ABSTRACT

Title of dissertation: OPTIMIZATION AND EVALUATION
OF SERVICE SPEED AND RELIABILITY
IN MODERN CACHING APPLICATIONS

Omri Bahat, Doctor of Philosophy, 2006

Dissertation directed by: Professor Armand M. Makowski
Department of Electrical and Computer Engineering
and the Institute for Systems Research

The performance of caching systems in general, and Internet caches in particular, is evaluated by means of the user-perceived service speed, reliability of downloaded content, and system scalability. In this dissertation, we focus on optimizing the speed of service, as well as on evaluating the reliability and quality of data sent to users.

In order to optimize the service speed, we seek optimal replacement policies in the first part of the dissertation, as it is well known that download delays are a direct product of document availability at the cache; in demand-driven caches, the cache content is completely determined by the cache replacement policy. In the literature, many ad-hoc policies that utilize document sizes, retrieval latency, probability of references, and temporal locality of requests, have been proposed. However, the problem of finding optimal policies under these factors has not been pursued in any

systematic manner. Here, we take a step in that direction: Still under the Independent Reference Model, we show that a simple Markov stationary policy minimizes the long-run average metric induced by non-uniform documents under optional cache replacement. We then use this result to propose a framework for operating caches under multiple performance metrics, by solving a constrained caching problem with a single constraint.

The second part of the dissertation is devoted to studying data reliability and cache consistency issues: A cache object is termed consistent if it is identical to the master document at the origin server, at the time it is served to users. Cached objects become stale after the master is modified, and stale copies remain served to users until the cache is refreshed, subject to network transmit delays. However, the performance of Internet consistency algorithms is evaluated through the cache hit rate and network traffic load that do not inform on data staleness. To remedy this, we formalize a framework and the novel hit* rate measure, which captures consistent downloads from the cache. To demonstrate this new methodology, we calculate the hit and hit* rates produced by two TTL algorithms, under zero and non-zero delays, and evaluate the hit and hit* rates in applications.

OPTIMIZATION AND EVALUATION OF SERVICE SPEED AND RELIABILITY IN MODERN CACHING APPLICATIONS

by

Omri Bahat

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:

Professor Armand M. Makowski, Chair
Professor John S. Baras
Professor Richard J. La
Professor Bruce Jacob
Professor Eric V. Slud

DEDICATION

To Orit

Contents

List of Figures

Chapter 1

Introduction

1.1 Web caching

The use of caches to increase performance in distributed information systems dates to the earliest days of the computer industry. Caches were first introduced in virtual shared memories to lessen the disparity in performance between ever-faster central processing units and relatively slower main memories, e.g., see [70] and references therein for a detailed historical overview.

With time, principles and guidelines utilized in the design of early caches were extended to accomodate the operational requirements of modern data storage and content distribution systems, e.g., distributed file sharing systems [14, 58], databases [26, 30], and the World Wide Web [11, 17, 76]. Common to these systems is the hosting of data on potentially large number of *servers* in an ubiquitous manner, so that each data item is accessible by the *users* at all times. However, in spite of these similarities, the nomenclature used to characterize rules of engagement and various cache design challenges is unique to each system; the work presented in this dissertation is

therefore focused on the World Wide Web.

On the Internet, proxy caches contain replicas of "popular" documents and are strategically placed between servers and users for the purpose of reducing network traffic, server load, and user-perceived retrieval latency. To date, Web caching is the most productive approach to handling the ever-increasing number of Web users and volume of server objects, while maintaining good *service speed*, *scalability*, and *reliability of data*, which are demanded by Internet users and server administrators.

1.2 Quality of service (QoS)

The performance of Web caching systems is typically evaluated from two different viewpoints: System and network operators responsible for guaranteeing the uptime of Web servers and network communication links, are primarily concerned with load and scalability issues. Consequently, metrics such as traffic volume and number of accesses to the server are often used to measure the cache performance. On the other hand, these operational aspects are of little importance to the users.

From a user perspective, key to the effetiveness of Web caches is the ability to serve requests with recent (i.e., fresh) documents in a timely manner, as we consider the possibility that the content of Web pages might change over time. These two factors, namely speed of service and *quality of data*[1] (QoD), significantly affect the user-experience, and therefore have a profound bearing on the *quality of service* (QoS) of the cache.

[1]We assimilate object freshness with the quality and reliability of data.

1.3 Speed of service

A user request for a Web document is first presented at the cache. If the cache contains a copy of the requested item (i.e., *cache-hit*), then a copy is sent to the user by the cache without contacting the server. When the requested object is not found in the cache (i.e., *cache-miss*), the request is forwarded to the server, which then transmits the document back to the cache, and from there to the user. Caches that follow these operational rules are termed *demand-driven* caches, in contrast to *prefetching* caches, whereby download requests are proactively submitted to the server by the cache [75, 76].

Each time a *cacheable* document is received by the cache, a decision must be made to either store or discard the new download. If cache-placement is *determined*, the cache then invokes a *replacement policy* that identifies the set of documents (if any) to be evicted from the cache, in order to make room for the retrieved object. The (re)placement policy therefore provides the sole means of shaping the content of the cache, which is central to ensuring good service speed. This was previously reported in [2, 17, 36, 65, 66] and is now explained below.

In demand-driven caches, service speed (equivalently, download delay) is primarily affected by the location of the cache in the network [46, 49], the cache storage capacity, and the bandwidth between the users and the cache and between the server and the cache. However, under fixed network infrastructure and storage space, the speed of service is a direct product of document availability at the cache, as well as of indexing and allocation algorithms that impact the cache processing delays [19, 78]. Thus, key to improving the service speed is the implementation of replacement algorithms that can yield low download latencies. Motivated by this fact, we set our focus

in Part I of this dissertation on finding efficient (provably optimal) cache replacement policies under the assumptions that user requests are independent and identically distributed (i.i.d.), and that document retrieval costs (e.g., delays) are *not* uniform.

1.4 Reliability and quality of data

As in most content distribution systems, pages on the Web evolve over time to reflect the latest services, features, and data available at the server (e.g., pages on news portals and commercial Web sites). One key problem that arises in the context of updatable documents is the *staleness* of objects stored at the cache: Once the *master* document at the *origin* server is altered, the previously cached version of the document becomes obsolete, and remains so until the changes are propagated to the cache. User requests that arrive to the cache before it is informed of the master update are served with stale copies, in the process degrading the quality of the downloaded data.

In order to remedy this state of affairs, consistency algorithms are implemented either at the cache or at the server for the sole purpose of increasing the likelihood that documents served to users by the cache are identical to those offered at the server. Consistency protocols exchange control messages between the server and the cache, and compare each copy with its corresponding master; cached objects are marked as *invalid* in the event of a mismatch.

Through the invalidation of stale copies, consistency algorithms allow the cache to achieve higher quality of data and improve the reliability of content sent to users: If a cached copy is valid, each request presented at the cache incurs a cache-hit and receives the stored replica. Otherwise, requests are forwarded to the server to retrieve the latest document version, and the new document is loaded into the cache.

Concerns regarding the download of stale (i.e., inconsistent) Web objects were outlined in numerous studies, e.g., see [18, 31, 34, 40, 67, 73] and references therein. However, the performance of Internet consistency algorithms is typically evaluated through the corresponding cache hit rate and network traffic load; we refer the reader to [18, 21, 22, 23, 24, 31, 42, 54, 79] for a sample literature. These metrics do not inform on the service of stale data and are therefore inadequate for evaluating the cache consistency performance under a given protocol, as previously concluded in [42].

To date, neither an analytical framework nor a suitable metric are available to model the service of stale Web documents to users. These issues are addressed in Part II of the dissertation, where we propose a framework and measures for evaluating cache consistency. In this analytical model, document requests and master updates are modeled by mutually independent point processes on $[0, \infty)$. The novel *hit* rate* then counts the number of *consistent* (i.e., fresh) downloads out of all user requests, and can be used to quantitavely capture the QoD.

Part I

Finding Optimal Replacement Policies to Improve the Speed of Service

Chapter 2

Cache Replacement Policies

2.1 Conventional replacement algorithms

A review of the literature quickly reveals that a large number of methods for file caching and virtual memory replacement have been developed [2, 20]. Unfortunately, they do not transfer well to Web caching. In the context of these *conventional* caching techniques, the underlying working assumption is the so-called *Independent Reference Model* (IRM), whereby document requests are assumed to form an i.i.d. sequence. It has been known for some time [2, 20] that under the IRM the miss rate (respectively, the hit rate) is minimized (respectively, maximized) by the policy A_0 according to which a document is evicted from the cache if it has the smallest probability of occurence (respectively, is the least popular) among the documents in the cache.

In practice, the probability of document request is not available and thus needs to be estimated on-line as requests are coming in. This naturally gives rise to the *Least-Frequently-Used* (LFU) policy, which calculates the access frequency based on

trace measurements of user requests, and dictates the eviction of the least frequently referenced item. The focus on miss and hit rates as performance criteria reflects the fact that historically, pages in memory systems were of equal size, and transfer times of pages from the primary storage to the cache were nearly constant over time and independent of the document transferred.

Interestingly enough, even in this restricted context, the popularity information as derived from the relative access frequencies of objects requested through the cache, is seldom maintained and is rarely used directly in the design of cache replacement policies. This is so because of the difficulty to capture this information in an on-line fashion in contrast to other attributes of the request stream, said attributes being thought indicative of the future popularity of the object. Typical examples include temporal locality via the recency of access and object size, which lead very naturally to the *Least-Recently-Used* (LRU) and *Largest-File-First* (LFF) replacement policies, respectively.

2.2 Conventional versus Web caching

At this point it is worth stressing the three primary differences between Web caching and conventional caching:

1. Web objects or documents are of variable size whereas conventional caching handles fixed-size documents or pages. Neither the policy A_0 nor the LRU policy (nor many other policies proposed in the literature on conventional caching) account for the variable size of documents;

2. The miss penalty or retrieval cost of missed documents from the server to the

proxy can vary significantly over time and per document. In fact, the cost value may not be known in advance and must sometimes be estimated on-line before a decision is taken. For instance, the download time of a Web page depends on the size of the document to be retrieved, on the available bandwidth from the server to the cache, and on the route used. These factors may vary over time due to changing network conditions (e.g., link failure or network overload);

3. Access streams seen by the proxy cache are the union of Web access streams from tens to thousands of users, instead of coming from a few programmed sources as is the case in virtual memory paging, so the IRM is not likely to provide a good fit to Web traces. In fact, Web traffic patterns were found to exhibit temporal locality (i.e., temporal correlations) in that recently accessed objects are more likely to be accessed in the near future [74]. To complicate matters, the popularity of Web objects was found to be highly variable (i.e., bursty) over short time scales but much smoother over long time scales [3, 29, 36].

These differences, namely variable size, variable cost and the more complex statistics of request patterns, preclude an easy transfer of caching techniques developed earlier for computer memory systems.

2.3 Replacement policies on the Web

A large number of studies have focused on the design of efficient replacement policies; see [36, 37, 38, 39] and references therein for a sample literature. Proposed policies typically exploit either access recency (e.g., the LRU policy) or access fre-

quency (e.g., the LFU policy) or a combination thereof (e.g., the hybrid LRFU policy). The numerous policies which have been proposed are often ad-hoc attempts to take advantage of the statistical information contained in the stream of requests, and to address the factors above. Their performance is typically evaluated via trace-driven simulations, and compared to that of other well-established policies.

As should be clear from the discussion above, the classical set-up used in [2, 20] is too restrictive to capture the salient features present in Web caching: The IRM captures document popularity (i.e., long-term frequencies of requested objects), yet fails to capture temporal locality (i.e., correlations among document requests). It also does not account for documents with variable sizes. Moreover, this literature implicitly assumes that document replacement is *mandatory* upon a cache-miss, i.e., a requested document not found in cache must be put in the cache.

With these difficulties in mind it seems natural to seek provably optimal caching policies under the following conditions: (i) The documents have non-uniform costs (as we assimilate cost to size and variable retrieval latency), (ii) There exist correlations in the request streams, and (iii) Document placement and replacement are optional upon a cache-miss.

In this dissertation we take an initial step in the directions (i) and (iii): While still retaining the IRM, we consider the problem of finding an optimal replacement policy with *non*-uniform costs under the option that a requested document not in cache is not necessarily put in cache after being retrieved from the server. Interestingly enough, this simple change in operational constraints allows us to determine completely the structure of the optimal replacement policy for the minimum average cost criterion (over both finite and infinite horizons).

2.4 Finding good replacement policies

One approach for designing good replacement policies is to couch the problem as one of sequential decision making in the presence of uncertainty. The analysis that produced the policy A_0 mentioned earlier (and its optimality under the IRM) is one based on Dynamic Programming as developed in the framework of Markov Decision Processes (MDPs) [32, 68]. Here, we modify the MDP framework used in [2, 6, 20] in order to incorporate the possibility of optional eviction.

The system model is presented first in Chapter 3, where we assume [Section 3.1] that a total of N documents are available over all servers, and that at any given time the cache can hold upto M documents, with $M \leq N$. Under this model, we proceed to identify the space of allowable system states and the corresponding action space [Section 3.2], and define the probability measure associated with the MDP [Section 3.3].

A generic cost function $c(i) > 0$ denotes the penalty incurred by the system upon a miss request for document $i,\ i = 1, \ldots, N$. With this one-step cost penalty, we introduce the finite and infinite horizon cost functionals [Section 3.4], and show that these costs can be specialized to express commonly used cache performance metrics, such as the hit rate, byte hit rate, and average download latency.

In order to improve the speed of service (as well as other performance measures mentioned above), in Chapter 4 we seek an optimal replacement policy that minimizes the expected average cost over the entire horizon. To find it, we make use of standard ideas from the theory of MDPs, and formulate the Dynamic Program that corresponds to the problem at hand. Indeed, we propose the (simple) Markov stationary policy $C_0^\star$, and show that this policy is optimal under both the finite and inifinite

horizon cost criteria.

The main contribution of the first part of the dissertation is presented in Chapter 5: As in most complex engineering systems, multiple performance metrics need to be considered when operating caches, sometimes leading to conflicting objectives. For instance, managing the cache to achieve as small a miss rate as possible does not necessarily ensure that the average latency of retrieved documents is as small as could be, since the latter performance metric depends on the size of retrieved documents while the former does not. In order to capture this multi-criterion aspect we introduce constraints, and formulate the problem of finding a replacement policy that minimizes an average cost under a single constraint in terms of another long-run average metric. Utilizing a methodology developed in the context of MDPs with a constraint [13], we obtain the structure of the constrained optimal replacement policy by randomizing two simple policies of the type $C_0^\star$.

Chapter 3

The Cache Model

3.1 The system model

A user request is first presented at the cache. If the cache contains a copy of the requested document (i.e., a cache-hit), then it is sent to the user by the cache. Otherwise, the request is forwarded to the server, which in turn sends the document to the cache, and from there to the user. Given our primary focus on designing efficient replacement policies employed by individual caches, collaboration between different caches is ruled out, and the discussion is therefore restricted to a single cache in isolation.

3.1.1 User requests

A total of N distinct cacheable objects are available over all servers, labeled $i = 1, \ldots, N$, and let $\mathcal{N} = \{1, \ldots, N\}$ denote the universe of all system documents. For each $t = 0, 1, \ldots,$ the $\mathcal{N}$-valued rv R_t represents the t^{th} request presented at the cache. The stream of successive requests arriving at the cache is then captured by the

sequence of rvs $\mathbf{R} = \{R_t,\ t = 0, 1, \ldots\}$.

The *popularity* of requests in the sequence $\{R_t,\ t = 0, 1, \ldots\}$ is defined as the pmf $\mathbf{p} = (p(1), \ldots, p(N))$, where for each $i = 1, \ldots, N$, we denote by $p(i)$ the long-term probability that document i is referenced, namely

$$p(i) = \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} \mathbf{1}\left[R_t = i\right] \quad a.s., \tag{3.1}$$

whenever the limit exists. This limit indeed exists for all cases considered in our analysis, as outlined below. Under the additional (and natural) restriction

$$p(i) > 0, \quad i = 1, \ldots, N, \tag{3.2}$$

every document is referenced infinitely often. A pmf $\mathbf{p}$ on $\{1, \ldots, N\}$ which satisfies (3.2) is said to be an *admissible* pmf.

3.1.2 The reference model

The statistics of user requests $\{R_t,\ t = 0, 1, \ldots\}$ is expressed through the *reference model* associated with the sequence $\mathbf{R}$. One model that is commonly used in the design and evaluation of replacement policies is the IRM under which the rvs $\{R_t,\ t = 0, 1, \ldots\}$ are i.i.d rvs distributed according to some pmf $\mathbf{p}$ on $\mathcal{N}$. Under the IRM, the pmf (3.1) clearly exists by the Strong Law of Large Numbers, and coincides with the given pmf $\mathbf{p}$.

The main disadvantage of the IRM lies in its inability to capture the temporal correlations observed in practical Web request streams [15, 36, 38, 39, 51]. In spite of this fact, most of the analysis presented in this work is carried out under the IRM assumption, since its simplicity permits the finding of efficient (provably optimal)

eviction policies[1], e.g., see the analysis of the optimal policy A_0 by Denning and Aho [2].

