes sure that when the master copy is modified, this fact is propagated to the copies. Maintaining strong consistency, where all copies are updated promptly when the master is modified, can be very resource consuming. Time-to-Live (TTL)-based consistency is a simple and effective weak consistency mechanism where the lifetime of each copy is limited to some fixed duration. TTL-based consistency allows for the benefits of using cached copies without the need to maintain state on the location of all the copies. The cached copies can be stale, but are guaranteed to be almost (as determined by the TTL duration) up to date.

The Hyper-Text Transfer Protocol (HTTP) [Fielding et al., 1999] that governs the distribution of Web content, and the Domain Name System[Mockapetris, 1987a, Mockapetris, 1987b] (DNS), that is responsible for translating between domain names and IP addresses, constitute examples of very large scale distribution systems that deploy TTL-based consistency. Objects are typically associated with one authority that originates and modifies them (their *origin* or *authoritative* server), but can be cached and further distributed from multiple *replicating* servers (caches). Caching and replication are widely deployed for reducing load on Web servers, network load, and user-perceived latency. Caches are located at different points in the network and include reverse proxies, proxy caches, and browser caches. By and large, servers rely on one widely supported and deployed mechanism for consistency of cached copies. Under this client-driven and expiration-based mechanism, the authoritative server provides an expiration time for each copy, beyond which it must be validated or discarded.

Each HTTP object has a URL which specifies its "location" and its authoritative server. The object is requested by sending an HTTP request and the content is sent back on the respective HTTP response. The response includes a header with important information on the object, including cache directives. The directives specify if the object can be cached, and may provide 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 request is processed locally. If the cached copy is *stale* (expired), it must be validated by contacting an external server with a fresh copy. To this end, HTTP provides conditional `GET` requests. Similarly, if there is no cached copy, the cache must obtain a fresh copy from an external server.

An HTTP-compliant cache [Berners-Lee et al., 1996, Fielding et al., 1999, Squid, 2001] calculates from the header fields of an object a *freshness lifetime*, which is the time interval during which the copy remains fresh (non-expired) since it left its authoritative server. The freshness lifetime is typically fixed for each object. The cache also determines from the headers the *age* of the copy, which is the elapsed time since it left its authoritative server (and resided in caches). If the age is smaller than the freshness lifetime then by subtracting the age of the copy from its freshness lifetime the cache obtains a *time-to-live (TTL)* duration for the copy (during which it is considered fresh). If the age is larger than the freshness lifetime, the object is considered stale and its TTL is zero.

In the DNS database, each resource record (RR) has a freshness lifetime duration assigned to it by its authoritative administrator. These records are cached in multiple servers and the age is calculated in a similar way as with HTTP. DNS performance is critical to many applications. In particular, name translation is necessary and precedes HTTP transfers.

Among the different performance objectives of caches, improving end-user Web experience, in particular reducing using-percieved latency, is gradually becoming the most pronounced. Many organizations are deploying caching servers in front of their LANs, mainly as a way to speed up users Web access. Generally, available bandwidth between end-users and their Internet Service Providers (ISPs) is increasing and is complemented by short round trip times. Thus, the latency bottleneck is shifting from being between end-users and cache to being between cache and origin servers. From the viewpoint of Web sites and Content Distribution Networks (like Akamai [Akamai, 2001]), decreasing costs of server-machines and backbone connectivity bandwidth along with increasing use of the Web for commercial purposes imply that server and network load are gradually becoming a lesser issue relative to end-user quality of service. At the limit, these trends indicate that communication time between local caches and remote servers increasingly dominates cache service-times and user-perceived latency, and that technologies which provide tradeoffs between traffic-increase and latency-decrease would become increasingly worthwhile for both Web sites and ISPs. User-percieved latency is incurred on cache misses, when the cache has no cached copy or a stale cached copy and therefore must communicate with an external server before serving the request.

Web caches are placed close to clients, in Web browsers, and as proxy caches or as reverse proxies, close to web servers [Inktomi, 2001, Squid, 2001]. They are also sometimes configured in hierarchies [ircache, 2001]. Thus, often there is more than one cache placed between the end user and the origin server. Different copies corresponding to the same URL and residing in different caches can have different ages and thus different TTLs. In particular, a copy obtained by a cache from another cache expires sooner than a copy obtained from an authoritative server. Therefore, a cache that forwards requests to another cache is likely to suffer more cache misses than a cache that forwards requests to an authoritative server. Furthermore, a cache is likely to obtain more client requests than an authoritative server since it serves older copies. So even if a cache maintains fresh copies, since the copies are aged, the cache is somewhat less effective than an authoritative server [Cohen and Kaplan, 2001a, Cohen and Kaplan, 2001b]. Age is a central performance factor where TTL-based consistency is used, as it affects frequently-requested objects, and it remains a performance issues even if storage is practically unlimited.

Overview

In Section 2 we start with a brief survey of other consistency mechanisms, before dwelling into exploration of TTL-based consistency. We also give an overview of the consistency mechanism used by HTTP.

In Section 3 we discuss validation latency: A considerable fraction of cache "hits" involve stale copies that turned out to be current (were not modified). These validations of current objects have small message sizes, but nonetheless, often induce latency comparable to full-fledged cache misses. Thus, the functionality of caches as a latency-reducing mechanism highly depends not only on content availability but also on its freshness.

Validation latency, and more generally, cache misses that occur on previously cached but already-expired objects can be confronted with *Refreshment policies*. Refreshment policies for Web caches were proposed and studied in [Cohen and Kaplan, 2002] and for the DNS system in [Cohen and Kaplan, 2001c]. We summarize work and results on these policies in Section 4.

TTL-based consistency affects the performance of cascaded caches, when there is more than a single cache on the path between the end user and the origin server and therefore caches can be served from caches. Sections 5 and 6 summarize results presented in [Cohen et al., 2001] and [Cohen and Kaplan, 2001b]. We will see how the miss rate of a client cache depends on the source from which it obtains its objects.

Section 5 presents and explores models for three different basic configurations. Under the first configuration, the client cache uses an authoritative server directly without a high-level cache. In the second configuration the client cache uses consistently for all its misses the same source cache. In the third configuration the client cache alternates between several source caches to resolve its misses. We will see that a cache which consistently uses the same source cache (for each object) would incur a lower miss rate than a cache that alternates between several caches. This suggests that when a set of caches handles the workload of a set of clients, it is better to partition the workload such that the same primary cache is used for each client-object pair. We then characterize how the miss rate degrades when using a source cache compared to using an authoritative server. For a single consistent source (a cache source that does not perform pre-term refreshes) we show that the miss rate is larger by at most a factor of 2. For alternating between sources, the miss rate could be somewhat worse, but not more than a factor of $e = 2.718..$ larger than the miss rate through an authoritative server [Cohen et al., 2001].