A second reference model often encountered in caching applications is the *Markov Reference Model* (MRM) according to which requests are modeled by a stationary and ergodic Markov chain [6, 41]. Under the MRM, correlations are tracked through the single-step transition probabilities

$$P_{ji} := \mathbf{P}\left[R_{t+1} = i | R_t = j\right], \quad i,\ j = 1, \ldots, N, \quad t = 0, 1, \ldots \tag{3.3}$$

and the initial pmf $\mathbf{p} = (p(1), \ldots, p(N))$ is the unique pmf which satisfies

$$p(i) = \sum_{j=1}^{N} p(j) P_{ji}, \quad i = 1, \ldots, N. \tag{3.4}$$

The transition probabilities (3.3) are determined by the request stream $\mathbf{R}$ through

$$P_{ji} = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} \mathbf{1}\left[R_{t+1} = i,\ R_t = j\right]}{\sum_{t=0}^{T} \mathbf{1}\left[R_t = j\right]} \ \ a.s. \quad i,\ j = 1, \ldots, N. \tag{3.5}$$

Once available, the stationary pmf $\mathbf{p}$ can then be obtained as the unique solution to the linear system (3.4). The MRM specializes to the IRM whenever $P_{ji} = p(i),\ i, j = 1, \ldots, N$.

Additional request models can be developed by specializing the transition probabilities of the MRM, one such model being the Partial Markov Model (PMM). Details concerning the PMM can be found in [6, 74] (and references therein).

3.1.3 Dynamics of the cache content

Throughout, let S_t denote the set of documents stored at the cache, just before the request R_t is presented. The set S_t is a subset of $\mathcal{N}$, and we assume that the cache

[1] Additional reference models and the impact of temporal correlations on the performance of practical caching algorithms can be found in the dissertation by Vanichpun [74].

can contain at most M objects with $M \leq N$.

The content of the cache evolves after the request R_t is handled according to the following operational rules: If S_t contains fewer than M objects (i.e., the cache is not full) and a copy of R_t is not available in the cache, then R_t is sent from the server and is placed in the cache. On the other hand, when the cache is full and R_t is retrieved from the server, then the cache must decide whether R_t should be placed in the cache, and if so, which single document U_t to purge from the cache in order to make room for the new document. In all other cases, i.e., when R_t is already contained in S_t, the set of documents in the cache remains unchanged in response to the user request. To summarize, we have

$$S_{t+1} \;=\; \tau(S_t, R_t, U_t) \;=\; \begin{cases} S_t & \text{if } R_t \in S_t \\ S_t + R_t & \text{if } R_t \notin S_t,\ |S_t| < M \\ S_t + R_t - U_t & \text{if } R_t \notin S_t,\ |S_t| = M \end{cases} , \qquad (3.6)$$

where $|S_t|$ denotes the cardinality of the set S_t, and $S_t + R_t - U_t$ is a subset of $\mathcal{N}$ obtained from S_t by adding R_t and removing U_t, in that order. Caches that operate under such rules are often referred to as *demand-driven* caches.

The eviction action U_t at time $t = 0, 1, \ldots$ is dictated by a cache replacement policy. *Mandatory eviction policies* require that R_t be placed in the cache upon every cache-miss, so that U_t in (3.6) is a single document contained in S_t. Alternatively, *optional eviction policies* permit U_t to be either an object in S_t or the request R_t. When U_t is selected to be the new download R_t, then R_t is not placed in the cache, and no document is evicted.

Under the operational assumptions (3.6) and the admissibility condition (3.2), the cache becomes full once M distinct documents have been referenced, and remains so from that time onward. As we have in mind to develop good eviction policies, and

since no objects are evicted until the cache fills up, there is no loss of generality in assuming that the cache is initially full, as we do from now on. In other words, we assume $|S_t| = M$ for all $t = 0, 1, \ldots$, in which case (3.6) simplifies to

$$S_{t+1} = \begin{cases} S_t & \text{if } R_t \in S_t \\ S_t + R_t - U_t & \text{if } R_t \notin S_t \end{cases} . \tag{3.7}$$

3.2 Cache states and replacement policies

We define the variable X_t as the *state of the cache* at time $t = 0, 1, \ldots$. The evolution of the cache is tracked by the collection $\{X_t,\ t = 0, 1, \ldots\}$ and is affected by the stream of incoming requests $\mathbf{R}$, and by the policy π that produces the eviction actions $\{U_t,\ t = 0, 1, \ldots\}$.

For a number of reasons that will be discussed shortly, we find it useful to select X_t as the pair $X_t = (S_t, R_t)$ for all $t = 0, 1, \ldots$: The set of cached documents S_t can be easily recovered from the state variable X_t, and the next cache state X_{t+1} is fully determined by X_t once U_t and R_{t+1} are provided, under the transition rule (3.7). Furthermore, the eviction decision U_t adopted by the replacement policy π in response to R_t is entirely based on the observed system history up to time t. To formalize this, let $\mathcal{U} = \{0, 1, \ldots, N\}$ denote the action space, so that U_t is in $\mathcal{U}$ for all $t = 0, 1, \ldots$. We denote by $\mathcal{S}_M$ the set of all possible collections of documents stored at the cache, i.e., all subsets of $\{1, \ldots, N\}$ with size M. The cache state space $\mathcal{X}$ is defined as $\mathcal{X} = \mathcal{S}_M \times \mathcal{N}$, so that X_t is in $\mathcal{X}$ for all $t = 0, 1, \ldots$.

The eviction decision U_t taken by the policy π at time $t = 0, 1, \ldots,$ is described

by the mapping

$$\pi_t : (\mathcal{X} \times \mathcal{U})^t \times \mathcal{X} \to \mathcal{U}, \tag{3.8}$$

with the convention that $U_t = 0$ whenever R_t is in S_t. When R_t is not in S_t, then $U_t \in S_t + R_t$ when optional replacement policies are considered, and $U_t \in S_t$ under mandatory eviction. In either case, the collection $\{\pi_t,\ t = 0, 1, \ldots\}$ defines the replacement policy π.

We shall find it useful to consider *randomized* replacement policies which are now defined: A randomized replacement policy π is a collection $\{\pi_t,\ t = 0, 1, \ldots\}$ of mappings

$$\pi_t : (\mathcal{X} \times \mathcal{U})^t \times \mathcal{X} \times \mathcal{U} \to [0, 1] \tag{3.9}$$

such that for all $t = 0, 1, \ldots$, we have

$$\sum_{u=0}^{N} \pi_t(X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t; u) = 1 \tag{3.10}$$

with

$$\pi_t(X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t; u) = \delta(u, 0), \quad R_t \in S_t \tag{3.11}$$

and

$$\pi_t(X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t; u) = 0, \quad R_t \notin S_t,\ u \notin S_t + R_t \tag{3.12}$$

for all $u = 0, 1, \ldots, N$. The class of all (possibly randomized) replacement policies is denoted by $\mathcal{P}$.

If a non-randomized policy π has the property

$$U_t = \pi_t(S_t, R_t), \quad t = 0, 1, \ldots \tag{3.13}$$

then we say that π is a Markov policy. If in addition π_t does not depend on t for all $t = 0, 1, \ldots$, then the policy is called a Markov stationary policy. Similar definitions can be given for randomized Markov stationary policies [32].

3.3 The probability measure

In demand-driven caching, the replacement policy is the single means by which engineers can shape the content of the cache. Efficient policies are those that manipulate the stored content to minimize a cost associated with the operation of the system (over the long run). Throughout the remainder of this chapter, we discuss several costs that are widely used in practical caching systems (e.g., the hit rate, byte hit rate, and others). Before defining them, we must first define the probability measure associated with a replacement policy π.

For each policy π in $\mathcal{P}$, we define the probability measure $\mathbf{P}_\pi$ on the product space $(\mathcal{X} \times \{0, 1, \ldots, N\})^\infty$ (equipped with its natural Borel σ-field) through the following requirements: For a randomized policy π, we have

$$\mathbf{P}_\pi \left[U_t = u | X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t\right] = \pi_t(X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t; u)$$

for each $t = 0, 1, \ldots$ and all $u = 0, 1, \ldots, N$. If π is a non-randomized policy, then this last requirement takes the form

$$\mathbf{P}_\pi \left[U_t = u | X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t\right] = \delta(u, \pi_t(X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t)).$$

In either case, for every state (S, r) in $\mathcal{X}$, it is also required that

$$\begin{aligned}
&\mathbf{P}_\pi \left[S_{t+1} = S, R_{t+1} = r | X_0, U_0, \ldots, X_{t-1}, U_{t-1}, X_t, U_t\right] \\
&= \mathbf{P}\left[R_{t+1} = r | R_0, \ldots, R_t\right] \mathbf{P}_\pi \left[S_{t+1} = S | X_t, U_t\right] \\
&= \mathbf{P}\left[R_{t+1} = r | R_0, \ldots, R_t\right] \mathbf{1}\left[\tau(S_t, R_t, U_t) = S\right]. \qquad (3.14)
\end{aligned}$$

Throughout, we denote by $\mathbf{E}_\pi$ the expectation operator associated with the probability measure $\mathbf{P}_\pi$.

3.4 Optimal replacement policies

A one-step generic cost function $c : \{1, \ldots, N\} \to \mathbb{R}_+$ represents the penalty incurred by the system in the event that the requested document is not found in the cache. This cost c can be specialized to reflect various costs proposed in the literature [4, 18, 36, 37, 71], as later illustrated through some examples.

Fix $T = 0, 1, \ldots$. For a given initial system state (S, r), the total expected cost over the horizon $[0, T]$ under the policy π in $\mathcal{P}$ is given by

$$J_c(\pi; T; (S, r)) = \mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \notin S_t\right] c(R_t) \mid S_0 = S,\ R_0 = r \right]. \tag{3.15}$$

With any initial state pmf $\mathbf{p}_0 = (p_0(S, r),\ (S, r) \in \mathcal{X})$ where $p_0(S, r) = \mathbf{P}\left[X_0 = (S, r)\right]$, the cumulative expected cost over the horizon $[0, T]$ becomes

$$J_c(\pi; T) := \mathbf{E}\left[J_c(\pi; T; X_0)\right] = \sum_{(S,r)\in\mathcal{X}} p_0(S, r) J_c(\pi; T; (S, r)) \tag{3.16}$$

upon averaging over all possible starting positions according to p_0.

Of primary interest is the expected average cost (over the infinite horizon) under the policy π defined by

$$\begin{aligned} J_c(\pi) &:= \lim_{T\to\infty} \sup \frac{1}{T+1} J_c(\pi; T) \\ &= \lim_{T\to\infty} \sup \frac{1}{T+1} \mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \notin S_t\right] c(R_t) \right]. \end{aligned} \tag{3.17}$$

The problem we wish to address can now be formulated as one of finding a cache replacement policy $\pi^\star$ in $\mathcal{P}$ that satisfies

$$J_c(\pi^\star) \leq J_c(\pi), \quad \pi \in \mathcal{P}. \tag{3.18}$$

Any policy for which this condition holds is referred to as an optimal policy under the expected average cost criterion. It is well known [32, 68] that under finite state and action spaces (as is the case in the problem at hand), there exists a Markov stationary policy that satisfies (3.18). In order to find one such optimal Markov stationary policy, we follow the procedure below.

First, we find it useful to extend the notion of optimality to the finite horizon. More specifically, for each $T = 0, 1, \ldots$, the collection $\{\pi_t,\ t = 0, 1, \ldots, T\}$ defines a replacement policy that dictates the eviction actions π_t (previously defined in Section 3.2) at time $t = 0, 1, \ldots, T$. The class of all such replacement policies is denoted by $\mathcal{P}_T$. A policy $\pi^\star$ in $\mathcal{P}_T$ is said to be an optimal replacement policy over the horizon $[0, T]$ if

$$J_c(\pi^\star; T) \leq J_c(\pi; T), \quad \pi \in \mathcal{P}_T. \tag{3.19}$$

Since the state and action spaces under our system model are finite, the Markov chain $\{(S_t, R_t),\ t = 0, 1, \ldots\}$ has a single communicating class under the measure (3.16). It is therefore well known that for every $T = 0, 1, \ldots$ there exists a Markov policy $\pi^\star$ that satisfies (3.19), independently of the initial state pmf $\mathbf{p}_0$ on $\mathcal{X}$ [32] [page 128].

In view of these facts, we focus our attention on finding a Markov policy $\pi^\star$ in $\mathcal{P}_T$ for which

$$J_c(\pi^\star; T; (S, r)) \leq J_c(\pi; T; (S, r)), \quad \pi \in \mathcal{P}_T \tag{3.20}$$

for every initial state (S, r) in $\mathcal{X}$. If such a Markov policy is obtained for a given value of T, then the policy $\pi^\star$ also minimizes the cost function (3.16). In addition, if the policy $\pi^\star$ is a Markov stationary policy and does not depend on T, then we can construct an optimal replacement policy that minimizes the expected average cost (3.17). This procedure is carried out in the next chapter.

3.5 The cost functionals

A number of situations can be handled by adequately specializing the cost-per-step c: If $c(i) = 1,\ i = 1, \ldots, N$, then $J_c(\pi; T)$ and $J_c(\pi)$ are the expected number of cache misses over the horizon $[0, T]$ and the average miss rate under policy π, respectively. Explicitly, the miss rate is given by the expression

$$M(\pi) = \lim_{T\to\infty} \sup \frac{1}{T+1} \mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \notin S_t\right] \right]. \tag{3.21}$$

On the other hand, if c is taken to be the byte size $s(i),\ i = 1, \ldots, N$, then the byte hit rate under policy π can be defined by

$$BH(\pi) = \lim_{T\to\infty} \inf \frac{\mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \in S_t\right] s(R_t) \right]}{\mathbf{E}_\pi \left[\sum_{t=0}^{T} s(R_t) \right]}, \tag{3.22}$$

where the liminf operation reflects the fact that this performance metric is maximized. To make use of earlier notation, first note that $\mathbf{E}_\pi \left[\sum_{t=0}^{T} s(R_t) \right]$ does *not* depend on the policy π. Furthermore, under the IRM, it holds that

$$\begin{aligned} \lim_{T\to\infty} \frac{1}{T+1} \mathbf{E} \left[\sum_{t=0}^{T} s(R_t) \right] &= \lim_{T\to\infty} \frac{1}{T+1} \sum_{j=1}^{N} s(j) \sum_{t=0}^{T} \mathbf{P}\left[R_t = j\right] \\ &= \sum_{j=1}^{N} s(j)p(j), \end{aligned} \tag{3.23}$$

in which case we get

$$\begin{aligned} BH(\pi) &= 1 - \lim_{T\to\infty} \sup \frac{\mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \notin S_t\right] s(R_t) \right]}{\mathbf{E}_\pi \left[\sum_{t=0}^{T} s(R_t) \right]} \\ &= 1 - \frac{J_s(\pi)}{\sum_{j=1}^{N} s(j)p(j)}, \end{aligned} \tag{3.24}$$

and maximizing the byte hit rate is equivalent to minimizing the average cost associated with s.

Another performance metric of great interest is the user-perceived download latency. The delay experienced in the service of a single data item consists of the propagation delay from the cache to the user, and of the server-cache transmission time, in the event of a cache-miss. Assuming that the bandwidth B_{cu} between the cache and the user is fixed, and that B_{sc} is the available (fixed) bandwidth on the server-cache communication link, the average download latency can be written as[2]

$$\begin{aligned} L(\pi) &= \lim_{T\to\infty} \sup \frac{1}{T+1} \mathbf{E}_\pi \left[\sum_{t=0}^{T} \left\{ \frac{s(R_t)}{B_{cu}} + \mathbf{1}\left[R_t \notin S_t\right] \frac{s(R_t)}{B_{sc}} \right\} \right] \\ &= \frac{1}{B_{cu}} \sum_{j=1}^{N} s(j)p(j) + \frac{J_s(\pi)}{B_{sc}}. \end{aligned} \tag{3.25}$$

Thus, the average document retrieval latency is minimized by minimizing the cost $J_s(\pi)$.

Additional costs of the form $J_c(\pi)$ can be obtained for specific applications, by an appropriate selection of the cost function c. One such example can be found in wireless and mobile ad hoc networks (MANETs), where the reduction of energy consumed by the various wireless nodes is of supreme importance [59, 61]. This goal can be achieved by associating with c the energy expended in the retrieval of documents.

[2]The uplink delay is assumed negligible as it entails the transmit delay of very short control messages.

Chapter 4

A Class of Optimal Replacement Policies

4.1 The Dynamic Program

In this chapter, we introduce and execute the technical procedure that yields the optimal Markov stationary replacement policy, under the assumption that cache eviction is *not* mandatory. Throughout the discussion below, we assume that user requests are i.i.d (hence modeled according to the IRM). To aid our analysis, let R denote any $\{1, \ldots, N\}$-valued rv distributed according to the stationary pmf $\mathbf{p}$ of the IRM, so that the probability of reference in (3.14) becomes

$$\mathbf{P}\left[R_{t+1} = r | R_0, \ldots, R_t\right] = \mathbf{P}\left[R = r\right] = p(r), \quad r = 1, \ldots, N. \tag{4.1}$$

For each $T = 0, 1, \ldots$ and any given initial system state $X_0 = (S, r)$ in $\mathcal{X}$, let $V_T(S, r)$ denote the value function that captures the optimal cost over the finite

horizon $[0, T]$, namely

$$V_T(S, r) = \inf_{\pi \in \mathcal{P}_T} J_c(\pi; T; (S, r)). \tag{4.2}$$

The value function satisfies the Dynamic Programming Equation (DPE): For each $T = 0, 1, \ldots$ it holds

$$\begin{aligned} V_{T+1}(S, r) &= \mathbf{1}[r \notin S] \left\{ c(r) + \inf_{u \in S+r} \mathbf{E}[V_T(S + r - u, R)] \right\} \\ &\quad + \mathbf{1}[r \in S] \, \mathbf{E}[V_T(S, r)], \end{aligned} \tag{4.3}$$

with

$$V_0(S, r) = \mathbf{1}[r \notin S] \, c(r)$$

for every state (S, r) in $\mathcal{X}$.