Section 6 presents an extension of the basic configurations. The basic models assumed that only expired copies can be refreshed. HTTP caches, however, may refresh fresh items. Such *pre-term refreshes* occur when a client request contains a *no-cache* request 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. Conceivably, a cache can deploy configured periodic pre-term refreshes (*rejuvenations*) to reduce the age of cached copies and as a result, improve the miss-rate at its client caches and consequently, the number of requests it receives. Rejuvenation increases traffic between a cache and the authoritative server but can decrease traffic between the cache and its clients. As illustrated in Figure 1.1, a single

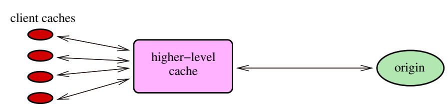


Figure 1.1. Client caches, higher-level cache, and origin server

cache can potentially serve a large number of (caching) clients, and thus rejuvenation can be very effective. As pre-term refreshes decrease the average age of cached copies, it may seem that they can only increase the effectiveness of a cache. We will see, however, that this is generally not the case. In particular, when pre-term refreshes occur in arbitrary points in time and "out of synch" with the expiration times of previous copies, performance can degrade by up to a factor of 2, which is tight under some robust conditions. We will also see, however, that when used "in synch" with previous expirations, pre-term refreshes can boost performance. The relative performance of rejuvenating sources is studied when inter-request times drawn from natural known distributions, namely, Poisson and Pareto arrivals and data from cache proxy logs.

Other work on TTL-based consistency

We summarize results that appeared in [Cohen and Kaplan, 2001c, Cohen and Kaplan, 2002, Cohen and Kaplan, 2001a, Cohen et al., 2001, Cohen and Kaplan, 2001b] but would like to mention some subsequent work on the subject.

Jung *et al* [Jung et al., 2001] explored the effect of the length of the freshness lifetime on the miss-rate. They concluded that relatively short freshness lifetime durations suffice to get most of the hits. Jung *et al* [Jung et al., 2003] supported their experimental work on DNS traces [Jung et al., 2001] with an analytical analysis. In this work they model requests as independent identically distributed random variable drawn from some fixed distribution. They develop formula for the miss rate generated by several particular distributions and analyze how freshness lifetime duration affects this miss-rate. For the distributions that fit best real traces they obtained results similar to the ones observed from the traces alone. That is, relatively short TTLs suffice to get low miss rate and the marginal affect of increasing the TTL further is small.

Hou *et al* [Hou et al., 2002, Hou et al., 2003] studied cache hierarchies with more than two levels. For requests generated by a Poisson distribution they calculate the miss-rate and average TTL at different levels of the hierarchy. Their analysis quantifies how the TTL shortens, and the miss rate increase, as we go to deeper caches in the hierarchy. They also analyze the effect of a rejuvenation mechanism on the TTL at various levels.

2. Preliminaries

Consistency mechanisms

The appeal of TTL-based consistency is in its simplicity. In particular, it requires no end-to-end support (in particular server support). To place it in context, we provide a brief discussion of other proposed consistency mechanisms.

With server-driven mechanisms, clients are notified by the server when objects are modified (e.g. [Howard et al., 1988, Li and Cheriton, 1999]); With client-driven mechanisms, the cache validates with the server objects with a stale cached copy; with hybrid approaches, validations are initiated either at the server or the client.

Leases are well-studied consistency mechanism where the server commits to notify a cache of modification, but only for a limited pre-agreed period [Gray and Cheriton, 1989, Cao and Liu, 1998, Yin et al., 1999, Duvvuri et al., 2000].

Server-driven mechanisms provide strong consistency and can eliminate all freshness misses. Hybrid approaches can provide a good balance of validation overhead and reduced validation traffic. While obviously strong consistency can be critical for some applications, none of these mechanisms, however, is widely deployed or even standardized. Implementation requires protocol enhancements and software changes not only at the cache, but at all participating Web servers, and may also require Web servers to maintain per-client state. Except for some proprietary coherence mechanisms deployed for hosted or mirrored content [Akamai, 2001, InktomiCDS, 2001] (which require control of both endpoints), the only coherence mechanism currently widely deployed and supported by HTTP/1.1 is *client-driven* based on TTL (time-to-live).

HTTP freshness control

We provide a simplified overview of the freshness control mechanism specified by HTTP and supported by compliant caches. For further details see [Fielding et al., 1999, Berners-Lee et al., 1996, Squid, 2001, Nottingham, 1999, Mogul, 1999]. 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 is performed using directives and values found in the object's HTTP response headers.

When the cache receives a client request for an object then it acts as follows:

- If the object is cached and fresh, the request constitutes *content and freshness hit* and the cached copy is immediately returned to the client.
- If the object is cached, but stale, the cache issues a conditional HTTP `GET` request to the origin server (or another appropriately-selected cache). The conditional `GET` uses the entity tag of the cached copy (with HTTP/1.1 and if an E-TAG value was provided with the response) or it issues an `If-Modified-Since (IMS)` request with the LAST-MODIFIED response header value (indicating last modification time of the object). If the source response is `Not-Modified` then the request constitutes a *content hit* but a *freshness miss*. If the object was modified, the request is a *content miss*.
- If the item is not found in the cache, then it is fetched from the origin (or another cache) and the request constitutes a *content miss*.
- If the request arrives from the client with a `no-cache` header then the cache forwards it to the origin server. The cache must forward the request even if it has a fresh copy. The cache uses the response to replace or refresh its older copy of the object. We refer to such requests as *no-cache* requests.

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 is supposed to indicate 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 then proceeds as follows. First, if a MAX-AGE directive is present, the value 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).

3. Validation latency

Servicing of a request by a cache involves remote communication if the requested object is not cached (in which case the request constitutes a *content miss*). Remote communication is also required if the cache contains a copy of the object, but the copy is *stale*, that is, its freshness lifetime had expired and it must be validated (by the origin server or a cache with a *fresh* copy) prior to being served. If the cached copy turns out to be modified, the request constitutes a content miss. Otherwise, the cached copy is *valid* and we refer to the request as a *freshness miss*. Validation requests that turn out as freshness misses typically have small-size responses but due to communication overhead with remote servers, often their contribution to user-perceived latency is comparable to that of full-fledged content misses.

Thus, cache service times can be improved by reducing both content and freshness misses. The content hit rate is measured per object or per byte and sometimes weighted by estimated object fetching cost. It is dictated by the available cache storage and the *replacement policy* used. Replacement policies for Web caches were extensively studied (e.g. [Williams et al., 1996, Cao and Irani, 1997, Bestavros et al., 1995, Cohen et al., 1998a, Cohen and Kaplan, 1999, Breslau et al., 1999, Feldmann et al., 1999, Young, 1998]).The freshness hit rate of a cache is not directly addressed by replacement policies or captured by the content hit rate metric.