Under finite state and action spaces, as is the case here, it is well known [68] that there exists a Markov policy $\pi^\star$ in $\mathcal{P}_T$ that minimizes[1] $J_c(\pi; T; (S, r))$. Consequently, the Markov policy $\pi^\star$ also minimizes the finite horizon cost function $J_c(\pi; T)$. The policy $\pi^\star$ can now be obtained by invoking the Principle of Optimality [68]: In order to minimize $J_c(\pi; T)$, the optimal actions to be taken at time $t = 0, 1, \ldots, T$ in each state (S, r) in $\mathcal{X}$ are given by

$$\pi_t^\star(T; (S, r)) := \begin{cases} \arg \min_{u \in S+r} \mathbf{E}[V_{T-t}(S + r - u, R)] & \text{if } r \notin S \\ 0 & \text{if } r \in S \end{cases}, \tag{4.4}$$

with a lexicographic tie-breaker for sake of concreteness. In this last equation, the possibility of non-eviction is reflected in the choice $u = r$ (obviously in $S + r$). A complete characterization of $\pi_t^\star(T) : \mathcal{X} \to \{0, 1, \ldots, N\}$ is provided in the following section.

[1]This fact permits us to replace *inf* with *min* in the definition of the value function (4.2) (and thus in equation (4.3) as well), since the minimum cost is attained by the Markov policy $\pi^\star$.

4.2 The optimal policy $C_0^\star$

We are now ready to state the main result of this chapter, namely the optimality of $C_0^\star$.

Theorem 4.1 *Fix $T = 0, 1, \ldots$. When cache eviction is optional and requests are modeled by the IRM, we have the identification*

$$\pi_t^\star(T; (S, r)) = \pi_0^\star(S, r), \quad t = 0, 1, \ldots, T, \tag{4.5}$$

for any state (S, r) in $\mathcal{X}$ whenever r is not in S, with

$$\pi_0^\star(S, r) := \arg \min_{u \in S+r} (p(u)c(u)). \tag{4.6}$$

A proof of Theorem 4.1 is given in Appendix A. Note that $\pi_t^\star(T; (S, r))$ does not depend on t or on the horizon T, therefore the policy $\pi^\star = (\pi_0^\star,\ \pi_0^\star,\ \ldots,\ \pi_0^\star)$ in $\mathcal{P}_T$ minimizes the cost (3.16) for all $T = 0, 1, \ldots$, and any initial pmf $\mathbf{p}_0$ on $\mathcal{X}$. Thus, the collection of actions $(\pi_0^\star,\ \pi_0^\star,\ \ldots)$ defines the non-randomized Markov stationary policy $C_0^\star$ in $\mathcal{P}$, which is optimal under the expected average cost criterion (3.18). Upon each cache miss, the policy $C_0^\star$ prescribes

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(p(j)c(j) : \ doc(j) \text{ in cache or request} \right), \end{aligned} \tag{4.7}$$

again with a lexicographic tie-breaker.

Different optimal policies can be derived from $C_0^\star$ by specializing the cost c per document. For example, consider the policy $A_0^\star$ obtained for $c(i) = 1,\ i = 1, \ldots, N$,

which dictates

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(p(j) : \ doc(j) \text{ in cache or request } \right). \end{aligned} \tag{4.8}$$

The policy $A_0^\star$ minimizes the miss rate incurred by the caching system.

Similarly, the byte hit rate and service speed are maximized by associating the cost c with the byte size function s. The resulting policy $C_0^\star$ prescribes

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(p(j)s(j) : \ doc(j) \text{ in cache or request } \right), \end{aligned} \tag{4.9}$$

and we refer to it as the policy $S_0^\star$.

4.3 The optimal cost

In order to calculate the long-run average cost incurred by the use of the policy $C_0^\star$, we first define the permutation σ_c of $\{1, \ldots, N\}$, which orders the values $p(i)c(i), i = 1, \ldots, N$, in decreasing order, namely

$$p(\sigma_c(1))c(\sigma_c(1)) \geq p(\sigma_c(2))c(\sigma_c(2)) \geq \ldots \geq p(\sigma_c(N))c(\sigma_c(N)), \tag{4.10}$$

again with a lexicographic tie-breaker. The key observation here is that since $p(i) > 0, \ i = 1, \ldots, N$, then every document is eventually referenced and therefore the long-term usage of the policy $C_0^\star$ results in M fixed documents in the cache, namely the documents $\sigma_c(1), \ldots, \sigma_c(M)$. Formally, if we denote by

$$S_\infty^c := \{\sigma_c(1), \ldots, \sigma_c(M)\} \tag{4.11}$$

the steady state content of the cache under the cost c, then

$$\lim_{t\to\infty} \mathbf{P}_{C_0^\star}\left[\sigma_c(i) \in S_t\right] = \begin{cases} 1, & i = 1, \ldots, M \\ 0, & i = M+1, \ldots, N \end{cases} . \tag{4.12}$$

This fact allows us to calculate the long-term average cost (3.17) incurred by $C_0^\star$, as reported in the following lemma.

Lemma 4.1 *The long-term average cost incurred by the operation of the optimal policy $C_0^\star$ is expressed by*

$$J_c(C_0^\star) = \sum_{i \notin S_\infty^c} p(i)c(i) = \sum_{i=M+1}^{N} p(\sigma_c(i))c(\sigma_c(i)), \tag{4.13}$$

provided user requests are modeled by the IRM with pmf $\mathbf{p}$.

Proof. The proof is immediate and utilizes the fact that under the condition $p(i) > 0,\ i = 1, \ldots, N$, there exists a finite time index, say $t^\star$, for which $S_{t^\star} = S_\infty^c$. Since $\sup_{i=1,\ldots,N} |c(i)| < \infty$, it is plain that

$$\lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{t^\star-1} \mathbf{1}\left[R_t \notin S_t\right] c(R_t) = 0. \tag{4.14}$$

Under $C_0^\star$ we have $S_t = S_{t^\star} = S_\infty^c$ for all $t \geq t^\star$, so that

$$\begin{aligned} \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=t^\star}^{T} \mathbf{1}\left[R_t \notin S_t\right] c(R_t) &= \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=t^\star}^{T} \mathbf{1}\left[R_t \notin S_\infty^c\right] c(R_t) \\ &= \mathbf{E}\left[\mathbf{1}\left[R \notin S_\infty^c\right] c(R)\right] \quad a.s., \end{aligned} \tag{4.15}$$

by the Strong Law of Large Numbers, and the rvs R and S_∞^c are independent. The claim (4.13) follows from (4.14), (4.15), and the Bounded Convergence Theorem. ■

The miss rate (3.21) obtained by the policy $A_0^\star$ at (4.8) can now be calculated with the help of Lemma 4.1: When $c(i) = 1,\ i = 1, \ldots, N$, the permutation σ_1 at (4.10) orders the set of available documents according to their popularity. In other words, $\sigma_1(i),\ i = 1, \ldots, N$, is identified with the i^{th} most popular document. The optimal resulting cost is given by (4.13) as

$$M(A_0^\star) = \sum_{i=M+1}^{N} p(\sigma_1(i)) = 1 - \mathbf{P}\left[R \in \{\sigma_1(1), \ldots, \sigma_1(M)\}\right].$$

The optimal byte hit rate and the maximum average service speed can both be calculated in a similar manner. If c is associated with the byte size function s, then

$$S_\infty^s = \{\sigma_s(1), \ldots, \sigma_s(M)\},$$

where σ_s is any permutation ensuring the ordering

$$p(\sigma_s(1))s(\sigma_s(1)) \geq \ldots \geq p(\sigma_s(N))s(\sigma_s(N)). \tag{4.16}$$

The byte hit rate (3.24) incurred by the long-term usage of $S_0^\star$ at (4.9) is given by

$$BH(S_0^\star) \;=\; 1 \;-\; \frac{\sum_{i=M+1}^{N} p(\sigma_s(i))s(\sigma_s(i))}{\sum_{j=1}^{N} p(j)s(j)} \;=\; \frac{\sum_{i=1}^{M} p(\sigma_s(i))s(\sigma_s(i))}{\sum_{j=1}^{N} p(j)s(j)},$$

and the optimal average service speed (3.25) can be expressed as

$$L(S_0^\star) \;=\; \frac{\sum_{j=1}^{N} p(j)s(j)}{B_{cu}} \;+\; \frac{\sum_{i=M+1}^{N} p(\sigma_s(i))s(\sigma_s(i))}{B_{sc}}. \tag{4.17}$$

4.4 Implementing $C_0^\star$

In practice, the popularity vector $\mathbf{p} = (p(1), \ldots, p(N))$ is not available and needs to be estimated on-line from incoming requests. By invoking the Certainty Equivalence Principle, the probabilities $p(i),\ i = 1, \ldots, N$, can be estimated by measuring the request frequency up to the k^{th} request, $k = 1, 2, \ldots$, through

$$\hat{p}_k(i) = \frac{1}{k} \sum_{t=0}^{k-1} \mathbf{1}\left[R_t = i\right], \quad i = 1, \ldots, N, \tag{4.18}$$

and the policy $C_0^\star$ is implemented by enforcing the rule

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min\left(\hat{p}(j)c(j) : \ doc(j) \text{ in cache or request } \right). \end{aligned} \tag{4.19}$$

Surveys of replacement policies [9, 65] reveal that numerous eviction algorithms of the form (4.19) have already been developed, and were proved to be efficient in practical systems. The well established Greedy-Dual* (GD*) used by the Squid cache [25], the Popularity Aware Greedy-Dual-Size (GDSP) suggested by Jin and Bestavros [36, 37], Cao and Irani's Greedy-Dual-Size (GDS) [18], and the Least Frequently Used - Document Aging (or LFU-AD in short) proposed in [4], are examples of such algorithms.[2] Since the GDSP and LFU-AD can be derived from the GD* by specializing its associated parameters, the GD* is now described in detail.

Let $c_{GD} : \{1, \ldots, N\} \to \mathbb{R}_+$ denote an arbitrary cost used by the GD* policy. Under optional eviction, when a request R_t presented at time $t = 0, 1, \ldots$ incurs a cache miss, the GD* policy prescribes

$$\text{Evict } doc(i)$$

[2] Additional algorithms can be found in the work by Starobinski [71].

$$\text{if} \quad i = \arg\min \left(W + \left(\frac{\hat{p}(j) c_{GD}(j)}{s(j)} \right)^{\frac{1}{\beta}} : \; j \in S_t + R_t \right)$$

where W is the contribution of the temporal locality of reference and $\beta > 0$ is a weight factor that modulates the contribution of the probability of reference, document size and its retrieval cost, to the eviction decision. Under the IRM we can take $W = 0$, in which case the GD* policy reduces to

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(\frac{\hat{p}(j) c_{GD}(j)}{s(j)} : \; j \in S_t + R_t \right). \end{aligned}$$

This simplified policy obviously follows (4.19), with cost function c given by

$$c(i) = \frac{c_{GD(i)}}{s(i)}, \quad i = 1, \ldots, N. \tag{4.20}$$

Here, the size function s is in the denominator, in contrast to the cost used by the policy $S_0^\star$, to ensure that large documents do not remain in the cache, and thus make room for more (smaller) data items.

In addition to the estimation of the pmf $\mathbf{p}$, it is sometimes required to estimate the cost values $c(i),\ i = 1, \ldots, N$, e.g., in the case of document latency where the document size might be fully known in advance, but the available bandwidth to the server needs to be measured on-line at request time. If $\hat{c}_k(i),\ i = 1, \ldots, N$, denote estimates of the document costs at the time instance of the k^{th} request, then the policy $C_0^\star$ is implemented through the eviction law

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(\hat{p}_k(j) \hat{c}_k(j) : \; doc(j) \text{ in cache or request } \right). \end{aligned}$$

4.5 The non-optimality of C_0

Some caching systems require that a document must be removed from the cache upon each instance of a cache miss. At this time, the structure of the optimal replacement policy for such applications is not known under an arbitrary retrieval cost c. A natural question is whether the policy C_0 given by

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \{\, p(j)c(j) : \ doc(j) \text{ in cache } \,\} \end{aligned} \tag{4.21}$$

is optimal, and if it is *not* optimal, then what is the penalty incurred by the use of the policy C_0 instead of the optimal policy $C_0^\star$.

To answer these queries, let $\mathcal{P}_{Mand}$ denote the class of all (possibly randomized) replacement policies in $\mathcal{P}$ that enforce mandatory eviction. Clearly, the set of policies $\mathcal{P}_{Mand}$ being a subset of $\mathcal{P}$, it is plain that

$$\inf_{\pi\in\mathcal{P}} J_c(\pi) \leq \inf_{\pi\in\mathcal{P}_{Mand}} J_c(\pi). \tag{4.22}$$

However, under the mandatory eviction restriction, it is still possible that the optimal replacement policy coincides with the policy C_0.

In view of the structure of the policy A_0 [2], which is optimal in the uniform cost case under mandatory eviction, it is very tempting to conclude that C_0 is optimal on the set $\mathcal{P}_{Mand}$. Unfortunately, C_0 is not optimal in general, as can be shown through simple examples: Take $N = 3$, $M = 2$, a stationary pmf $\mathbf{p} = (0.009, 0.001, 0.99)$, and the associated costs $c = (20, 5, 1)$. By running the Dynamic Program over a long period of time, we find that the optimal action under the average cost criterion is obtained by removing $doc(3)$ from the cache, when a miss occurs. This action disproves the optimality of C_0, which would dictate the eviction of $doc(2)$.

4.6 The impact of using C_0 instead of $C_0^\star$

The main disadvantage of the policy $C_0^\star$ lies in the fact that once the documents in the set $\{\sigma_c(1), \ldots, \sigma_c(M)\}$ have been requested, they are never removed from the cache. In order to increase the content dynamics at the cache, it is preferable that C_0 be employed, in which case the $M-1$ objects $\{\sigma_c(1), \ldots, \sigma_c(M-1)\}$ are never evicted, and the M^{th} cached document (for sufficiently large value of t) is always the object referenced by the last request that prompted a cache miss.

In order to understand the tradeoffs associated with the selection of C_0 over $C_0^\star$, we calculate the difference between their resulting average costs. To do so, we must first obtain the average cost incurred by the use of the policy C_0.

Theorem 4.2 *Under the IRM with pmf* **p***, the long-term average cost associated with the policy C_0 induced by c via (4.21) is given by*

$$J_c(C_0) \;=\; \sum_{i=\sigma_c(M)}^{\sigma_c(N)} p(i)c(i) - \frac{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p^2(i)c(i)}{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p(i)}$$

under the convention (4.10).

Proof. The proof follows the steps presented in the proof of Lemma 4.1. Under the condition $p(i) > 0,\ i = 1, \ldots, N$, there exists a finite (random) time index $t^\star$ such that the documents in the set[3] $\hat{S}_\infty^c = \{\sigma_c(1), \ldots, \sigma_c(M-1)\}$ are contained in $S_{t^\star}$, and (4.14) also holds under C_0. We can therefore write the average cost under C_0 as

$$\lim_{T\to\infty} \frac{1}{T+1} \sum_{t=t^\star}^{T} \left[1 - \mathbf{1}\left[R_t \in S_t\right]\right] c(R_t) = \mathbf{E}\left[c(R) - \mathbf{1}\left[R \in S\right] c(R)\right] \;\; a.s., \text{ (4.23)}$$

[3]We use the notation $\hat{S}_\infty^c$ to denote the set of $M-1$ documents that are never evicted once requested under C_0, to distinguish it from the set S_∞^c in (4.11), which contains M documents.

where S denotes the rv of cache documents after time $t^\star$ (i.e., in steady state), immediately before document R is requested, and the rvs S and R are independent. First, it is plain that

$$\mathbf{E}\left[c(R)\right] = \sum_{i=\sigma_c(1)}^{\sigma_c(N)} p(i)c(i). \tag{4.24}$$

Next, to calculate $\mathbf{E}\left[\mathbf{1}\left[R \in S\right] c(R)\right]$, note that

$$\mathbf{E}\left[\mathbf{1}\left[R \in S\right] \mathbf{1}\left[R \in \hat{S}_\infty^c\right] c(R)\right] = \sum_{j \in \hat{S}_\infty^c} p(j)c(j), \tag{4.25}$$

and

$$\begin{aligned}
&\mathbf{E}\left[\mathbf{1}\left[R \in S\right] \mathbf{1}\left[R \notin \hat{S}_\infty^c\right] c(R)\right] \\
&= \mathbf{E}\left[\mathbf{1}\left[R_1 = R_0,\ R_0 \notin \hat{S}_\infty^c\right] c(R_1)\right] \\
&+ \mathbf{E}\left[\mathbf{1}\left[R_2 = R_0,\ R_0 \notin \hat{S}_\infty^c,\ R_1 \in \hat{S}_\infty^c\right] c(R_2)\right] \\
&+ \mathbf{E}\left[\mathbf{1}\left[R_3 = R_0,\ R_0 \notin \hat{S}_\infty^c,\ R_1 \in \hat{S}_\infty^c,\ R_2 \in \hat{S}_\infty^c\right] c(R_3)\right] \\
&+ \ldots \\
&= \sum_{t=0}^{\infty} \mathbf{E}\left[\mathbf{1}\left[R_{t+1} = R_0, R_0 \notin \hat{S}_\infty^c, R_s \in \hat{S}_\infty^c, s = 1, \ldots, t\right] c(R_{t+1})\right] \\
&= \sum_{j \notin \hat{S}_\infty^c} p^2(j)c(j) \cdot \left[1 + \sum_{j \in \hat{S}_\infty^c} p(j) + \left(\sum_{j \in \hat{S}_\infty^c} p(j)\right)^2 + \ldots\right] \\
&= \frac{\sum_{j \notin \hat{S}_\infty^c} p^2(j)c(j)}{\sum_{j \notin \hat{S}_\infty^c} p(j)}.
\end{aligned} \tag{4.26}$$

Inserting the results (4.24), (4.25) and (4.26) into (4.23), we obtain the desired average cost. ■

Corollary 4.1 *The penalty incurred by the use of the non-optimal policy C_0 instead*

of the optimal policy $C_0^\star$ is captured by the difference

$$\begin{aligned} J_c(C_0) - J_c(C_0^\star) &= p(\sigma_c(M))c(\sigma_c(M)) - \frac{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p^2(i)c(i)}{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p(i)} \\ &= \frac{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p(i)\left(p(\sigma_c(M))c(\sigma_c(M)) - p(i)c(i)\right)}{\sum_{i=\sigma_c(M)}^{\sigma_c(N)} p(i)}. \end{aligned} \tag{4.27}$$

The cost difference (4.27) depends on the stationary pmf $\mathbf{p}$, and on the retrieval costs $c(j), j = 1, \ldots, N$. Given this fact, another question that is imperative for understanding the tradeoffs of using C_0 instead of $C_0^\star$ is whether there exist a pmf $\mathbf{p}$ and document costs under which the penalty (4.27) is minimized.