The expiration time of each object is determined when it is brought into the cache, according to attached HTTP response headers provided by the origin server. Expired content must be validated before being served. Most current caching platforms validate their content *passively* i.e. only when a client request arrives and the cached copy of the object is stale. They perform validation via a conditional `GET` request (typically this is an `If-Modified-Since (IMS) Get` request). This means that validation requests are always performed "online," while the end-user is waiting. Here we promote proactive refreshment where the cache initiates unsolicited validation requests for selected content. Such "offline" validations extend freshness time of cached objects and more client requests can be served directly from the cache. Our motivation is that the most significant cost issue associated with freshness misses is their direct effect on user-perceived latency rather than their effect on server and network load, and thus it is worth performing more than one "offline" validation in order to avoid one performed "online."

A mechanism that reduces validation latency with TTL-based consistency is that the server piggy backs validations on responses to requests for related objects [Krishnamurthy and Wills, 1997, Cohen et al., 1998b, **?**]. "Related" objects are those that are often requested after the current request; the cache then updates the freshness status of its content of these related objects and therefore eliminates validation latency on these requests. This scheme requires server support. Another line of work proposed to transfer stale cached data from the cache to the client's browser while the data validity is being verified [Dingle and Partl, 1996] or while the modified portion (the "delta") is being computed [Banga et al., 1997]. These schemes require browser support and are mostly effective when there is limited bandwidth between end-user and cache (such as with modem users).

Refreshment policies, which extend freshness periods of selected cached objects were proposed in [Cohen and Kaplan, 2002] (for Web caches) and in [Cohen and Kaplan, 2001c] for caches of Domain Name System (DNS) servers. With these policies, caches proactively validate selected objects as they become stale, and thus allow for more client requests to be processed locally. Refreshment policies operate within the existing protocols and exploit natural properties of request patterns such as frequency and recency. Although not yet supported by some widely deployed Web caching platforms (e.g., Squid [Squid, 2001]), proactive refreshment is offered by some cache vendors [cacheman, 2001, ipworx, 2001]. Such products allow refreshment selections to be configured manually by the administrator or integrate policies based on object popularity.

For DNS caches, refreshing an object is equivalent to issuing a new request: there is no `If-Modified-Since` mechanism and also not much need for it as record sizes are typically small. Each DNS resource record (RR) has a TTL value initially set by its authoritative server. A cached RR becomes stale when its TTL expires. Client queries can be answered much faster when the information is available in the cache, and hence, refreshment policies which renew cached RRs offline increase the cache hit rate and decrease user-perceived latency.

4. Refreshment policies

Refreshment policies are designed to balance the overhead cost of additional validation requests to origin servers against the increase in freshness hits. The decision of which objects to renew upon expiration varies between policies and is guided by natural properties of the request history of each object such as time-to-live (TTL) values, popularity, and recency of previous requests. Refreshment policies can also be viewed as prefetching of freshness. Their methodology and implementation, however, is closer to replacement policies than of prefetching algorithms. Refreshment policies resemble common cache replacement policies (such as LRU and LFU) in the way objects are prioritized. First, policies prefer renewing recently- or frequently-requested objects. Second, implementation is similar since object value is determined only by the request-history of the object rather than by considering request history of related objects. Another difference of refreshment and document prefetching is that validations typically have considerably smaller response-sizes than complete documents, but due to communication overhead, the latency gap is not nearly as pronounced. Hence, refreshment potentially provides considerably better tradeoffs of bandwidth vs. rreduced latency compared to object prefetching.

The experimental study indicates that the best among the refreshment policies considered can eliminate about half of the freshness misses at a cost of 2 additional validation requests per eliminated freshness miss. Since freshness misses constitute a large fraction (30%-50%) of cache hits in a typical large scale Web cache, incorporating a refreshment policy can considerably improve latency.

Policies

Refreshment policies associate with every cached object a *renewal credit* (a nonnegative integer). When a cached copy is about to expire (according to its respective TTL interval), and it has nonzero renewal credit, a renewal request is sent to the respective authoritative server, and the renewal credit is decremented.

The association of renewal credits to objects is governed by the particular policy. The policies we consider may increment the renewal credit only upon a client request. Renewal scheduling can be implemented via a priority queue, grouping together objects with close-together expiration, or possibly by an event-triggered process.

- PASSIVE: passive validation, objects are validated only as a result of a freshness miss i.e. when a request arrives and there is a stale cached copy. This is the way most caches work today.

- OPT(i): An approximation to the optimal omniscient offline policy. This policy assumes knowledge of the time of the subsequent request for the object and whether the object copy would still be valid then. If the subsequent request is such that the current copy remains valid and it is issued within $c \leq i$ freshness-lifetime-durations after the expiration of the current copy, then the renewal credit is set to c. Otherwise, no renewals are performed. This policy viewed as analogous to Belady's cache replacement algorithm [Belady, 1966]. The approximation collapses to the optimal solution if renewals can be granted for fractional TTL values.

- RECENCY(k): The renewal credit is reset to k following any request for the object, including no-cache requests. RECENCY(k), similarly to the cache replacement policy LRU, exploits the *recency property* of request sequences, which states that future requests are more likely to be issued to recently-requested objects.

- FREQ(j, m): increment the renewal credit by j for any request that would have been a freshness miss under PASSIVE. In other words, we add j to the renewal credit upon any request which is issued more than freshness-lifetime-duration units of time after a previous request that caused the passive policy to contact the origin server. In addition, upon any request (except for no-cache requests) the renewal credit is set to m if it is less than m. With $m = 0$, this policy FREQ$(j, 0)$ purely exploits the *frequency property* that states that objects that were frequently requested in the past are more likely to be requested soon. (A cache replacement policy that exploits the frequency property is LFU.) For $m > 1$, the policy FREQ(j, m) is a hybrid of FREQ$(j, 0)$ and RECENCY(m).

- TH-FREQ(th, m) keep renewing objects until the ratio of would-have been PASSIVE freshness misses to number of freshness-lifetime-durations since beginning of log drops below a threshold. In other words, upon each request which would have been a freshness miss for PASSIVE we increment the renewal credit such that we would keep renewing until the ratio drops below a threshold. In addition, upon any request (except for no-cache requests) the renewal credit is increased to m if it was less than m. This policy exploits the frequency property, and normalizes it by freshness-lifetime-duration. It also naturally provides a more continuous range of tradeoffs, since th is not necessarily an integer. With $m = 0$ the policy is purely frequency-based whereas higher values of m correspond to hybrids with RECENCY(m) policy.

Simulation results

Simulations were performed using 6 days NLANR cache traces [ircache, 2001] collected from the UC and SD caches from January 20th till January 25th, 2000. The NLANR caches run Squid [Squid, 2001] which logs and labels each request with several attributes such as the request time, service time, cache action taken, the response code returned to the client, and the response size.

Under the simulated baseline policy PASSIVE (where objects are refreshed only as a result of client requests), 48% of content hits constituted freshness misses on the UC trace, and 53% were freshness misses on the SD trace. The performance of the different refreshment policies is evaluated by the tradeoff of *overhead* vs. *coverage*. The coverage (reduction in freshness-misses) is calculated as the fraction of the number of freshness misses according to passive that are eliminated by the respective policy. More precisely, if x denotes the number of freshness misses of PASSIVE, and y the number of freshness misses of a policy P then the coverage of P is the fraction $(x-y)/x$.

The *overhead* of policy P is the difference between the number of validation requests issued by P and the number of validation requests issued by PASSIVE. Since the cache issues a validation request for each freshness miss, the request overhead is the total number of renewals performed minus the number of freshness misses eliminated (converted to freshness hits). The overhead is normalized by dividing it with the total number of freshness misses that were converted to freshness hits.

The coverage/overhead tradeoff for each type of policy is obtained by sweeping the value of its respective parameter. For example, the points on the curve of RECENCY correspond to runs with RECENCY(1), RECENCY(2), RECENCY$(3), \ldots$ and the points for FREQ were obtained on runs with FREQ$(1,0)$, FREQ$(2,0), \ldots$. Note that OPT(0), RECENCY(0), FREQ$(0,0)$, and TH-FREQ$(\infty,0)$ are in fact PASSIVE (these policies are able to achieve "continuous" tradeoffs by mixing two consecutive integral values of the respective parameter, for example, applying RECENCY(1) on some URLs and RECENCY(2) on others.) These tradeoffs are shown in Figure 1.2.

Under all types of policies, the coverage peaks at about 63%-67%. The remaining 33%-37% of freshness misses mostly occur on objects with freshness-lifetime-duration of 0, on which refreshment is not effective. The OPT policy eliminates all "addressable" freshness misses with overhead of about 1.3 requests per miss, and eliminates the bulk of these misses with an overhead of 0.5.

The simulation results for OPT and RECENCY show the locality effect, where most freshness misses that can be eliminated occur within a small number of freshness-lifetime-durations after a previous request. This goes well with the

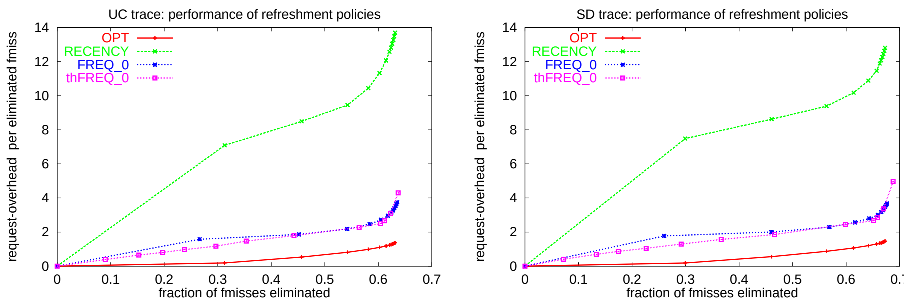
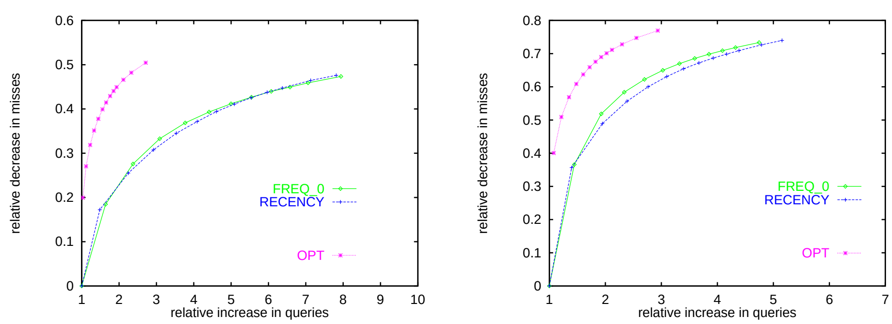


Figure 1.2. Performance of the different refreshment policies when simulated on the UC and SD traces



AT&T log UC NLANR log

Figure 1.3. Performance of refreshment policies on DNS traces. The x-axis gives the increase in the number of DNS queries due to refreshments.)

refreshment framework which keeps objects fresh continuously till the following "hit."

The left-most point on the curves of RECENCY and OPT, which correspond to RECENCY(1) and OPT(1), show that about 30% of freshness misses occur within one freshness-lifetime-duration after the expiration of the cached copy. The second points on the left correspond to RECENCY(2) and OPT(2) and indicate that about additional 15% of freshness misses occur between one and two freshness-lifetime durations passed the expiration. We note that the observed fact that a very small number of freshness misses occur more than 10 freshness-lifetime-durations passed the expiration is not only due to locality but also reflects the interaction of the log duration of six days and the most common freshness-lifetime-duration of 24 hours [Cohen and Kaplan, 2001a].

The fact that coverage of RECENCY and the frequency-based policies peaks at about the same place indicates that a very small fraction of freshness misses are incurred on very infrequently requested objects (since the frequency-based policies do not perform any renewals on the first request and thus can not eliminate misses incurred on the second request.). The correspondence in peak coverage of OPT and other policy-types is due to a "threshold phenomenon" where most freshness misses occur on objects with a freshness-lifetime-duration of 0 or occur within a small number of freshness-lifetime-durations following a previous request.

The frequency-based policies FREQ$(j, 0)$ and TH-FREQ$(th, 0)$ significantly outperformed RECENCY(k). This reflects the fact that the vast majority of freshness misses which can be eliminated occur on the more popular URLs. The gap is caused by the very large number of cachable URLs that were requested only once. The RECENCY(k) policy performed up to k unproductive renewals on each such request. (It could be less than k since we did not perform renewals passed the termination time of the log.) The hybrid policies FREQ and TH-FREQ with $m > 0$ performed considerably worse than the pure frequency-based policies (that correspond to $m = 0$) and hence only results for $m = 0$ are shown. This behavior is not surprising given that RECENCY yielded much worse tradeoffs. Overall, the results indicate that frequency-based object prioritization is more effective than recency-based prioritization.

The domination of frequency-based policies is also consistent with studies of cache replacement policies for Web contents [Breslau et al., 1999, Cohen and Kaplan, 1999], since different URLs tend to have a wide range of characteristic popularities, a property that is captured better by frequency-based policies. It is interesting to contrast the performance of refreshment policies on DNS records (see Figure 1.3 and [Cohen and Kaplan, 2001c] for more details.) For DNS caches, the recency and frequency-based policies exhibited similar performance. We explain this by the observation that at the hostname level there is significantly smaller fraction of "objects" that are resolved only once.

The performance of FREQ and TH-FREQ is similar, although TH-FREQ, which normalizes the frequency by freshness-lifetime-duration, performed somewhat better. The similarity is mostly explained by the fact that the large majority of freshness-lifetime-durations are of the same length (one day) and also because for shorter durations, frequency of requests is correlated with the duration. The policy TH-FREQ provides a spectrum of tradeoffs and better performance, particularly in the low-overhead range. TH-FREQ, however, may require more book-keeping than FREQ. The particular tradeoffs obtained by the frequency-based policies shows that significant improvements can be obtained with fairly low overhead. About 10% of freshness misses can be eliminated with the overhead of half of a validation request per eliminated freshness miss; 25% of freshness misses can be eliminated with overhead of a single request per eliminated miss; 50% can be eliminated with overhead of two; and 65% of freshness misses can be eliminated with overhead of three.