Theorem 4.3 *For any given cost function $c : \{1, \ldots, N\} \to \mathbb{R}_+$, there exists a pmf* **p** *for which the cost difference (4.27) is zero, in which case the policy C_0 is optimal amongst all replacement policies in $\mathcal{P}$.*

Proof. Let σ_c denote a permutation of $c(1), \ldots, c(N)$, which sorts the document costs in descending order, i.e., $c(\sigma_c(1)) \geq c(\sigma_c(2)) \geq \ldots \geq c(\sigma_c(N))$. Pick the probabilities $p(\sigma_c(1)), \ldots, p(\sigma_c(M-1))$, so that

$$p(\sigma_c(1)) \geq p(\sigma_c(2)) \geq \ldots \geq p(\sigma_c(M-1)) \geq p(\sigma_c(M)),$$

under the restriction

$$p(\sigma_c(1)) + \ldots + p(\sigma_c(M-1)) < 1.$$

One selection that meets these requirements is $p(\sigma_c(j)) = \frac{1}{M},\ j = 1, \ldots, M-1$. The cost difference (4.27) will be zero if we select

$$p(\sigma_c(j)) = \frac{p(\sigma_c(M))c(M)}{c(j)}, \quad j = M+1, \ldots, N \tag{4.28}$$

provided there exists $p(\sigma_c(M))$ in $(0,1)$ which satisfies (4.28) under the constraint

$$\sum_{j=1}^{N} p(\sigma_c(j)) = \sum_{j=1}^{M-1} p(\sigma_c(j)) + p(\sigma_c(M)) \left(1 + \sum_{j=M+1}^{N} \frac{c(\sigma_c(M))}{c(\sigma_c(j))}\right) = 1. \quad (4.29)$$

By rewriting (4.29) we get

$$p(\sigma_c(M)) = \frac{1 - \sum_{j=1}^{M-1} p(\sigma_c(j))}{1 + \sum_{j=M+1}^{N} \frac{c(\sigma_c(M))}{c(\sigma_c(j))}},$$

a quantity which is clearly in $(0,1)$. It is also simple to verify that

$$p(\sigma_c(1))c(\sigma_c(1)) \geq p(\sigma_c(2))c(\sigma_c(2)) \geq \ldots \geq p(\sigma_c(N))c(\sigma_c(N)), \quad (4.30)$$

and the permutation σ_c therefore follows the convention (4.10). Since the calculated pmf **p**, the cost function $c : \{1, \ldots, N\} \rightarrow \mathbb{R}_+$, and the permutation σ_c meet the conditions of Lemma 4.1 and Theorem 4.2, we can utilize Corollary 4.1 to conclude that the cost difference (4.27) is indeed zero, which completes the proof. ■

4.7 On the optimal policy with Markov requests

We conclude this chapter with a brief discussion regarding optimal replacement policies under the Markov Reference Model. Several studies have focused on identifying properties of replacement policies when requests are modeled according to a stationary and ergodic Markov chain: Karlin et al. [41] showed that in the case of mandatory eviction and uniform costs (i.e., $c(i) = 1,\ i = 1, \ldots, N$), if $N = M + 1$, then there exists an optimal replacement policy under which $M-1$ documents are never evicted from the cache once they have been requested. Moreover, results reported in [41] indicate that paging algorithms developed earlier under the IRM, such as the Random

Replacement (RR) and the Least Frequently Used (LFU) policies, do no perform well under the MRM. Instead, the heuristic *Commute* algorithm was proposed, and was proved to be efficient by deriving an upper bound on the resulting miss rate.[4]

To date, the structure of the optimal replacement policy under the MRM is not yet known (to the best of the author's knowledge), even in the simplified context of uniform costs, for either mandatory or optional eviction. Markov requests are expressed in our system model through

$$\mathbf{P}\left[R_{t+1} = r | R_0, \ldots, R_t\right] = \mathbf{P}\left[R_{t+1} = r | R_t\right], \quad t = 0, 1, \ldots \tag{4.31}$$

in (3.14) where R_0 is distributed according to the unique pmf $\mathbf{p}$ that satisfies (3.4). Optimal replacement policies can therefore be found once the costs $c(i),\ i = 1, \ldots, N,$ and the transition probabilities $P_{ji},\ i, j = 1, \ldots, N,$ are available, by solving the Dynamic Program presented earlier in Section 4.1 (with all the appropriate modfications to account for the MRM).

[4] Additional observations regarding eviction policies under the MRM can be found in [6, 43].

Chapter 5

Optimal Caching Under a Constraint

5.1 Problem formulation

One possible approach to capture the multi-criteria aspect of running caching systems is to introduce constraints. Here, we revisit the caching problem with optional eviction and when user requests are modeled by the IRM, under a single constraint.

Formulation of the caching problem with a constraint requires two cost functions, say $c, d : \{1, \ldots, N\} \to \mathbb{R}_+$. As before, $c(R_t)$ and $d(R_t)$ represent different costs of retrieving the requested document R_t if not in the cache S_t at time t. For instance, we could take $c(i) = 1$ and $d(i) = s(i)$ to reflect interest in the miss rate and the document retrieval latency, respectively.

The problem of interest can now be formulated as follows: Given some $\alpha > 0$, we say that the policy $\pi \in \mathcal{P}$ satisfies the constraint at level α if

$$J_d(\pi) \leq \alpha. \tag{5.1}$$

Let $\mathcal{P}(d; \alpha)$ denote the class of all cache replacement policies in $\mathcal{P}$ that satisfy the constraint (5.1). The problem is to find a cache replacement policy $\pi^\star$ in $\mathcal{P}(d; \alpha)$

such that

$$J_c(\pi^\star) \leq J_c(\pi), \quad \pi \in \mathcal{P}(d;\alpha). \tag{5.2}$$

Any such policy $\pi^\star$ is referred to as a constrained optimal policy (at level α). With the choice $c(i) = 1$ and $d(i) = s(i)$ this formulation would focus on minimizing the miss rate with a bound on average latency of document retrieval.

One natural approach to solving this problem is to consider the corresponding Lagrangian functional defined by

$$\tilde{J}_\lambda(\pi) = J_c(\pi) + \lambda J_d(\pi), \quad \pi \in \mathcal{P},\ \lambda > 0. \tag{5.3}$$

The basic idea is then to find for each $\lambda \geq 0$, a cache replacement policy $\pi^\star(\lambda)$ in $\mathcal{P}$ such that

$$\tilde{J}_\lambda(\pi^\star(\lambda)) \leq \tilde{J}_\lambda(\pi), \quad \pi \in \mathcal{P}. \tag{5.4}$$

Now, if for some $\lambda^\star \geq 0$, the policy $\pi^\star(\lambda^\star)$ happens to *saturate* the constraint at level α, i.e., $J_d(\pi^\star(\lambda^\star)) = \alpha$, then the policy $\pi^\star(\lambda^\star)$ belongs to $\mathcal{P}(d;\alpha)$ and its optimality implies

$$\tilde{J}_{\lambda^\star}(\pi^\star(\lambda^\star)) \leq \tilde{J}_{\lambda^\star}(\pi), \quad \pi \in \mathcal{P}. \tag{5.5}$$

In particular, for any policy π in $\mathcal{P}(d;\alpha)$, this last inequality readily leads to

$$J_c(\pi^\star(\lambda^\star)) \leq J_c(\pi), \quad \pi \in \mathcal{P}(d;\alpha), \tag{5.6}$$

and the policy $\pi^\star(\lambda^\star)$ solves the constrained optimization problem.

The only glitch in this approach resides in the use of the limsup operation in the definition (3.17), so that $\tilde{J}_\lambda(\pi)$ is not necessarily the long-run average cost under the policy π for some appropriate one-step cost. Thus, finding the optimal cache replacement policy $\pi^\star(\lambda)$ specified by (5.5) cannot be achieved in a straightforward manner.

5.2 A Lagrangian approach

Following the treatment in [13], we now introduce an alternate Lagrangian formulation which circumvents this technical difficulty and allows us to carry out the program outlined above: For each $\lambda \geq 0$ we define the one-step cost function $b_\lambda : \{1, \ldots, N\} \to \mathbb{R}_+$ by

$$b_\lambda(i) = c(i) + \lambda d(i), \quad i = 1, \ldots, N \tag{5.7}$$

and consider the corresponding long-run average functional (3.17), i.e., for any policy π in $\mathcal{P}$ we set

$$J_\lambda(\pi) := J_{b_\lambda}(\pi) = \limsup_{T\to\infty} \frac{1}{T+1} \mathbf{E}_\pi \left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \notin S_t\right] b_\lambda(R_t) \right]. \tag{5.8}$$

With these definitions we get

$$J_{b_\lambda}(\pi) \leq \tilde{J}_\lambda(\pi), \quad \pi \in \mathcal{P} \tag{5.9}$$

by standard properties of the limsup, with equality

$$J_{b_\lambda}(\pi) = \tilde{J}_\lambda(\pi) \tag{5.10}$$

whenever π is a Markov stationary policy.

For each $\lambda \geq 0$, the (unconstrained) caching problem associated with the cost b_λ coincides with the system model described in Chapter 3, under which both the state and action spaces are finite. Thus, there exists a non-randomized stationary Markov policy, denoted g_λ, which is optimal [32], i.e.,

$$J_\lambda(g_\lambda) \leq J_\lambda(\pi), \quad \pi \in \mathcal{P}, \tag{5.11}$$

and earlier remarks yield

$$\tilde{J}_\lambda(g_\lambda) \leq \tilde{J}_\lambda(\pi), \quad \pi \in \mathcal{P}. \tag{5.12}$$

In other words, the stationary Markov policy g_λ also minimizes the Lagrangian functional (5.3), and the relation

$$J_\lambda(g_\lambda) = \inf_{\pi \in \mathcal{P}} J_\lambda(\pi) = \inf_{\pi \in \mathcal{P}} \tilde{J}_\lambda(\pi) \tag{5.13}$$

holds. Consequently, as argued in Section 5.1, if for some $\lambda^\star \geq 0$, the policy $g_{\lambda^\star}$ saturates the constraint at level α, then the policy $g_{\lambda^\star}$ solves the constrained optimization problem.

The difficulty is that *a priori* we may have $J_\lambda(g_\lambda) \neq \alpha$ for all $\lambda \geq 0$. However, the arguments given above still show that the search for the constrained optimal policy can be recast as the problem of finding $\gamma \geq 0$ and a (possibly randomized) stationary Markov policy $g^\star$ such that

$$J_d(g^\star) = \alpha \tag{5.14}$$

and

$$J_\gamma(g^\star) \leq J_\gamma(\pi), \quad \pi \in \mathcal{P}. \tag{5.15}$$

5.3 On the way to finding the optimal policy

The appropriate multiplier γ and the policy $g^\star$ appearing in (5.14) and (5.15) will be identified in the next section. To help us in this process we need some technical facts and notation which we now develop.

Theorem 5.1 *The optimal cost function $\lambda \to J_\lambda(g_\lambda)$ is a non-decreasing concave function which is piecewise linear on $\mathbb{R}_+$.*

Some observations are in order before giving a proof of Theorem 5.1: Fix $\lambda \geq 0$. In view of Theorem 4.1 we can select g_λ as the policy $C_0^\star$ induced by b_λ, i.e., for each

$t = 0, 1, \ldots$, the policy g_λ prescribes

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \arg\min \left(p(j)(c(j) + \lambda d(j)) : \ j \in S_t + R_t \right). \end{aligned}$$

Let σ_λ denote the permutation (4.10) of $\{1, \ldots, N\}$ that orders the values $p(i)b_\lambda(i)$, $i = 1, \ldots, N$, in increasing order, namely $p(\sigma(1))b_\lambda(\sigma(1)) \geq \ldots \geq p(\sigma(N))b_\lambda(\sigma(N))$, with a lexicographic tie-breaker. Let $S_\infty(\lambda)$ denote the steady state stack induced by the policy g_λ, namely the collection of documents in the cache that results from the long-term usage of the policy g_λ. Obviously, we have[1]

$$S_\infty(\lambda) = \{\sigma_\lambda(1), \ldots, \sigma_\lambda(M)\} \tag{5.16}$$

so that

$$J_\lambda(g_\lambda) = J_{b_\lambda}(g_\lambda) = \sum_{i \notin S_\infty(\lambda)} p(i)b_\lambda(i) \tag{5.17}$$

upon rephrasing the result of Lemma 4.1.

Given the affine nature (in the variable λ) of the cost, there must exist a finite and stricly increasing sequence of non-zero scalar values $\lambda_1, \ldots, \lambda_L$ in $\mathbb{R}_+$ with $0 < \lambda_1 < \ldots < \lambda_L$ such that for each $\ell = 0, \ldots, L$, it holds that

$$S_\infty(\lambda) = S_\infty(\lambda_\ell), \quad \lambda \in I_\ell := [\lambda_\ell, \lambda_{\ell+1}) \tag{5.18}$$

with the convention that $\lambda_0 = 0$ and $\lambda_{L+1} = \infty$, but with

$$S_\infty(\lambda_\ell) \neq S_\infty(\lambda_{\ell+1}), \quad \ell = 0, 1, \ldots, L-1. \tag{5.19}$$

In view of (5.17) it is plain that

$$J_\lambda(g_\lambda) = \sum_{i \notin S_\infty(\lambda_\ell)} p(i)b_\lambda(i) \tag{5.20}$$

[1]The steady state stack S_∞ given in (4.11) corresponds to the case $\lambda = 0$.

whenever λ belongs to the interval I_ℓ for some $\ell = 0, 1, \ldots, L$. These facts are described through an example in Figure 5.1, in which $N = 3$, $M = 2$, the cost vectors are $c = (1, 2, 3)$ and $d = (1, 0.5, 0.25)$, and the reference pmf is given by $\mathbf{p} = (1/3, 1/3, 1/3)$.

Proof of Theorem 5.1. For each policy π in $\mathcal{P}$, the quantities $J_c(\pi)$ and $J_d(\pi)$ are non-negative as the one-step cost function c and d are non-negative. Thus, the mapping $\lambda \to \tilde{J}_\lambda(\pi)$ (5.3) is non-decreasing and affine, and we conclude from (5.13) that the mapping $\lambda \to J_\lambda(g_\lambda)$ is indeed non-decreasing and concave. Its piecewise-linear character is a straightforward consequence of (5.20). ■

In order to proceed we make the following simplifying assumption:

Assumption (A) If for some $\lambda \geq 0$ it holds that $p(i)b_\lambda(i) = p(j)b_\lambda(j)$ for some distinct $i, j = 1, \ldots, N$, then there does not exist any $k \neq i, j$ with $k = 1, \ldots, N$ such that $p(i)b_\lambda(i) = p(j)b_\lambda(j) = p(k)b_\lambda(k)$.

This assumption can be removed at the cost of a more delicate analysis without affecting the essence of the optimality result to be derived shortly.

For each $\ell = 0, 1, \ldots, L$, the relative position of the quantities $p(i)b_\lambda(i), i = 1, \ldots, N$, remains unchanged as λ sweeps through the interval $(\lambda_\ell, \lambda_{\ell+1})$. Under assumption (A), when going through $\lambda = \lambda_{\ell+1}$, a *single* reversal occurs in the relative position with

$$S_\infty(\lambda_\ell) = \{\sigma_{\lambda_\ell}(1), \ldots, \sigma_{\lambda_\ell}(M-1), \sigma_{\lambda_\ell}(M)\} \tag{5.21}$$

and

$$S_\infty(\lambda_{\ell+1}) = \{\sigma_{\lambda_\ell}(1), \ldots, \sigma_{\lambda_\ell}(M-1), \sigma_{\lambda_\ell}(M+1)\}. \tag{5.22}$$

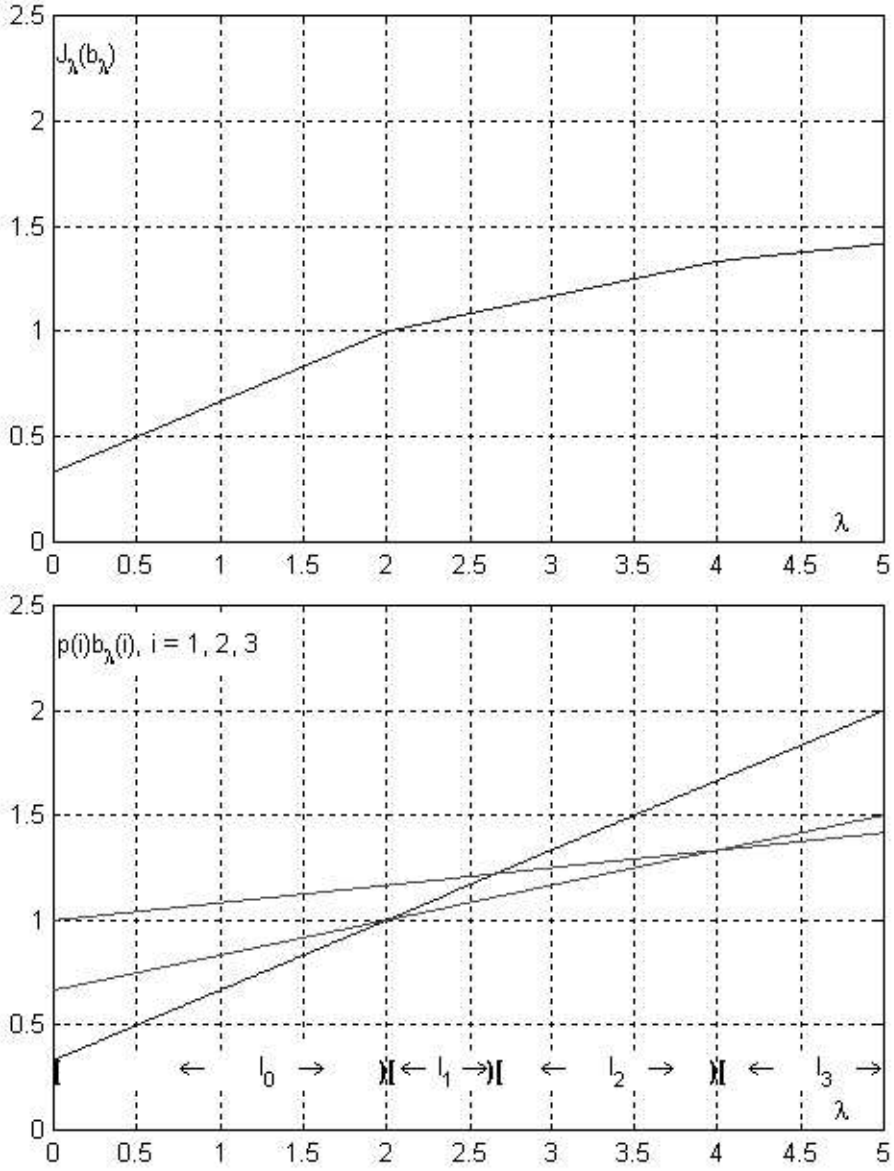

Figure 5.1: An illustration of the intervals I_l, $l = 0, 1, \ldots, L$, with $L = 3$, and the resulting optimal average cost $J_\lambda(g_\lambda)$

By continuity we must have

$$p(\sigma_{\lambda_\ell}(M))b_{\lambda_{\ell+1}}(\sigma_{\lambda_\ell}(M)) = p(\sigma_{\lambda_\ell}(M+1))b_{\lambda_{\ell+1}}(\sigma_{\lambda_\ell}(M+1)). \tag{5.23}$$

Theorem 5.2 *Under the foregoing assumptions, the mapping $\lambda \to J_d(g_\lambda)$ is a non-increasing piecewise constant function on $\mathbb{R}_+$.*

Proof. The analog of (5.20) holds in the form

$$J_d(g_\lambda) = \sum_{i \notin S_\infty(\lambda_\ell)} p(i)d(i) \tag{5.24}$$

whenever λ belongs to I_ℓ for some $\ell = 0, \ldots, L$. Hence, the mapping $\lambda \to J_d(g_\lambda)$ is piecewise constant.

Now pick $\ell = 0, 1, \ldots, L-1$ and consider λ and μ in the *open* intervals $(\lambda_\ell, \lambda_{\ell+1})$ and $(\lambda_{\ell+1}, \lambda_{\ell+2})$, respectively. The desired monotonicity will be established if we can show that $J_d(g_\mu) - J_d(g_\lambda) \leq 0$. First, from (5.24), note that

$$\begin{aligned} J_d(g_\mu) - J_d(g_\lambda) &= \sum_{i \in S_\infty(\lambda)} p(i)d(i) - \sum_{i \in S_\infty(\mu)} p(i)d(i) \\ &= p(\sigma_\lambda(M))d(\sigma_\lambda(M)) - p(\sigma_\lambda(M+1))d(\sigma_\lambda(M+1)) \end{aligned} \tag{5.25}$$

given the steady state stacks (5.21) and (5.22), as we recall that $S_\infty(\lambda) = S_\infty(\lambda_\ell)$ and $S_\infty(\mu) = S_\infty(\lambda_{\ell+1})$.

Next, pick $\epsilon > 0$ such that $\lambda + \epsilon$ and $\mu + \epsilon$ are still in the *open* intervals $(\lambda_\ell, \lambda_{\ell+1})$ and $(\lambda_{\ell+1}, \lambda_{\ell+2})$, respectively. By (5.20) we get $S_\infty(\lambda + \epsilon) = S_\infty(\lambda)$ and

$$\begin{aligned} J_{\lambda+\epsilon}(g_{\lambda+\epsilon}) - J_\lambda(g_\lambda) &= \sum_{i \notin S_\infty(\lambda+\epsilon)} p(i)b_{\lambda+\epsilon}(i) - \sum_{i \notin S_\infty(\lambda)} p(i)b_\lambda(i) \\ &= \sum_{i \notin S_\infty(\lambda)} p(i)b_{\lambda+\epsilon}(i) - \sum_{i \notin S_\infty(\lambda)} p(i)b_\lambda(i) \\ &= \epsilon \sum_{i \notin S_\infty(\lambda)} p(i)d(i). \end{aligned} \tag{5.26}$$

Similarly,

$$J_{\mu+\epsilon}(g_{\mu+\epsilon}) - J_{\mu}(g_{\mu}) = \epsilon \sum_{i \notin S_{\infty}(\mu)} p(i)d(i). \tag{5.27}$$

By Theorem 5.1, the mapping $\lambda \to J_{\lambda}(g_{\lambda})$ is concave , hence

$$J_{\mu+\epsilon}(g_{\mu+\epsilon}) - J_{\mu}(g_{\mu}) \leq J_{\lambda+\epsilon}(g_{\lambda+\epsilon}) - J_{\lambda}(g_{\lambda}).$$

Making use of (5.26) and (5.27) in the last inequality, we readily conclude that

$$\sum_{i \in S_{\infty}(\lambda)} p(i)d(i) \leq \sum_{i \in S_{\infty}(\mu)} p(i)d(i). \tag{5.28}$$

But $S_{\infty}(\lambda) = S(\lambda_{\ell})$ and $S_{\infty}(\mu) = S(\lambda_{\ell+1})$, whence (5.28) is equivalent to

$$p(\sigma_{\lambda_{\ell}}(M))d(\sigma_{\lambda_{\ell}}(M)) \leq p(\sigma_{\lambda_{\ell}}(M+1))d(\sigma_{\lambda_{\ell}}(M+1)). \tag{5.29}$$

The desired conclusion $J_d(g_{\mu}) - J_d(g_{\lambda}) \leq 0$ is now immediate from (5.25). ■

5.4 The constrained optimal replacement policy

We are now ready to discuss the form of the optimal replacement policy for the constrained caching problem. Throughout we assume Assumption (A) to hold. Several cases need to be considered:

Case 1 - The unconstrained optimal replacement policy g_0 satisfies the constraint, i.e., $J_d(g_0) \leq \alpha$, in which case $g^{\star}$ is simply the optimal replacement policy $C_0^{\star}$ for the unconstrained caching problem, associated with the generic cost c. This case is trivial and requires no proof since by Theorem 4.1 the average cost is minimized and the constraint satisfied.

Case 2 - The unconstrained optimal replacement policy does not satisfy the constraint, i.e., $J_d(g_0) > \alpha$, but there exists $\lambda > 0$ such that $J_d(g_\lambda) \leq \alpha$. Two subcases of interest emerge in this context and are presented in Theorems 5.3 and 5.4 below.

Case 2a - The situation when the policy g_λ above saturates the constraint at level α was covered earlier in the discussion; its proof is therefore omitted.

Theorem 5.3 *If there exists $\lambda > 0$ such that $J_d(g_\lambda) = \alpha$, then the policy g_λ can be taken as the optimal replacement policy $g^\star$ for the constrained caching problem (and the constraint is saturated).*

Case 2b - The case of greatest interest arises when the conditions of Theorem 5.3 are not met, i.e., $J_d(g_0) > \alpha$, $J_d(g_\mu) \neq \alpha$ for all $\mu \geq 0$ but there exists $\lambda > 0$ such that $J_d(g_\lambda) < \alpha$. In that case, by the monotonicity result of Theorem 5.2, the quantity

$$\gamma := \inf\{\lambda \geq 0 : J_d(g_\lambda) \leq \alpha\} \tag{5.30}$$

is a well defined scalar in $(0, \infty)$. In fact, we have the identification

$$\gamma = \lambda_{\lambda_{\ell+1}} \tag{5.31}$$

for some $\ell = 0, 1, \ldots, L-1$, and it holds that

$$J_d(g_{\lambda_{\ell+1}}) < \alpha < J_d(g_{\lambda_\ell}). \tag{5.32}$$

For each p in the interval $[0, 1]$, define the Markov stationary policy f_p obtained by randomizing the policy g_{λ_ℓ} and $g_{\lambda_{\ell+1}}$ with bias p. Thus, the randomized policy f_p prescribes

$$\begin{aligned} &\text{Evict } doc(i) \\ &\text{if} \quad i = \begin{cases} \arg\min\{p(j) b_{\lambda_\ell}(j) : j \in S_t + R_t\} & \text{w.p. } p \\ \arg\min\{p(j) b_{\lambda_{\ell+1}}(j) : j \in S_t + R_t\} & \text{w.p. } 1-p \end{cases} . \end{aligned} \tag{5.33}$$

Theorem 5.4 *The optimal cache replacement policy $g^\star$ for the constrained caching problem is any randomized policy f_p of the form (5.33) with p determined through the saturation equation*

$$J_d(f_p) = \alpha. \tag{5.34}$$

Proof. For the most part we follow the arguments of [13]: Let $\ell = 0, 1, \ldots, L-1$ be the integer appearing in the identification (5.31). Pick λ and μ in the *open* interval $(\lambda_\ell, \lambda_{\ell+1})$ and $(\lambda_{\ell+1}, \lambda_{\ell+2})$, respectively, in which case

$$g_\lambda = g_{\lambda_\ell} \quad \text{and} \quad g_\mu = g_{\lambda_{\ell+1}} \tag{5.35}$$

with

$$J_d(g_\mu) < \alpha < J_d(g_\lambda). \tag{5.36}$$

Thus, as in the proof of Theorem 4.4 in [13], let λ and μ go to $\lambda_{\ell+1}$ monotonically under their respective constraints. The resulting limiting policies $\underline{g}$ and $\overline{g}$ (in the notation of [13]) are simply given here by $\underline{g} = g_{\lambda_{\ell+1}}$ and $\overline{g} = g_{\lambda_\ell}$ with [2]

$$J_\gamma(f_p) = J_\gamma(g_{\lambda_{\ell+1}}) = J_\gamma(g_{\lambda_\ell}) \tag{5.37}$$

for *every* p in the interval $[0, 1]$, and the optimality

$$J_\gamma(f_p) \leq J_\gamma(\pi), \quad \pi \in \mathcal{P} \tag{5.38}$$

follows. Moreover, the mapping $p \to J_d(f_p)$ being continuous [52], with $J_d(f_p)_{p=0} = J_d(g_{\lambda_{\ell+1}})$ and $J_d(f_p)_{p=1} = J_d(g_{\lambda_\ell})$, there exists at least one value p in $(0, 1)$ such that

[2] See details in the proof of Theorem 4.4 in [13].

(5.34) holds. The proof of optimality is now complete in view of comments made in Section 5.3. ■

It is possible to give a somewhat explicit expression for $J_d(f_p)$ for p in $[0, 1]$.

Lemma 5.1 *Let $S^\star_\infty$ denote the set of documents that are never removed from the cache once requested, namely*

$$S^\star_\infty := S_\infty(\lambda_\ell) \cap S_\infty(\lambda_{\ell+1}) = \{\sigma_{\lambda_\ell}(1), \ldots, \sigma_{\lambda_\ell}(M-1)\}. \tag{5.39}$$

The average cost $J_d(f_p)$ can then be expressed as

$$J_d(f_p) = \mathbf{E}\left[d(R)\right] - \lim_{T\to\infty} \frac{1}{T+1}\mathbf{E}_{f_p}\left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \in S_t\right] d(R_t)\right] \tag{5.40}$$

with

$$\begin{aligned}\lim_{T\to\infty} \frac{1}{T+1}\mathbf{E}_{f_p}&\left[\sum_{t=0}^{T} \mathbf{1}\left[R_t \in S_t\right] d(R_t)\right] \\ &= \sum_{i\in S^\star_\infty} p(i)d(i) + r(p)p(\sigma_{\lambda_\ell}(M))d(\sigma_{\lambda_\ell}(M)) \\ &\quad +(1-r(p))p(\sigma_{\lambda_\ell}(M+1))d(\sigma_{\lambda_\ell}(M+1)),\end{aligned} \tag{5.41}$$

where $r(p)$ represents the asymptotic fraction of time that the cache contains the document $\sigma_{\lambda_\ell}(M)$, and is given by

$$r(p) = \frac{p \cdot p(\sigma_{\lambda_\ell}(M))}{p \cdot p(\sigma_{\lambda_\ell}(M)) + (1-p) \cdot p(\sigma_{\lambda_\ell}(M+1))}. \tag{5.42}$$

Proof. Under the policy f_p and i.i.d requests, if the document $\sigma_{\lambda_\ell}(M)$ is in the cache, it can only be removed from the cache when $\sigma_{\lambda_\ell}(M+1)$ is requested. The time

until the eviction of $\sigma_{\lambda_\ell}(M)$ is therefore a Geometric rv (of type II) with parameter $(1-p)\cdot p(\sigma_{\lambda_\ell}(M+1))$. Similarly, if $\sigma_{\lambda_\ell}(M+1)$ is in the cache, the time until its removal is a Geometric rv with parameter $p\cdot p(\sigma_{\lambda_\ell}(M))$. The fraction of time that $\sigma_{\lambda_\ell}(M)$ is in the cache, is then simply

$$\begin{aligned} r(p) &= \frac{\frac{1}{p(\sigma_{\lambda_\ell}(M+1))\cdot(1-p)}}{\frac{1}{p(\sigma_{\lambda_\ell}(M+1))\cdot(1-p)}+\frac{1}{p(\sigma_{\lambda_\ell}(M))\cdot(p)}} \\ &= \frac{p\cdot p(\sigma_{\lambda_\ell}(M))}{p\cdot p(\sigma_{\lambda_\ell}(M))+(1-p)\cdot p(\sigma_{\lambda_\ell}(M+1))}, \end{aligned}$$

and (5.41) follows. ■

Case 3 - Finally, assume that $J_d(g_\lambda) > \alpha$ for all $\lambda \geq 0$. This situation is of limited interest as we now argue: Fix $\lambda > 0$. For each policy π in $\mathcal{P}$, we can use the optimality of g_λ to write

$$\alpha < \lambda^{-1} J_c(g_\lambda) + J_d(g_\lambda) \leq \lambda^{-1} J_c(\pi) + J_d(\pi). \tag{5.43}$$

Thus, letting λ go to infinity, we conclude that

$$\alpha \leq J_d(\pi), \quad \pi \in \mathcal{P}. \tag{5.44}$$

The constrained caching problem has no feasible solution unless there exists a policy that saturates the constraint. Typically the inequality above will be strict.

Part II

Modeling and Measuring Internet Cache Consistency and Quality of Data

Chapter 6

Internet Cache Consistency

6.1 The cache consistency problem

The discussion in Part I of the dissertation implicitly assumes that server documents are *never* modified. In other words, the content of documents at the server remains fixed over time. This operational assumption does *not* hold in practical Web caching systems.

A cached object is termed *consistent* if it is identical to the master document, at the time it is served to the user from the cache. The *consistency model* in use reflects the degree to which cached replicas are kept consistent. In order to increase document consistency, Web servers and Internet caches employ consistency algorithms that invalidate cached copies upon freshness expiration. Each user request that finds an invalid copy is forwarded to the server, which sends a fresh object back to the user, and the stale cached replica is replaced with the latest version.

In reality, Web pages are updated at rates that are determined by the application of the hosting server, the services offered in each document, as well as the rollout

schedule for the production server. Consistency assurance becomes more challenging when server documents are updated rapidly (e.g., weather updates, sports scores, news portals), since frequent server-cache communications are required to properly track the freshness at the cache. Furthermore, since all communications between the server and the cache are subject to constraints imposed by the network infrastructure, consistency is impeded by the network transmit delays. This fact is noticeable in wireless applications, mobile devices, and satellite systems, where the non-negligible communication latency can significantly impact the QoD [72].

6.2 Consistency algorithms on the Web

Consistency issues have been extensively studied in distributed shared memories [47], hierarchical virtual memories [63], network and file sharing systems [48], and distributed databases [30]. In spite of this fact, analytical models and consistency metrics developed for those earlier systems cannot be applied toward the performance evaluation of Web consistency algorithms in a straightforward manner; operational rules pertaining to those systems do not coincide with those used on the Web.[1]

On the Internet, algorithms fall in one of three categories, namely *Time-To-Live* (TTL), *client polling*, and *server invalidation*. Operational principals characterizing algorithms in each category are now described below.

[1]Additional details regarding individual system characteristics and (dis)similarities to Internet caches can be found in [18, 35, 42] and their references.

6.2.1 TTL algorithms

When TTL algorithms are employed, each document accepted by the cache is equipped with a Time-To-Live (TTL) parameter, which is either provided by the server or determined at the cache. Once a document is placed in the cache, it is considered *valid* until the time specified by the Time-To-Live elapses. A user request that finds a valid cached copy incurs a *cache-hit* and is served directly by the cache. On the other hand, requests presented after the Time-To-Live has expired, are *cache-miss* requests that are forwarded to the server, which in turn sends a fresh copy to the cache, and from there to the user.