For DNS caches, the data consisted from Web proxy logs, extrapolated DNS queries, and performed name resolutions to extract the corresponding TTL (freshness lifetime) durations (see [Cohen and Kaplan, 2001c] for more details). Figure 1.3 shows the performance of refreshment policies on two logs. For each policy we measure the tradeoff between the relative decrease in cache misses and the relative increase in DNS queries (issued by the name-server). As the natural baseline we use the passive policy, whose performance on the two logs is as follows. The UC NLANR log had 941K DNS misses (which were 8.7% from the total number of HTTP requests). Out of these misses, 9.7% were on the first logged request to an item. The AT&T log had 32K misses, that were 6.5% of the total number of logged HTTP requests. 33% of these DNS misses were on the first logged request to an item. The number of queries issued by the passive policy is equal to the number of DNS misses. Better tradeoffs were obtained for the larger (NLANR) log because the fraction of necessary misses that occur on the first request to an item decrease with log period and with the number of clients. On the UC NLANR log, RECENCY and FREQ eliminate about 60% of misses that occurred on previously-seen items with query overhead of 2 and eliminate 80% of these misses with query overhead of 5. The respective reductions for the AT&T log are 36% and 63%.

5. Basic Source Types of Cascaded caches

We define three types of *sources* that capture different relationships between a client-cache and its data source(s). The different sources are illustrated in Figure 1.4. The TTL value obtained through each source as a function of time is illustrated in Figure 1.5.

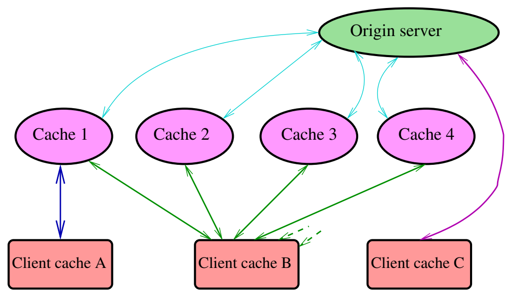


Figure 1.4. Different types of sources: Cache A uses cache 1 as an EXC source. Cache B uses caches 1,2,. . . as an IND source. Cache C uses an AUTH source.

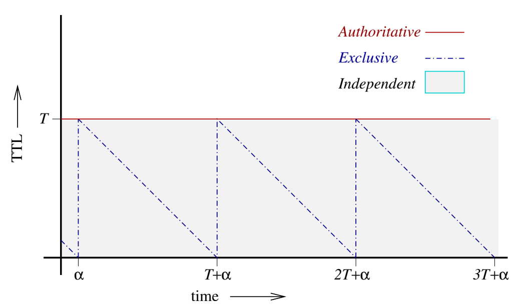


Figure 1.5. TTL obtained through different types of sources. AUTH source always provides zero age and TTL of T. EXC source (shown with displacement α) provides age that cycles from 0 to T and thus TTL which cycles from T to 0. IND source provides at each point in time age drawn independently from $U[0, T]$ and thus TTL drawn from $U[0, T]$.

1 AUTH: An authoritative source. AUTH always provides a copy with zero age (TTL that equals the freshness lifetime).

2 EXC: Let α be drawn uniformly at random from the interval $[0, T]$ (the distribution $U[0, T]$), where T is the lifetime of the object. We call α the *displacement*. At time t an EXC source provides a copy whose age is $(t - \alpha)$ mod$T$ ("mod" is a generalized modulus operation to arbitrary nonnegative numbers a mod $b = a - b * \lfloor a/b \rfloor$. If $b = 0$ then we define a mod $b \equiv 0$.) TTL equals to $T - (t - \alpha)$ mod T. Note that the choice of α is fixed for each "run" on the sequence, and the performance of EXC is the *expected* performance over all runs with different displacements.

   This source models a scenario where the client-cache upon each miss fetches a copy from the *same* high-level-cache. This high-level-cache maintains a fresh copy by refreshing it through an AUTH source each time it expires. The averaging over different displacements captures independence of the request times at the client-cache and the particular "refresh period" of the high-level-cache.

3 IND: Upon each miss at the client-cache, the IND source provides a copy with age independently drawn from $U[0, T]$ (thus a TTL that is also drawn from $U[0, T]$).

   This source models a scenario where upon each miss, the client-cache forwards the request to a different independent EXC-type high-level-cache. Independence means that the displacements of the different high-level-caches are not correlated.

The EXC, IND, and AUTH sources model pure scenarios. Currently, many web caches direct requests for a particular URL through a high-level-cache (e.g., a proxy or a reverse proxy) and hybrids of the IND and EXC sources capture scenarios where several caches are used for load balancing purposes.

We use the notion of source *consistency*. A source is *consistent* if the age provided at any given time is consistent with past values. Formally, if at time t, the source provided an age of Φ then at time $t + \Delta$ the source provides the object with age of at most $\Phi + \Delta$. The AUTH and EXC sources are consistent. In general, we expect sources that model a single server to be consistent.

Our analysis assumes that the request sequence is independent of the performance of the client-cache (that is, future requests do not depend on which previous requests constituted a hit or a miss).

Relationships Between the Basic Sources

We summarize results on basic relationships of the performances of a client cache through the different sources.

We first present proofs that on any request sequence

- The miss rate of a client-cache through an AUTH source is no greater than the miss rate of the client-cache through an EXC source, and
- The miss rate of a client-cache through an EXC source is no greater than the miss rate of the client-cache through an IND source.

The proof utilizes the following two basic lemmas that relate the miss rate at a client cache to the distribution of ages (and therefore TTLs) provided by the source. These lemmas will be also useful in later sections. (The proof of the second Lemma is omitted).

LEMMA 1.1 *Consider two consistent sources s_1 and s_2 such that at any given point in time, the TTL available from s_1 is at least as large as the TTL available from s_2. Then, for any sequence of requests, the number of misses through s_1 is at most the number of misses through s_2.*

Proof. The proof is by induction on the length of the request sequence. The lemma clearly holds for sequences of lengths 1 and 2. Consider a request sequence τ of length l. If the last request is either a miss or a hit through both s_1 and s_2, or a miss through s_2 but a hit through s_1, then the lemma follows by using the induction hypothesis on the prefix of length $l-1$ of τ.