TTL algorithms are widely implemented in general Internet applications, e.g., DNS, FTP, and HTTP caches [21, 40, 55, 56]. This fact is a direct consequence of their simplicity, sufficiently good performance, the flexibility to assign a TTL value to a single cached data item, and the ease of deployment in hierarchical caching systems [34]. In the literature, several TTL protocols are available, with each algorithm utilizing a distinct Time-To-Live calculation technique. Commonly encountered algorithms include the *fixed TTL* whereby the Time-To-Live is always set to a constant T [21, 22, 23, 24, 34, 40], the *LM-Factor* used in Squid caches that calculates the Time-To-Live based on the last master modification time [18, 77], and the non-causal *perfect TTL* (also referred to as the *optimal TTL* algorithm or *precise-expiration* protocol) according to which the Time-To-Live expires at the exact moment of next master update [44, 54, 79].

6.2.2 Client polling

A client polling consistency algorithm is invoked according to a schedule that is dictated by the cache. Each time it runs, the algorithm connects to the server and initiates an *If-Modified-Since* request, accompanied by the identifiers (i.e., *validators* [28]) of one or more cached items. Commonly used validators include the *Last-Modified-Timestamp*, document version, header, and the *entity-tag* (ETag). If the server finds that the value of the validators match those of the master (i.e., the cached copy is up to date), it sends a *304 Not-Modified* reply to the cache, and the cache continues to serve the valid copy in response to user requests; otherwise the server returns a *200 OK* message together with the latest master version, and the stale cached copy is replaced with the new download.

Practical implementations of client polling protocols include the *if-modified-since* algorithm (also known as the *polling-every-time* protocol) under which every request that arrives to the cache prompts an If-Modified-Since message to the server, the *periodic-polling* method that connects to the server every P time units [57], and the *piggyback-cache-validation* whereby freshness control messages are embedded in other (i.e., non-consistency related) transmissions to the server.

In spite of their simplicity, polling algorithms are not favored in general Internet applications. The if-modified-since algorithm clearly results in good QoD, but can significantly increase the server and network loads, and user-service latency [79]. Other client polling protocols that are less resource intensive than the if-modified-since algorithm typically result in poor cache consistency.

6.2.3 Server invalidation

Server invalidation algorithms are invoked by the server upon each update to the master (or shortly thereafter). Once launched, the algorithm connects to one or more caches that contain a copy of the modified document, and informs each cache of the changes. Some algorithms perform this notification by sending a new master copy to the cache. Other algorithms only invalidate the stored copies, and the first request that arrives after the invalidation is a *cache-miss* that is forwarded to the server.

Invalidation protocols are implemented in many commercial Web caches, and are particularly popular in servers that contain dynamically generated Web pages. Common protocols include the *replication* algorithm that forwards the master changes to all caches immediately after the update [79], the *invalidation-report* used in wireless and mobile applications to inform end-user caches of content updates [80], as well as the *piggyback-server-invalidation* under which the invalidation messages are encapsulated in non-consistency related communications with the cache [45].

The main advantage of invalidation algorithms lies in the high degree of cache consistency attained by most protocols in this category. However, other (e.g., TTL) consistency solutions are preferred in general, since a server that employs an invalidation algorithm must maintain a list of all system caches.

6.3 Weak vs. strong consistency models

In order to capture the freshness of a cached item with respect to the master, we find it useful to introduce the following freshness classification: A *server-fresh* document is defined as the latest serviceable document at the server, whereas a *cache-fresh* document is defined as a valid cached copy, or equivalently, as one thought to be server-

fresh by the cache. We define a *cache-fresh** document as one that is both cache-fresh and server-fresh, and a document that is not cache-fresh* is termed *cache-stale**.

The schedule of the consistency algorithm and the non-zero server-cache transmit delay Δ make it possible for a cached item to be considered valid by the cache despite the availability of a later (i.e., fresher) version at the server. Such occurrences would result in the undesirable download of cache-stale* (thus inconsistent) documents from the cache. Following Cao and Liu [18], *strong consistency* algorithms are characterized by users being served strictly server-fresh documents under zero delays and processing times. A consistency algorithm that is not strong is termed *weak*, in which case there is a possibility that users download inconsistent copies (i.e., ones that are cache-fresh yet not server-fresh), even under zero delays.

In general, causal TTL and most client polling algorithms are weakly consistent, whereas the if-modified-since, the perfect TTL, and most server invalidation protocols are strongly consistent. To appreciate this fact, consider the fixed TTL and the if-modified-since algorithms, when all delays are zero:

- With the fixed TTL algorithm, each document placed in the cache remains valid for T time units, and all requests presented until the TTL expiration are served directly from the cache. If the master is updated prior to the expiration, then all requests that arrive after the update yet before the TTL has expired are served with a cache-stale* copy. The fixed TTL is therefore a weak consistency protocol.
- The if-modified-since algorithm sends a conditional *GET* directive to the server in response to every user request that arrives to the cache. If the server finds that the cached item is fresh, a *304 Not-Modified* message is returned to the cache; otherwise the server forwards the latest master version and the new object is loaded into the cache. Either way, once the server-cache communications complete, the re-

quested document is sent to the user from the cache. If $\Delta = 0$ (i.e., *instantaneous* transmissions) then users are always served with server-fresh copies, hence the if-modified-since is a strong consistency protocol.

6.4 Quantifying cache consistency and QoD

For a given consistency algorithm, we define a *cache-hit** as a cache-hit that results in a server-fresh download from the cache to the user. The complementary situation of a cache-stale* download is termed a *cache-miss**. Each cache-hit* is therefore a cache-hit, but the reverse clearly does *not* hold. The hit rate and the hit* rate are then the long-run average download rates of cache-fresh and cache-fresh* documents from the cache, respectively. Since each cache-hit* is necessarily a cache-hit, the hit* rate can never exceed the hit rate, and equality is achieved for strong consistency algorithms if $\Delta = 0$.

The hit* rate is most useful for measuring the fraction of fresh downloads when the cache utilizes weak protocols, for then the likelihood that users retrieve stale objects is potentially large: Consider the fixed TTL algorithm when T is very large, and server documents are updated frequently. A large value for T guarantees a high hit rate and lowered bandwidth utilization [34, 40], but results in poor quality of data (QoD), even when the communication latency is negligible (e.g., broadband connectivity). Measuring consistency is also important for strong consistency mechanisms deployed in a hierarchy of caches, or when the delay is large (e.g., satellite and wireless networks [67, 80]), as previously concluded in [73].

In order to evaluate the consistency performance under a given protocol, we formulate a framework that allows consistency issues to be investigated in a quantitative

manner. This framework is presented in Chapter 7, where the focus is on a single server-cache pair and a single data object, as we attempt to isolate relevant issues. User requests and master updates are modeled by mutually independent point processes. Then, the hit and hit* rates of any given consistency algorithm can in principle be evaluated, and various design parameters could be tuned on the basis of the resulting performance.

In Chapters 8 and 9, we apply the modeling framework to the analysis of the fixed and perfect TTL algorithms, respectively. Hit rate and hit* rate results are obtained for each protocol under any value of the download delay $\Delta \geq 0$. Closed form expressions for the calculated rates are available in some special cases when the requests are generated according to a Poisson process. Computable bounds are derived for each algorithm when the request process is a renewal process, and are tightened when the inter-request time distribution belongs to certain subclasses of distributions (e.g., IFR, DFR, NBUE and NWUE). On the basis of these results, we can now explore the degradation of the hit and hit* rates as a function of the communication delay Δ, and of additional algorithm-specific parameters.

To conclude, we note that the proposed modeling framework can also be used to investigate consistency issues in polling and invalidation algorithms, as well as techniques employed by other distributed systems (e.g., virtual shared memories and file sharing systems [1, 35, 63]).

Chapter 7

A Framework for Measuring Cache Consistency

7.1 The system model

The system is made up of a site called the *origin* where the current authoritative version of the data is maintained, and of *requestors*. Each requestor is identified with a cache that is used either by users or by client-caches. Thus, the origin and requestors are synonymous with server and caches, respectively.

Caches are assumed to be of *infinite* size, reflecting the fact that storage is ample. The need to specify a replacement policy is moot, and only the operational rules of the consistency algorithms matter. In particular, once a document has been placed in the cache, a copy is always available at the requesting cache, although said copy may be either fresh or stale at any given time.

Under these circumstances, there is no loss of generality in abstracting a caching system into a single cache-server pair, and in considering a single cacheable data

item, say D, in isolation, as we do from here on.

7.1.1 Modeling requests

User requests for the document D arrive according to a point process $\{T_n^r, n = 0, 1, \ldots\}$ with the understanding that the n^{th} request occurs at time T_n^r. Thus, $T_n^r \leq T_{n+1}^r$ for each $n = 0, 1, \ldots$ with $T_0^r = 0$. Let $\{R_{n+1},\ n = 0, 1, \ldots\}$ denote the sequence of inter-request times with $R_{n+1} = T_{n+1}^r - T_n^r$ for each $n = 0, 1, \ldots$. The point process $\{T_n^r, n = 0, 1, \ldots\}$ is assumed to be *simple* in that $R_{n+1} > 0$ a.s. for $n = 0, 1, \ldots$, so multiple requests cannot occur simultaneously.

As customary, the counting process $\{R(t),\ t \geq 0\}$ associated with the point process $\{T_n^r, n = 0, 1, \ldots\}$ is given by

$$R(t) = \sup\{n = 0, 1, \ldots :\ T_n^r \leq t\}, \quad t \geq 0 \tag{7.1}$$

so that $R(t)$ counts the number of requests in the interval $(0, t]$. The corresponding residual lifetime process $\{R_e(t),\ t \geq 0\}$ is defined by

$$R_e(t) = T_{R(t)+1}^r - t, \quad t \geq 0.$$

If $T_n^r \leq t < T_{n+1}^r$ for some $n = 0, 1, \ldots$, then $R(t) = n$ and $R_e(t) = T_{n+1}^r - t$, i.e., $R_e(t)$ represents the amount of time that will elapse until the occurrence of the next request after time t.

We assume at minimum that the point process $\{T_n^r,\ n = 0, 1, \ldots\}$ admits a rate in the sense that there exists a finite constant $\lambda_R > 0$ given by the limit

$$\lambda_R = \lim_{t \to \infty} \frac{R(t)}{t} \quad a.s.$$

We refer to λ_R as the request rate.

7.1.2 Modeling document updates

The document D changes over time, and is updated according to the second point process $\{T_m^u, m = 0, 1, \ldots\}$ where T_m^u is the epoch at which the m^{th} update takes place. We denote by $\{U_{m+1},\ m = 0, 1, \ldots\}$ the sequence of inter-update times with $U_{m+1} = T_{m+1}^u - T_m^u$ for each $m = 0, 1, \ldots$. Here as well, $T_m^u \leq T_{m+1}^u$ for each $m = 0, 1, \ldots$ with $T_0^u = 0$. The point process $\{T_m^u, m = 0, 1, \ldots\}$ is assumed to be simple so that multiple updates are ruled out.

In analogy with (C.8), the counting process $\{U(t),\ t \geq 0\}$ associated with the point process $\{T_m^u,\ m = 0, 1, \ldots\}$ is given by

$$U(t) = \sup\{m = 0, 1, \ldots :\ T_m^u \leq t\}, \quad t \geq 0$$

with $U(t)$ counting the number of updates in the interval $(0, t]$. The corresponding residual lifetime process $\{U_e(t),\ t \geq 0\}$ is defined by

$$U_e(t) = T_{U(t)+1}^u - t, \quad t \geq 0$$

so that $U_e(t)$ represents the amount of time that will elapse until the next update after t.

As before, we assume that the process $\{T_m^u, m = 0, 1, \ldots\}$ admits a rate, referred to as the update rate, in the sense that there exists a finite constant $\lambda_U > 0$ given by

$$\lambda_U = \lim_{t\to\infty} \frac{U(t)}{t} \quad a.s.$$

7.2 Basic assumptions

Throughout, the point processes $\{T_n^r, n = 0, 1, \ldots\}$ and $\{T_m^u, m = 0, 1, \ldots\}$ are assumed to be *mutually independent*. This reflects the lack of correlation between user behavior and the evolution of data content.

For reasons of brevity and mathematical simplicity, these point processes are assumed to be *renewal* processes. For the requests, this means that the inter-request times $\{R_{n+1},\ n = 0, 1, \ldots\}$ form a sequence of i.i.d. rvs distributed according to the common cdf F_R. Let R denote any rv distributed according to F_R. Similarly, when the update process is a renewal process, the inter-update times $\{U_{m+1},\ m = 0, 1, \ldots\}$ form a sequence of i.i.d. rvs distributed according to the cdf F_U. We denote by U any rv distributed according to F_U. The cdfs F_R and F_U are assumed to have finite mean $m(F_R)$ and $m(F_U)$, in which case it is well known [69] that $\lambda_R = m(F_R)^{-1}$ and $\lambda_U = m(F_U)^{-1}$.

Let $\Delta \geq 0$ denote the fixed download delay of D over the network, i.e., if a document is sent from the server at time t, it is received by the cache at time $t + \Delta$, at which point it is ready for access by the users. In the other direction, communication from the cache to the server is deemed *instantaneous* as it entails the transmission of very short control messages.

7.3 Hit rates and QoD

With a given consistency algorithm, we can associate the two performance measures mentioned earlier, namely the hit and hit* rates. These metrics reflect the quality (or freshness) of D from two different viewpoints, namely cache freshness and server freshness, and capture the fraction of hit and hit* occurrences out of all user requests, respectively.

In order to count the number of cache-hit instances under the network delay Δ, an $\mathbb{R}_+$-valued validator process $\{L_\Delta(t),\ t \geq 0\}$ is introduced to track the cache freshness of the cached copy of D; a hit occurs at request time T_n^r *if and only if*

$L_\Delta(T_n^r-) > 0$. The hit rate is then simply defined by

$$H(\Delta) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[L_\Delta(T_n^r-) > 0\right]. \tag{7.2}$$

Similarly, in order to identify cache-hit* requests, the server freshness of the cached version of D is monitored through another $\mathbb{R}_+$-valued process $\{L_\Delta^\star(t),\ t \geq 0\}$, so that a hit* occurs at request time T_n^r *if and only if* $L_\Delta^\star(T_n^r-) > 0$, and the hit* rate is defined as

$$H^\star(\Delta) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[L_\Delta^\star(T_n^r-) > 0\right]. \tag{7.3}$$

The limits in both (7.2) and (7.3) are taken in the a.s. sense and are assumed to exist with $H(\Delta)$ and $H^\star(\Delta)$ *constants*. This will be the case for all algorithms considered in this work. In that case, the a.s. limit

$$\lim_{N\to\infty} \frac{\sum_{n=1}^{N} \mathbf{1}\left[L_\Delta^*(T_n^r-) > 0\right]}{\sum_{n=1}^{N} \mathbf{1}\left[L_\Delta(T_n^r-) > 0\right]} = \frac{H^\star(\Delta)}{H(\Delta)} \tag{7.4}$$

also exists as a constant. This ratio represents the fraction of server-fresh hits out of all hits, and therefore measures the QoD produced by a given algorithm.

7.4 Requests and updates in applications

The experimental validation of the proposed model has already been carried out in numerous studies for the case of $\Delta = 0$ (e.g., see [10, 27, 60, 62, 67, 79] and references therein). In these studies, request epochs and update timestamps are extracted from log files of popular caches in order to calculate the total number of hit* occurrences among all user requests.

The selection of the distributions F_R and F_U that best models arrivals and updates is specific to each Internet application. Inter-request times in HTTP and FTP caches

follow the Weibull and Pareto heavy tailed distributions, as reported in [27, 62] and emphasized by Bestavros et al. in [10]. Locality of reference, whereby recently accessed documents are likely to be shortly requested again, can be (partially) expressed through an appropriate selection of F_R [29, 64]. Web pages that contain stocks and weather information on the Yahoo portal are updated periodically every few seconds [67]; logs collected from news servers [60] and Harvest caches [18] suggest a bimodal inter-update time distribution.

As we shall later observe, the expressions for the hit and hit* rates for the consistency algorithms analyzed in the next chapters depend crucially on the (non-delayed) renewal function $t \rightarrow \mathbf{E}\left[R(t)\right],\ t \geq 0$. It is therefore rather difficult (if at all possible) to calculate these rates for general applications owing to the simple fact that $\mathbf{E}\left[R(t)\right]$ is not known in closed form except in some special cases (e.g., when R is lattice or uniformly distributed, or for a class of matrix-exponential distributions [5]). In order to circumvent this difficulty and apply the obtained results in general applications, we derive distribution-free bounds on the calculated rates with the help of well known bounds on the renewal function. Bounds on the hit and hit* rates are then tightened when F_R belongs to several subclasses of distributions of interest, as well as for specific inter-request time distributions.

7.5 Bounds on the renewal function

7.5.1 Distribution-free bounds

Bounds are available for the renewal function associated with any distribution F_R. First, recall that the forward recurrence time rv R_e associated with the rv R is dis-

tributed according to

$$\mathbf{P}\left[R_e \leq t\right] = \lambda_R \int_0^t \mathbf{P}\left[R > u\right] du, \quad t \geq 0. \tag{7.5}$$

Lorden [50] has shown the upper bound

$$\mathbf{E}\left[R(t)\right] \;\leq\; \lambda_R t + \lambda_R^2 \mathbf{E}\left[R^2\right] - 1, \quad t \geq 0 \tag{7.6}$$

while Marshall [53] proved the lower bound

$$\mathbf{E}\left[R(t)\right] \;\geq\; \lambda_R t - \mathbf{P}\left[R_e \leq t\right] \;\geq\; \lambda_R t - 1, \quad t \geq 0. \tag{7.7}$$

We refer to these results as distribution-free bounds as they only depend on the first and second moments of the distribution F_R. These bounds may not be useful when $\lambda_R^2 \mathbf{E}\left[R^2\right]$ or $\mathbf{P}\left[R_e \leq t\right]$ are large for then there is a risk that the upper bound (7.6) is too loose, or the lower bound (7.7) close to zero for $t \leq \mathbf{E}\left[R\right]$ (hence useless).[1]

To avoid the technical challenges associated with the distribution-free bounds (7.6) and (7.7), we now seek alternative bounds that do not suffer from the disadvantages mentioned above. We achieve this goal by focusing the discussion on subclasses of distributions of interest (i.e., ones that are commonly used in network traffic modeling), for which tighter bounds are known to exist.