Consider the case where the last request is a miss through s_1 but a hit through s_2. Let the ith request be the last request on which the client working through s_2 had a miss. The client working through s_1 must have had a hit on the ith request and all the requests following it including the next to last request. (Otherwise our assumptions guarantee that it would have got a TTL that covers the last request.) Thus by applying the induction hypothesis on the prefix of τ of length $i-1$ the lemma follows. ■

LEMMA 1.2 *Consider two sources s_1 and s_2 that serve objects with TTLs that are independently drawn from distributions with CDFs (Cumulative Distribution Functions) F_1 and F_2, respectively. Suppose that for all $x \geq 0$, $F_1(x) \leq F_2(x)$, i.e. s_1 provides longer TTLs. Then for any sequence of requests, the expected number of misses through the source s_1 is no larger than the expected number of misses through the source s_2.*

As a consequence of Lemma 1.1 we obtain that AUTH has smaller miss rate than EXC. Furthermore, as a consequence of Lemma 1.2 we obtain that AUTH has smaller miss rate than IND. The following Lemma establishes that EXC always outperforms IND.

LEMMA 1.3 *For any sequence of requests, the miss rate through an* EXC *source is never higher than through an* IND *source.*

Proof. Consider the ith request. Its likelihood of being a miss with EXC is $\min\{1, (t_i - t_{i-1})/T\}$. Let $m[j]$ and $h[j]$ denote the events that the jth request is a miss or a hit respectively, with an IND source. The likelihood that the ith request constitutes a miss with IND is

$$\begin{aligned} p(m[i]) &= p(m[i-1]) \min\{1, \frac{t_i - t_{i-1}}{T}\} \\ &\quad + p(h[i-1] \cap m[i-2]) \min\{1, \frac{t_i - t_{i-2}}{T}\} \\ &\quad + p(h[i-1] \cap h[i-2] \cap m[i-3]) \min\{1, \frac{t_i - t_{i-3}}{T}\} + \dots \\ &\geq \min\{1, \frac{t_i - t_{i-1}}{T}\} \,. \end{aligned}$$

Since the likelihood of every request to be a miss through an EXC source is smaller than through an IND source, we obtain that the miss rate through EXC is smaller than the miss rate through IND. ■

Note that with respect to one fixed displacement value α, the EXC source could perform worse than IND. Consider the following sequence where requests are made at times $(2i - \epsilon)T$ and $(2i + \epsilon)T$ for integral $i > 0$. Suppose that the source refreshes at times iT. Then all requests would constitute misses with EXC, and only $1/2 + 2\epsilon$ would constitute misses with IND. Lemma 1.3 shows that on average over all displacements, EXC performs at least as well as IND.

6. Rejuvenating Sources

We next consider replicating servers that refresh selected objects as soon as their TTL drops below some threshold (rather than wait for it to completely expire). We refer to such configured periodic pre-term refreshes as *rejuvenation*.

We extend the definitions of our basic sources to include rejuvenation. The source EXC_v is an EXC source that refreshes its copy of the object when the age exceeds v fraction of the lifetime value. Formally, let α be drawn from $U[0, vT]$. At time t, an EXC_v source return the object with age $(t - \alpha) \bmod (v * T)$ (so the TTL is $T - (t - \alpha) \bmod (v * T)$). As with an EXC source, α is fixed for a "run", and performance is the expected performance over runs with different displacements. We say that a client-cache uses an IND_v source if upon each miss it forwards the request to a different independent EXC_v source. Hence, IND_v source returns copies with age drawn from $U[0, vT]$ and thus TTL drawn from $U[(1 - v)T, T]$. The TTL as a function of time for the different sources is illustrated in Figure 1.6. For both IND_v and EXC_v sources, a rejuvenation interval of $v = 1$ corresponds to the respective pure source: $\text{EXC}_1 \equiv \text{EXC}$ and $\text{IND}_1 \equiv \text{IND}$. A rejuvenation interval of $v = 0$ corresponds to a pure AUTH source. That is, $\text{EXC}_0 \equiv \text{AUTH}$ and $\text{IND}_0 \equiv \text{AUTH}$.

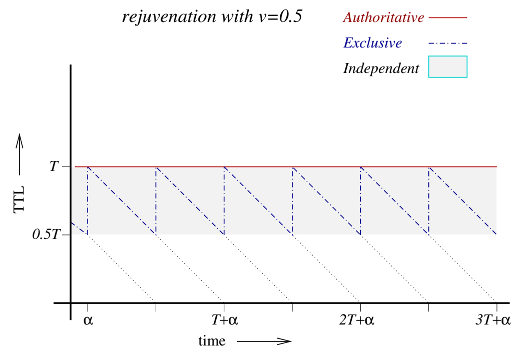


Figure 1.6. TTL as a function of time for rejuvenating sources with $v = 0.5$. $\text{EXC}_{0.5}$ has a period of $0.5T$ where the age cycles from $T/2$ to T and thus, the TTL cycles from T to $T/2$. For $\text{IND}_{0.5}$ the age at any point in time is independently drawn from $U[T/2, T]$ and thus 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.

Intuitively, we might expect a monotonic improvement in miss rate as v decreases from $v = 1$ to $v = 0$. We show later on that this is the case with IND_v sources but not with EXC_v sources.

Does Rejuvenating Pay Off?

At first glance, it seems that rejuvenations can only reduce the miss rate of client caches. Somewhat surprising is the fact that this is not the case for EXC rejuvenating sources.

LEMMA 1.4 *Let $v < 1$, and let $u = \frac{1}{\lceil 1/v \rceil}$. Note that $v \le u \le 1$. Consider a request sequence $\{t_i\}$, such that the object is requested at least once every $(\lceil \frac{1}{v} \rceil v - 1)T$ time units. Let m_v denote the miss rate through EXC_v on $\{t_i\}$ and m_u the miss rate through EXC_u on $\{t_i\}$. Then, $m_v = \dfrac{u}{v} m_u$.*

Lemma 1.4 shows that the performance of a cache which receives frequent requests to the object can be strictly worse through EXC_v with $v < 1$ than through a non-rejuvenating EXC source. For example, for $v > 0.5$, and for sequences satisfying that the object is requested at least once every $(2v - 1)T$ time units, by Lemma 1.4 the miss rate is strictly worse through EXC_v than with a non-rejuvenating EXC source. In this case, rejuvenating does not pay off.

In contrast, the following lemma shows that IND sources do exhibit monotonic dependence of the miss rate on v.

LEMMA 1.5 *Let IND_{v_1} and IND_{v_2} be two sources such that $v_1 \le v_2$. Then, the miss rate through IND_{v_1} is no larger than the miss rate through IND_{v_2}.*

Proof. By the definition of IND_v the TTL's through IND_{v_1} are drawn from $U[(1-v_1)T, T]$ and the TTL's through IND_{v_2} are drawn from $U[(1-v_2)T, T]$. Let F_1 be the CDF of $U[(1-v_1)T, T]$ and let F_2 be the CDF of $U[(1-v_2)T, T]$. Since for all $x \geq 0$, $F_1(x) \leq F_2(x)$ Lemma 1.2 implies that the miss rate through IND_{v_1} is no larger than the miss rate through IND_{v_2}. ■

The following corollary of Lemma 1.1 shows that although generally rejuvenation does not always improve the performance, rejuvenation cannot degrade performance on any sequence if the source is *synchronized* with the client cache. (The source and client-cache are synchronized if after the client-cache contains a copy of the object which expires at some time t, requests directed to the source at times $t + \Delta$ (for any $\Delta > 0$) obtain a copy whose age is not more than Δ.)

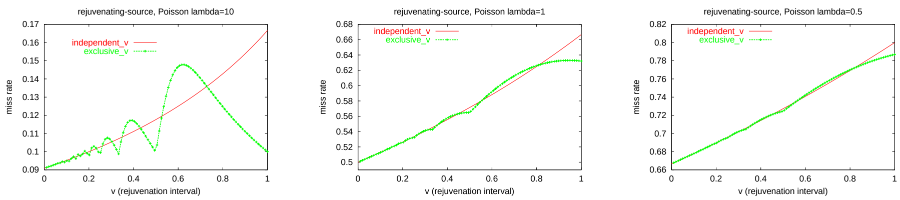
COROLLARY 1.6 *Suppose a rejuvenating* EXC *source adheres to the original refresh schedule, refreshing the object at times $\alpha + iT$ for integral i in addition to possibly rejuvenating it at other points in time. Then on any sequence of requests, the number of misses is not higher than the number of misses through* EXC.

In particular it follows from Corollary 1.6 that the performance through EXC_v with integral $1/v$, (i.e., $v = 1/2, 1/3, \ldots$), is at least as good as through EXC. Moreover, even if we restrict ourselves to EXC_v, with integral $1/v$, the miss rate is not always monotonic as a function of v. The following example shows that EXC_v $(7/8 > v > 3/4)$ can induce miss-rate even higher than through the basic IND source:

EXAMPLE 1.7 *Consider the source EXC_v with $v = 3/4+x$ (for fixed $0 < x < 1/4$). Consider the sequence of requests to the client-level cache at regular intervals of $1/2 + 2x$. Since there is at least one request every $(2v-1)T$ time, there is exactly one miss of the client-level cache every vT time. Consider now the performance of* IND. *With each miss we associate the time period till the next miss. We compute the expected length of that interval between misses. The interval length is $(1/2 + 2x)$ with probability $(1/2 + 2x)$ and is $1 + 4x$ otherwise. Thus, the expected length of the interval is $3/4 + 2x - 8x^2$. Note that $2x - 8x^2 > x$ for $0 < x < 1/8$. Hence,* IND *source would outperform EXC_v in this case.*

We saw 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.

Poisson arrivals We now consider the performance of a client-cache that receives requests generated by a Poisson process with rate λ (we normalize the rate by the freshness lifetime interval T, so the expected number of requests in



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

Figure 1.7. Miss rate dependence on rejuvenation interval for different Poisson arrivals rates.

time T is λ). The client cache contacts a rejuvenating source upon a miss. The miss rate as a function of v and λ is

- $\frac{1}{1+(2-v)\lambda/2}$, through an IND_v source, and

- $\frac{1}{\lambda v(\lfloor 1/v \rfloor + \exp(\lambda v(1/v - \lfloor 1/v \rfloor))/(\exp(-v\lambda)-1))}$, through an EXC_v source.

Figure 1.7 illustrates dependence of miss-rate on v, for 3 representative request rates, for a cache using EXC_v and IND_v sources.

Pareto arrivals The *Pareto* family of distributions is a widely-used model for inter-arrival times [Leland et al., 1993, Paxson and Floyd, 1995]. 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 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. 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 $1/T$). 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 1.8 shows simulation results for some representative "request-rates" and values of the power α.

Trace-based simulations Figure 1.9 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 NLANR trace mentioned in Section 4. The simulations show the presence of patterns obtained in our Poisson analysis.

Elementary derivations establish the following for Poisson arrivals, and simulations suggest that these patterns hold for Pareto arrivals and real traces as well.

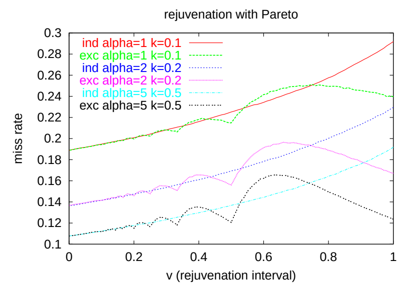


Figure 1.8. Rejuvenation with Pareto arrivals

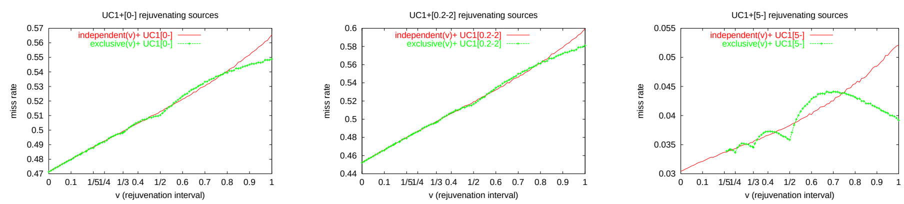


Figure 1.9. Miss rate dependence on rejuvenation interval for UC trace, UC$[0.2$–$2]$, and UC$[5$–$\infty]$. The notation UC$[a$–$b]$ includes all requests made to objects with rate in $[a, b)$.

- 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. We note that even though we observed this for Pareto arrivals and in our traces, this is not universally true: There are sequences on which the miss-rate with $\text{EXC}_{1/2}$ is strictly lower than with $\text{EXC}_{1/3}$.

Worst-case analysis

We summarize some worst-case bounds on the performance of EXC and IND rejuvenating sources. For a source S and a request sequence $\{t_i\}$ let $\rho(\{t_i\}, S)$ be the ratio between the expected miss rate of $\{t_i\}$ through S and through an

AUTH source. The maximum over all sequences $\{t_i\}$ of $\rho(\{t_i\}, S)$ is called the *competitive ratio* of S, and is denoted by $\rho(S)$. We use the term *rejuvenating policy* for any source that corresponds to a single high-level cache which maintains a fresh copy of the object by refreshing it (rejuvenating) from an authoritative server before or when it expires. In particular AUTH, EXC, and EXC_v, are rejuvenating policies. We start witha bound on the competitive ratio of any rejuvenating policy:

LEMMA 1.8 *The competitive ratio of any rejuvenating policy is at most 2.*

The following Theorem establishes a corresponding lower bound of 2 on the competitive ratio of any rejuvenating policy that cannot rejuvenate the object continuously but must have a small gap of ϵ between consecutive rejuvenations.

THEOREM 1.9 *Consider a rejuvenating policy P such that the time interval between any two rejuvenation points must exceed some $\epsilon > 0$. Then $\rho(P) = 2$.*

The Theorem implies that rejuvenating in fixed intervals, (an EXC_v source with $v > 0$), has a competitive ratio 2.

Another related problem is to compare the worst-case performance of an IND_v source with respect to an AUTH source:

THEOREM 1.10 *The competitive ratio of* IND_v *is* $\displaystyle\sum_{k=0}^{\lfloor \frac{1}{1-v} \rfloor} \frac{(1-k(1-v))^k}{k! v^k}$.