7.5.2 NBUE and NWUE distributions

A distribution F_R defined on $[0, \infty)$ is said to be Increasing Failure Rate, denoted IFR, if the mapping

$$t \to \frac{\mathbf{P}\left[R > t + r\right]}{\mathbf{P}\left[R > t\right]}, \quad t \geq 0 \tag{7.8}$$

[1] $\lambda_R^2 \mathbf{E}\left[R^2\right] \geq 1$ always since this is equivalent to $\mathrm{var}(R) \geq 0$. Similarly, $\mathbf{P}\left[R_e \leq t\right] \leq \lambda_R t$ is a simple consequence of (7.5).

is non-increasing in t for each $r \geq 0$. Conversely, if the mapping (7.8) is non-decreasing then F_R is termed Decreasing Failure Rate, or DFR in short.[2]

If a distribution F_R on $[0, \infty)$ satisfies the condition

$$\mathbf{P}\left[R_e > t\right] \leq \mathbf{P}\left[R > t\right], \quad t \geq 0,$$

then F_R belongs to the class of New Better Than Used in Expectation (NBUE) distributions, and the associated renewal function is bounded from above by

$$\mathbf{E}\left[R(t)\right] \leq \lambda_R t, \quad t \geq 0. \tag{7.9}$$

On the other hand, when

$$\mathbf{P}\left[R_e > t\right] \geq \mathbf{P}\left[R > t\right], \quad t \geq 0,$$

then we say that F_R is New Worse Than Used in Expectation (NWUE). In this case, the corresponding renewal function is bounded from below by

$$\mathbf{E}\left[R(t)\right] \geq \lambda_R t, \quad t \geq 0. \tag{7.10}$$

It is well known [12] that if F_R is IFR (equivalently, DFR), then it is also NBUE (equivalently, NWUE).

To summarize, if F_R is either IFR or NBUE, then the following bounds on the renewal function are readily available from (7.7) and (7.9), namely

$$\lambda_R t - \mathbf{P}\left[R_e \leq t\right] \leq \mathbf{E}\left[R(t)\right] \leq \lambda_R t, \quad t \geq 0. \tag{7.11}$$

In a similar manner, the combination of (7.6) and (7.10) for F_R NWUE yields

$$\lambda_R t \leq \mathbf{E}\left[R(t)\right] \leq \lambda_R t + \lambda_R^2 \mathbf{E}\left[R^2\right] - 1, \quad t \geq 0. \tag{7.12}$$

[2]Much of this material can be found in the monograph by Barlow and Proschan [12].

Finally, if F_R is DFR, then a tighter upper bound than the one specified in (7.6) is reported by Brown [16]. Combining the upper bound by Brown with the lower bound (7.10) gives

$$\lambda_R t \;\leq\; \mathbf{E}\left[R(t)\right] \;\leq\; \lambda_R t + \frac{\lambda_R^2 \mathbf{E}\left[R^2\right]}{2} - 1, \quad t \geq 0. \tag{7.13}$$

7.5.3 Distribution-specific bounds

Linear bounds on the renewal function are presented by Marshall [53]. These bounds are specific for each given distribution and can be used to further improve the bounds on the renewal function listed thus far.

For a given distribution F_R on $[0, \infty)$, define

$$b_\ell \;=\; \inf_{t \in A} \frac{\mathbf{P}\left[R \leq t\right] - \mathbf{P}\left[R_e \leq t\right]}{1 - \mathbf{P}\left[R \leq t\right]} \;\leq\; 0 \tag{7.14}$$

and

$$b_u \;=\; \sup_{t \in A} \frac{\mathbf{P}\left[R \leq t\right] - \mathbf{P}\left[R_e \leq t\right]}{1 - \mathbf{P}\left[R \leq t\right]} \;\geq\; 0 \tag{7.15}$$

where $A = \{t \geq 0 : \mathbf{P}\left[R \leq t\right] < 1\}$. Then, the renewal function associated with the distribution F_R is bounded from above by

$$\mathbf{E}\left[R(t)\right] \;\leq\; \lambda_R t + b_u, \tag{7.16}$$

and from below by

$$\mathbf{E}\left[R(t)\right] \;\geq\; \lambda_R t + b_\ell \tag{7.17}$$

for all values of $t \geq 0$.

Previous comments regarding NBUE and NWUE distributions allow the linear bounds (7.16) and (7.17) to be sharpened: Whenever F_R is NBUE we have

$$\lambda_R t \;\leq\; \mathbf{E}\left[R(t)\right] \;\leq\; \lambda_R t + b_u, \quad t \geq 0, \tag{7.18}$$

and whenever F_R is NWUE we conclude

$$\lambda_R t + b_\ell \;\leq\; \mathbf{E}\left[R(t)\right] \;\leq\; \lambda_R t, \quad t \geq 0. \tag{7.19}$$

7.6 Poisson, Weibull, and Pareto requests

The Poisson, Weibull, and Pareto distributions are often used to model time gaps in network packet arrivals, as well as inter-request times experienced by Web caching systems. Consequently, special attention is given to evaluating the hit and hit* rates under these inter-request time distributions, for each of the algorithms examined in the following chapters.

Poisson requests. Poisson requests correspond to the generic inter-request time rv R being exponentially distributed, say with rate λ_R, i.e.,

$$F_R(t) \;=\; 1 - e^{-\lambda_R t}, \quad t \geq 0.$$

In this case, the renewal function is available in closed form [69] with

$$\mathbf{E}\left[R(t)\right] \;=\; \lambda_R t, \quad t \geq 0.$$

This fact permits the derivation of closed form expressions for the hit and hit* rates, provided that the inter-update time distribution F_U is also at hand.

Of even greater interest are the Weibull and Pareto distributions, which were proved more suitable (than the exponential distribution) in representing inter-arrival times of user requests at the cache [27, 39, 62].

Weibull requests. The Weibull distribution is characterized by

$$F_R(t) \;=\; 1 - e^{-(\beta t)^\alpha}, \quad t \geq 0,\ \alpha, \beta > 0,$$

with the two first moments given by

$$\mathbf{E}\left[R\right] = \frac{1}{\beta}\,\Gamma(\alpha^{-1}+1)$$

and

$$\mathbf{E}\left[R^2\right] = \frac{1}{\beta}\,\Gamma(\frac{2}{\alpha}+1)$$

where $\Gamma(t)$ is the Gamma function defined through

$$\Gamma(t) = \int_0^\infty u^{t-1}e^{-u}\,du\;, \quad t>0.$$

The residual lifetime rv R_e associated with the Weibull rv R has the distribution

$$\mathbf{P}\left[R_e \le t\right] \;=\; \frac{\lambda_R}{\alpha\beta}\cdot\gamma(\frac{1}{\alpha},(\beta t)^\alpha), \quad t\ge 0$$

with $\gamma(t,r)$ denoting the lower incomplete Gamma function given by

$$\gamma(t,r) = \int_0^r u^{t-1}e^{-u}\,du, \quad t>0,\; r\ge 0.$$

It is well known [12] that the Weilbull distribution is DFR for $\alpha \le 1$. Thus, the DFR bounds in (7.13) can now be used to obtain bounds on the renewal function associated with the Weibull inter-request time distribution, in the process yielding

$$\lambda_R t \;\le\; \mathbf{E}\left[R(t)\right] \;\le \lambda_R t + \frac{{\lambda_R}^2}{2\beta^2}\Gamma(1+\frac{2}{\alpha}) - 1, \quad t\ge 0.$$

On the other hand, when $\alpha \ge 1$ the Weibull distribution is IFR [12], and the appropriate IFR bounds (7.11) provide us with

$$\lambda_R t - \frac{\lambda_R}{\alpha\beta}\cdot\gamma\left(\frac{1}{\alpha},(\beta t)^\alpha\right) \;\le\; \mathbf{E}\left[R(t)\right] \;\le\; \lambda_R t, \quad t\ge 0.$$

Here, the Marshall bounds (7.16) and (7.17) are omitted by the simple fact that $b_u = \infty$ for $\alpha < 1$ and $b_l = -1$ whenever $\alpha > 1$, so the resulting bounds are less effective than those listed above.

Pareto requests. Similar arguments apply to the Pareto inter-request time distribution defined by

$$F_R(t) = 1 - \left(\frac{k}{k+t}\right)^{\alpha}, \quad t \geq 0,\ \alpha, k > 0.$$

This distribution is DFR, and we restrict the discussion to $\alpha > 2$, in which case the first two moments

$$\mathbf{E}[R] = \frac{k}{\alpha - 1} \tag{7.20}$$

and

$$\mathbf{E}\left[R^2\right] = \frac{2k^2}{(\alpha-2)(\alpha-1)}$$

are finite. In this case, the distribution of the residual lifetime rv R_e is given by

$$\mathbf{P}[R_e \leq t] = 1 - \left(\frac{k}{k+t}\right)^{\alpha-1}, \quad t \geq 0.$$

It is a simple matter to check that the DFR bounds in (7.13) yield

$$\lambda_R t \ \leq\ \mathbf{E}[R(t)] \ \leq\ \lambda_R + \frac{1}{\alpha - 2}, \quad t \geq 0. \tag{7.21}$$

In addition, since $b_u = 1$ in the Marshall bound (7.18), alternative bounds are readily available in the form

$$\lambda_R t \ \leq\ \mathbf{E}[R(t)] \ \leq\ \lambda_R + 1, \quad t \geq 0. \tag{7.22}$$

Upon making use of these results, we conclude that

$$\lambda_R t \ \leq\ \mathbf{E}[R(t)] \ \leq\ \lambda_R + \min\left(\frac{1}{\alpha - 2}, 1\right), \quad t \geq 0$$

for all values of $\alpha > 2$.

Chapter 8

The Fixed TTL Algorithm

8.1 Operational rules of the fixed TTL

With the nomenclature introduced in Chapter 7, the fixed TTL algorithm can be described as follows: Whenever the server receives a request for D, say at time T_n^r for some $n = 0, 1, \ldots$, it returns the current version together with the TTL field $T > 0$. The first miss request for D arrives to the cache at $T_0^r = 0$, and the cache makes the returned version available at $T_0^r + \Delta$. Any subsequent request for D at the cache in the time interval $(T_0^r + \Delta, T_0^r + \Delta + T)$ is served directly from the cache without contacting the server.

If a request at time T_n^r for some $n = 1, 2, \ldots$ is the first one presented after the TTL has expired, then it is forwarded to the server, in which case requests arriving during the interval $(T_n^r + \Delta, T_n^r + \Delta + T)$ are all hits. The discussion is carried out under the assumption that the requests presented in $(T_n^r, T_n^r + \Delta)$ for $n = 0, 1, \ldots$, are sent to the server as well. However, the copies of D received in response to these requests are neither placed nor do they reset the TTL, as assumed in [18] for the

empirical evaluation of various TTL algorithms.[1]

8.2 Zero delays

In this section we drop Δ from earlier notation as it is now set to zero. Cache freshness of the object D is completely described by the validator process $\{L(t),\ t \geq 0\}$, which continuously tracks the TTL value at the cache. The process has right-continuous sample paths with left limits, and is defined by

$$L(t) = \left(L(T_n^r) - (t - T_n^r)\right)^+, \quad T_n^r \leq t < T_{n+1}^r$$

for each $n = 0, 1, \ldots$, with the update rule

$$L(T_n^r) = \begin{cases} T & \text{if } L(T_n^r-) = 0 \\ L(T_n^r-) & \text{if } L(T_n^r-) > 0 \end{cases}. \tag{8.1}$$

Operational assumptions made earlier lead to the initial condition $L(0-) = 0$ so that $L(0) = T$ by (8.1). The timer will have expired at time $t > 0$ if and only if $L(t-) = 0$. Thus, the n^{th} request at time T_n^r produces a hit if $L(T_n^r-) > 0$; otherwise it will be a miss. The hit rate[2] $H(T)$ for the fixed TTL is now as defined in (7.2). Its evaluation has been carried out already by Jung et al. [40].

Proposition 8.1 *If the point process $\{T_{n+1}^r,\ n = 0, 1, \ldots\}$ is a renewal process, then it holds that*

$$H(T) = \frac{\mathbf{E}\left[R(T-)\right]}{1 + \mathbf{E}\left[R(T-)\right]}.$$

[1] We restrict the analysis to the case $T \geq \Delta$, as customary on the Web.

[2] The notation $H(T)$ is used here to reflect the dependency of the hit rate on the fixed TTL parameter T.

Note that $\mathbf{E}\left[R(T-)\right] = \mathbf{E}\left[R(T)\right]$ for all $T > 0$ as soon as F_R admits a density, a common occurrence in applications. Proposition 8.1 being a special case of Proposition 8.3 with $\Delta = 0$, its proof is therefore omitted.

As we now turn to the hit* rate, we note that even in the absence of transmission delays between the server and the cache, there is a possibility that a request incurs a hit for a stale copy. The consistency of the cached object with that offered by the server is captured by the process $\{L^*(t),\ t \geq 0\}$ which tracks the time until the expiration of the cache-fresh* copy. This process has right-continuous sample paths with left limits, and is defined by

$$L^\star(t) = \left(L^\star(T_n^r) - (t - T_n^r)\right)^+, \quad T_n^r \leq t < T_{n+1}^r$$

for each $n = 0, 1, \ldots$, with the update rule

$$L^\star(T_n^r) = \begin{cases} \min(L(T_n^r), U_e(T_n^r)) & \text{if } L^\star(T_n^r-) = 0 \\ L^\star(T_n^r-) & \text{if } L^\star(T_n^r-) > 0 \end{cases}.$$

The initial condition is taken to be $L^\star(0-) = 0$ so that $L^\star(0) = \min(T, U_e(0))$. The hit* rate $H^\star(T)$ is given by (7.3) with $\{L^*(t),\ t \geq 0\}$ as defined above.

Proposition 8.2 *Under the assumptions of Proposition 8.1, it holds that*

$$H^\star(T) = \frac{\mathbf{E}\left[R(\min(T, U_e)-)\right]}{1 + \mathbf{E}\left[R(T-)\right]} \tag{8.2}$$

provided the point process $\{T_{m+1}^u,\ m = 0, 1, \ldots\}$ *is also a renewal process.*

Proposition 8.2 follows from the analogous result for the general case $\Delta \geq 0$ given in Proposition 8.4.

In this last expression the stationary forward recurrence time U_e is taken to be independent of the counting process $\{R(t),\ t \geq 0\}$; its distribution is given by

$$\mathbf{P}\left[U_e \leq t\right] = \lambda_U \int_0^t \mathbf{P}\left[U > u\right] du, \quad t \geq 0. \tag{8.3}$$

An alternative expression for $H^\star(T)$ follows by specializing (8.8) with $\Delta = 0$, and is given by

$$\begin{aligned} H^\star(T) &= H(T)\mathbf{P}\left[U_e > T\right] + \lambda_U \int_0^T \frac{\mathbf{E}\left[R(u)\right]}{1+\mathbf{E}\left[R((T)-)\right]}\mathbf{P}\left[U > u\right]\, du \\ &= H(T) - \lambda_U \int_0^T \frac{\mathbf{E}\left[R(T-)\right] - \mathbf{E}\left[R(u)\right]}{\mathbf{E}\left[R(T-)\right]+1}\mathbf{P}\left[U > u\right]\, du. \end{aligned} \tag{8.4}$$

8.3 Non-zero delays

In the presence of a network delay $\Delta > 0$, cache freshness is monitored through the validator process $\{L_\Delta(t),\ t \geq 0\}$, which continuously tracks the TTL value at the cache (see Figure 8.1). This process has right-continuous sample paths with left limits, and is defined as follows: First, define the $\mathbb{N}$-valued rvs $\{\mu_k,\ k = 0, 1, \ldots\}$ recursively by

$$\mu_{k+1} = \inf\left\{n > \mu_k :\ T^r_{\mu_k} + \Delta + T \leq T^r_n\right\} \tag{8.5}$$

for each $k = 0, 1, \ldots$ with $\mu_0 = 0$. The rv μ_k identifies the k^{th} request forwarded to the server that resets the TTL value to T, as we recall that requests made in the interval $(T^r_{\mu_k}, T^r_{\mu_k} + \Delta)$ are forwarded as well, but do not affect the TTL at the cache in that interval. For each $k = 0, 1, \ldots$ we can then write

$$L_\Delta(t) = \left(L_\Delta(T^r_{\mu_k} + \Delta) - (t - (T^r_{\mu_k} + \Delta))\right)^+$$

on the interval $[T^r_{\mu_k} + \Delta, T^r_{\mu_{k+1}} + \Delta)$, with the update rule $L_\Delta(T^r_{\mu_k} + \Delta) = T$. We initially take $L_\Delta(t) = 0$ for $0 \leq t < \Delta$, and the hit rate $H(T, \Delta)$ can be written as in (7.2) with $\{L_\Delta(t),\ t \geq 0\}$.

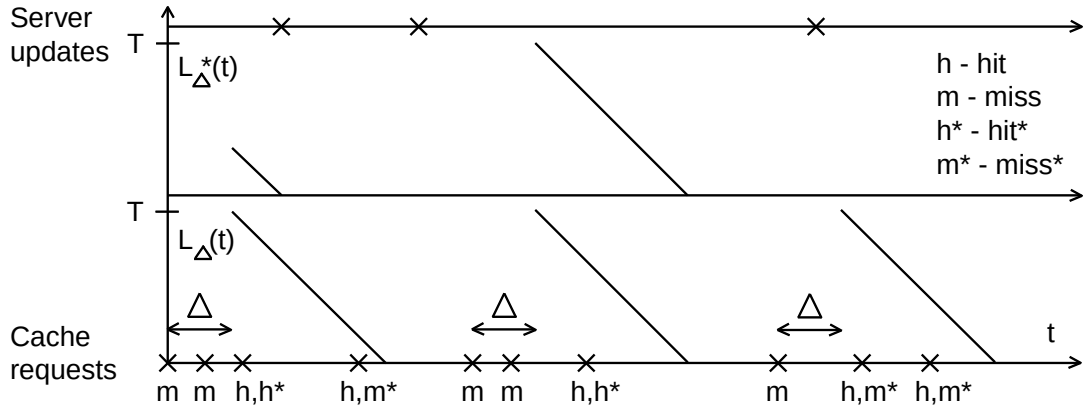

Figure 8.1: A time line diagram of requests, updates, and the freshness tracking processes for the fixed TTL with $\Delta > 0$.

Proposition 8.3 *If the point process* $\{T^r_{n+1},\ n = 0, 1, \ldots\}$ *is a renewal process, then we have*

$$H(T, \Delta) = \frac{\mathbf{E}\left[R((T+\Delta)-)\right] - \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T+\Delta)-)\right]} \tag{8.6}$$

for each $\Delta \geq 0$.

As expected, this result specializes for $\Delta = 0$ to the one stated in Proposition 8.1 and in [40]. A proof of Proposition 8.3 is available in Appendix B.

Next, the evaluation of the hit* rate is made possible through the server-freshness tracking process $\{L^\star_\Delta(t),\ t \geq 0\}$ at the cache. This process has right-continuous sample paths with left limits, and is defined as follows: For each $n = 0, 1, \ldots$ we set

$$L^\star_\Delta(t) = \left(L^\star_\Delta(T^r_n + \Delta) - (t - (T^r_n + \Delta))\right)^+$$

whenever $T^r_n + \Delta \leq t < T^r_{n+1} + \Delta$ with the update

$$L^\star_\Delta(T^r_n + \Delta) = \begin{cases} \min(L_\Delta(T^r_n + \Delta), (U_e(T^r_n) - \Delta)^+) & \text{if } L^\star_\Delta((T^r_n + \Delta)-) = 0 \\ L^\star_\Delta((T^r_n + \Delta)-) & \text{if } L^\star_\Delta((T^r_n + \Delta)-) > 0 \end{cases}.$$

The initial conditions are taken to be $L^*_\Delta(t) = 0,\ 0 \le t < \Delta$, as illustrated in Figure 8.1. The hit* rate $H^\star(T, \Delta)$ is given by (7.3), this time with $\{L^*_\Delta(t),\ t \ge 0\}$, and is evaluated in the following proposition.

Proposition 8.4 *Under the assumptions of Proposition 8.2, we have*

$$H^\star(T, \Delta) = \frac{\mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \tag{8.7}$$

for each $\Delta \ge 0$.

As before, U_e is taken to be independent of the counting process $\{R(t),\ t \ge 0\}$, which allows us to rewrite the hit* rate as

$$\begin{aligned} &H^\star(T, \Delta) \qquad\qquad (8.8)\\ &= H(T, \Delta)\mathbf{P}\left[U_e > T + \Delta\right] + \lambda_U \int_\Delta^{T+\Delta} \frac{\mathbf{E}\left[R(u)\right] - \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \mathbf{P}\left[U > u\right]\, du \\ &= H(T, \Delta)\mathbf{P}\left[U_e > \Delta\right] - \lambda_U \int_\Delta^{T+\Delta} \frac{\mathbf{E}\left[R((T + \Delta)-)\right] - \mathbf{E}\left[R(u)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \mathbf{P}\left[U > u\right]\, du. \end{aligned}$$

The result (8.7) reduces to Proposition 8.2 for the case of $\Delta = 0$. A proof of Proposition 8.4, as well as the derivation of the alternative expression (8.8), are provided in Appendix B.

8.4 Properties of the hit and hit* rates

Relationships between the derived rates are readily obtained from (8.8) for any value of $\Delta \ge 0$, under the assumptions of Proposition 8.2. The ratio (7.4) that captures the fraction of non-stale hits satisfies the bounds

$$\mathbf{P}\left[U_e > T + \Delta\right] \le \frac{H^*(T, \Delta)}{H(T, \Delta)} \le \mathbf{P}\left[U_e > \Delta\right], \tag{8.9}$$

which clearly show the interplay between network delays and update statistics.

The ability of the fixed TTL to ensure consistency is degraded as the delay increases, as is the case for most algorithms in practice. Better performance can be achieved by lowering the value of T, yet it is possible that frequent updates prevent users from ever being served with server-fresh data.[3] Furthermore, if documents are rarely updated, i.e., $\lambda_U \simeq 0$, then (8.9) implies $H^*(T, \Delta) \simeq H(T, \Delta)$ as would be expected.

Key to understanding the performance of practical Web caches under the fixed TTL algorithm are the effects of T and Δ on the hit and hit* rates. To do so, we now focus on monotonicity properties and on the asymptotics of the calculated rates as a function of these system parameters.

Lemma 8.1 *For any inter-request time distribution and fixed delay $\Delta \geq 0$, the hit rate produced by the fixed TTL is a non-decreasing function of T.*

The proof of Lemma 8.1 is immediate and is therefore omitted.

Lemma 8.2 *Assume that $\mathbf{E}[R^2]$ and $\mathbf{E}[U^2]$ are finite. Then, for each $\Delta \geq 0$, we have the asymptotics*

$$\lim_{T \to \infty} H(T, \Delta) = 1$$

but

$$\lim_{T \to \infty} H^\star(T, \Delta) = 0.$$

[3] e.g., when the master is updated every Δ time units in which case $\mathbf{P}[U_e > \Delta] = 0$.

Proof. Starting with the hit rate, we note by the Basic Renewal Theorem [69] that

$$\lim_{t\to\infty} \frac{\mathbf{E}\left[R(t)\right]}{t} = \lambda_R \tag{8.10}$$

and the desired conclusion follows since

$$\lim_{T\to\infty} \frac{\mathbf{E}\left[R((T+\Delta)-)\right] - \mathbf{E}\left[R(\Delta)\right]}{T+\Delta} \cdot \frac{T+\Delta}{1+\mathbf{E}\left[R((T+\Delta)-)\right]} = 1.$$

For the hit* rate, applying the distribution-free bounds (7.6) and (7.7) we get the inequality

$$\frac{\mathbf{E}\left[R(u)\right] - \mathbf{E}\left[R(\Delta)\right]}{1+\mathbf{E}\left[R(T+\Delta)-\right]} \;\leq\; \frac{\lambda_R u + \lambda_R^2 \mathbf{E}\left[R^2\right]}{\lambda_R(T+\Delta)}, \quad \Delta \leq u,$$

which leads to the result

$$\begin{aligned}
& \lim_{T\to\infty} H^\star(T,\Delta) \\
= \;& \lim_{T\to\infty} \left\{ H(T,\Delta)\mathbf{P}\left[U_e > T+\Delta\right] + \lambda_U \int_\Delta^{T+\Delta} \frac{\mathbf{E}\left[R(u)\right] - \mathbf{E}\left[R(\Delta)\right]}{1+\mathbf{E}\left[R((T+\Delta)-)\right]} \mathbf{P}\left[U > u\right]\, du \right\} \\
\leq \;& \lim_{T\to\infty} \frac{\lambda_U}{\lambda_R(T+\Delta)} \int_\Delta^{T+\Delta} \left(\lambda_R u + \lambda_R^2 \mathbf{E}\left[R^2\right]\right) \mathbf{P}\left[U > u\right]\, du \\
\leq \;& \lim_{T\to\infty} \frac{\lambda_U \lambda_R \mathbf{E}\left[U^2\right] + \lambda_R^2 \mathbf{E}\left[R^2\right]}{\lambda_R(T+\Delta)} \\
= \;& 0.
\end{aligned}$$

This last inequality is a consequence of the assumption $\mathbf{E}\left[R^2\right] < \infty$ and $\mathbf{E}\left[U^2\right] < \infty$; a common occurrence in applications. ■

Although very tempting, it is erroneous to conclude that either one of the hit or hit* rates are monotone in Δ for any given inter-request time distribution F_R. This monotonicity property can be deduced in some special cases (e.g., Poisson requests), and sufficient conditions for it are provided in the following lemma.

Lemma 8.3 *Given the TTL parameter T, the hit rate incurred by the fixed TTL is non-increasing in Δ on $[0,T]$ if the inter-request time distribution F_R satisfies the condition*

$$\mathbf{E}\left[R(\Delta_2) - R(\Delta_1)\right] \geq \mathbf{E}\left[R(T+\Delta_2) - R(T+\Delta_1)\right], \quad 0 \leq \Delta_1 < \Delta_2 \leq T, \tag{8.11}$$

On the other hand, if F_R and the inter-update time distribution F_U satisfy the more relaxed condition

$$\begin{aligned} &\mathbf{P}\left[U_e > \Delta_2\right]\left(\mathbf{E}\left[R(\Delta_2) - R(\Delta_1)\right]\right) \\ &\geq \mathbf{P}\left[U_e > T+\Delta_1\right]\left(\mathbf{E}\left[R(T+\Delta_2) - R(T+\Delta_1)\right]\right), \quad 0 \leq \Delta_1 < \Delta_2 \leq T, \end{aligned} \tag{8.12}$$

then the hit rate is non-increasing in Δ on $[0,T]$.*

Proof. The lemma only considers the delay Δ in the interval $[0,T]$, to comply with the ongoing assumption $T \geq \Delta$ that was used throughout the analysis of the hit and hit* rates.

Observe that if $\Delta_2 > \Delta_1$, then $\mathbf{E}\left[R(T+\Delta_2)\right] \geq \mathbf{E}\left[R(T+\Delta_1)\right]$, and the sufficient condition (8.11) immediately ensures $H(T,\Delta_1) - H(T,\Delta_2) \geq 0$. To derive the condition on the hit* rate, we refer to the expression for $H^\star(T,\Delta)$ in (8.7) and define

$$K(T,\Delta) = \mathbf{E}\left[R(\min(T+\Delta, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right].$$

The desired monotonicity $H^\star(T,\Delta_1) - H^\star(T,\Delta_2) \geq 0$ is satisfied if $K(T,\Delta_1) - K(T,\Delta_2) \geq 0$ for $T \geq \Delta_2 > \Delta_1 \geq 0$. A closer examination of this requirement yields

$$K(T,\Delta_1) - K(T,\Delta_2)$$

$$
\begin{aligned}
&= \mathbf{E}\left[\mathbf{1}\left[\Delta_1 < U_e \leq \Delta_2\right]\left(R(U_e) - R(\Delta_1)\right)\right] \\
&\quad + \mathbf{E}\left[\mathbf{1}\left[\Delta_2 < U_e \leq T + \Delta_1\right]\left(R(\Delta_2) - R(\Delta_1)\right)\right] \\
&\quad + \mathbf{E}\left[\mathbf{1}\left[T + \Delta_1 < U_e \leq T + \Delta_2\right]\left(R(\Delta_2) - R(\Delta_1) + R(T + \Delta_1) - R(U_e)\right)\right] \\
&\quad + \mathbf{E}\left[\mathbf{1}\left[T + \Delta_2 < U_e\right]\left(R(\Delta_2) - R(\Delta_1) + R(T + \Delta_1) - R(T + \Delta_2)\right)\right] \\
&\geq \mathbf{E}\left[\mathbf{1}\left[\Delta_2 < U_e\right]\left(R(\Delta_2) - R(\Delta_1)\right)\right] \\
&\quad - \mathbf{E}\left[\mathbf{1}\left[T + \Delta_1 < U_e\right]\left(R(T + \Delta_2) - R(T + \Delta_1)\right)\right] \\
&\geq 0.
\end{aligned}
$$

Under the assumption that U_e is independent of the inter-request time rv R, this last inequality is equivalent to the monotonicity condition (8.12). ■

It is now left to explore the asymptotic behavior of the hit and hit* rates as Δ becomes very large. Before doing so, it is important to point out that the sufficient conditions listed in Lemma 8.3 are clearly satisfied when the inter-request time is exponentially distributed (i.e., Poisson requests).

Lemma 8.4 *For each $T > 0$ we have the asymptotics*

$$\lim_{\Delta \to \infty} H^{\star}(T + \Delta, \Delta) = 0,$$

and

$$\lim_{\Delta \to \infty} H(T + \Delta, \Delta) \geq \frac{1}{2}.$$

The proof of Lemma 8.4 follows similar arguments as those used in the proof of Lemma 8.2, and is therefore omitted.

8.5 Evaluating the hit and hit* rates

8.5.1 Exponential inter-request times

An important special case arises when requests occur according to a Poisson process, in which case quite explicit expressions are available, namely

$$H_{Pois}(T,\Delta) = \frac{\lambda_R T}{\lambda_R(T+\Delta)+1}$$

and

$$H^{\star}_{Pois}(T,\Delta) = \frac{\lambda_R}{\lambda_R(T+\Delta)+1} \cdot \mathbf{E}\left[\min\left(\Delta+T,U_e\right) - \min\left(\Delta,U_e\right)\right].$$

While a simple closed form expression is available for the hit rate, the hit* rate can be evaluated in principle once the distribution F_U is specified. For instance, consider the case when updates occur periodically every Δ^u time units. It is plain that $H^{\star}(T,\Delta) = 0$ whenever $\Delta^u \leq \Delta$ (as would be expected). However if $\Delta < \Delta^u$, as we recall that U_e is uniformly distributed on the interval $[0,\Delta^u]$, simple calculations show that

$$H^{\star}_{Pois}(T,\Delta) = \frac{\lambda_R}{2\Delta^u\left(\lambda_R(T+\Delta)+1\right)}\left(\left(\Delta^u-\Delta\right)^2 - \left(\left(\Delta^u-\Delta-T\right)^+\right)^2\right).$$

This expression allows for a comparison between the hit and hit* rates incurred by the fixed TTL for the practical systems examined in [67], as illustrated in Figure 8.2 for several values of T and Δ.

8.5.2 Distribution-free results

Bounds presented in Section 7.5.1 for the renewal function associated with any distribution F_R can now be used to get rough estimates of the hit rate under the fixed TTL.

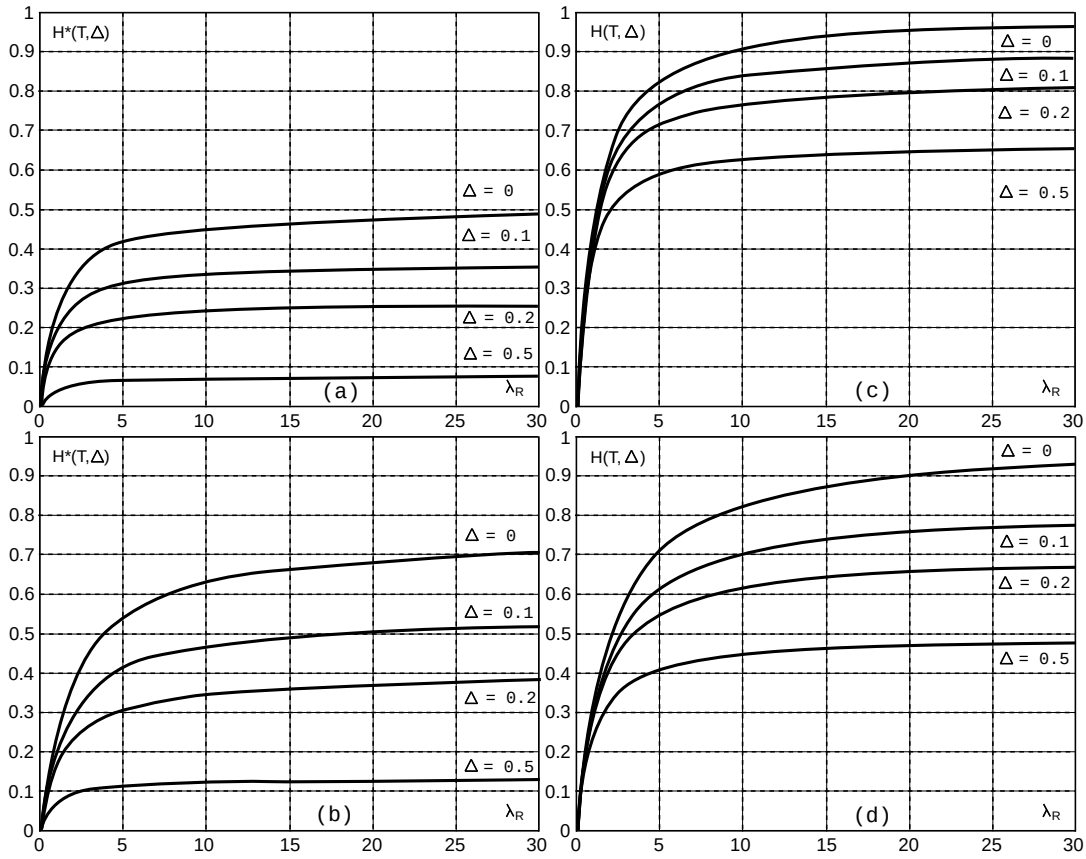

Figure 8.2: Hit and hit* rates