By Theorem 1.10, for $v < 1/2$, the competitive ratio of an IND_v source is 2, and as v approaches 1, the competitive ratio approaches $e \approx 2.718$. The higher competitive ratio of IND_v comparable to the competitive ratio of EXC_v fits well with the superiority of EXC on IND established earlier.

These worst-case bounds do not distinguish between the performance of sources with different rejuvenation rates, as all EXC_v sources $(0 < v \leq 1)$ and all IND_v sources $(0 < v \leq 1/2)$ have the same tight competitive ratio of 2.

Lessons

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 inferior performance of a client-cache relative to its performance under a basic EXC source.

Synchronization between a client and its 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 serves sporadic `no-cache` requests can preserve synchronization with its other clients by performing 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 1.1). This tradeoff should guide the selection of rejuvenation interval or the follow-up action on a sporadic pre-term refresh.

The objective of rejuvenation policies is to maximize the benefit, that is, minimize the total number of misses at client-caches given some bound on the cost, that is crudely measured by the number of unsolicited refresh requests issued by the high-level cache. 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. A challenge left for future work is to efficiently estimate this benefit, possibly by keeping some summarized per-client history for a sample of the clients.

References

Akamai (2001). Akamai. `http://www.akamai.com`.

Banga, G., Douglis, F., and Rabinovich, M. (1997). Optimistic deltas for WWW latency reduction. In *Proceedings of the USENIX Annual Technical Conference*. USENIX Association.

Belady, L. A. (1966). A study of replacement algorithms for virtual storage computers. *IBM systems journal*, 5:78–101.

Berners-Lee, T., Fielding, R., and Frystyk, H. (1996). RFC 1945: Hypertext Transfer Protocol — HTTP/1.0.

Bestavros, A., Carter, R., Crovella, M., Cunha, C., Heddaya, A., and Mirdad, S. (1995). Application-level document caching in the internet. In *Proceedings of the Second Intl. Workshop on Services in Distributed and Networked Environments (SDNE '95)*. Available as http://cs-www.bu.edu/faculty/crovella/paper-archive/www-sdne95/paper.html.

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM'99 Conference*.

cacheman (2001). IBM WebSphere Cache Manager. `http://www.software.ibm.com/webservers/cacheman`.

Cao, P. and Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California. `http://www.usenix.org/events/usits97`.

Cao, P. and Liu, C. (1998). Maintaining strong cache consistency in the world wide web. *IEEE Transactions on Computers*, 47(4):445–457.

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

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

Cohen, E. and Kaplan, H. (2001a). The age penalty and its effect on cache performance. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*.

Cohen, E. and Kaplan, H. (2001b). Aging through cascaded caches: performance issues in the distribution of Web content. In *Proceedings of the ACM SIGCOMM Conference*. ACM.

Cohen, E. and Kaplan, H. (2001c). Proactive caching of DNS records: addressing a performance bottleneck. In *Proceedings of the Symposium on Applications and the Internet*. IEEE.

Cohen, E. and Kaplan, H. (2002). Refreshment policies for Web content caches. *Computer Networks*, 38:795–808.

Cohen, E., Krishnamurthy, B., and Rexford, J. (1998a). Evaluating server-assisted cache replacement in the Web. In *Proceedings of the 6th European Symposium on Algorithms*, pages 307–319. Springer-Verlag, Lecture Notes in Computer Science Vol. 1461.

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

Dingle, A. and Partl, T. (1996). Web cache coherence. In *Proceedings of the Fifth International World Wide Web Conference*, pages 907–920. Computer Networks and ISDN Systems 28:7-11.

Duvvuri, V., Shenoy, P., and Tewari, R. (2000). Adaptive leases: a strong consistency mechanism for the World Wide Web. In *Proceedings of the 19th IEEE INFOCOM Conference*.
`http://www.ieee-infocom.org/2000/papers/`.

Feldmann, A., Cáceres, R., Douglis, F., Glass, G., and Rabinovich, M. (1999). Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of the IEEE INFOCOM'99 Conference*.

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

Gray, C. and Cheriton, D. (1989). Leases: an efficient fault tolerant mecahnism for distributed file cache consistency. In *Proceedings of the 12th ACM symposium on operating systems principles*, pages 202–210.

Hou, Y., Pan, J., Li, B., Tang, X., and Panwar, S. (2002). Modeling and analysis of an expiration-based hierarchical caching system. In *Proceedings of IEEE Globecom Internet Performance Symposium*. IEEE.

Hou, Y., Pan, J., Wang, C., and Li, B. (2003). On prefetching in hierarchical caching systems. In *Proceedings of IEEE ICC Global Services and Infrastructure for Next Generation Networking Symposium*. IEEE.

Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. (1988). Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81.

Inktomi (2001). Inktomi Traffic Server.
`http://www.inktomi.com`.

InktomiCDS (2001). Inktomi Content Delivery Suite.

ipworx (2001). Lucent IPWorX.
`http://www.lucentipworx.com`.

ircache (2001). A Distributed Testbed for National Information Provisioning.
`http://www.ircache.net`.

Jung, J., Berger, A., and Balakrishnan, H. (2003). Modeling TTL-based internet caches. In *Proceedings of the IEEE Infocom*.

Jung, J., Sit, E., Balakrishnan, H., and Morris, R. (2001). DNS performance and the effectiveness of caching. In *Proc. ACM SIGCOMM Internet Measurement Workshop*.

Krishnamurthy, Balachander and Wills, Craig E. (1997). 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.

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

Li, D. and Cheriton, D. R. (1999). Scalable web caching of frequently updated objects using reliable multicast. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 1–12.

Mockapetris, P. (1987a). Domain names – concepts and facilities. RFC 1034, ISI.
`http://www.dns.net/dnsrd/rfc/rfc1034/rfc1034.html`.

Mockapetris, P. (1987b). Domain names – implementation and specification. RFC 1035, ISI.
`http://www.dns.net/dnsrd/rfc/rfc1035/rfc1035.html`.

Mogul, J. C. (1999). Errors in timestamp-based HTTP header values. Technical Report 99/3, Compaq Western Research Lab.
`http://www.research.digital.com/wrl/techreports/abstracts/99.3.html`.

Nottingham, M. (1999). Optimizing object freshness controls in Web caches. In *The 4th International Web Caching Workshop*.
`http://www.ircache.nlanr.net/Cache/workshop99/Papers`.

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

Squid (2001). Squid internet object cache.
`http://squid.nlanr.net/Squid`.

Williams, Stephen, Abrams, Marc, Standbridge, Charles R., Abdulla, Ghaleb, and Fox, Edward A. (1996). Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM Conference*, pages 293–305.

Yin, J., Alvisi, L., Dahlin, M., and Lin, C. (1999). Volume leases for consistency in large-scale systems. *IEEE transactions on knowledge and data engineering*.

Young, N. (1998). On line file caching. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM.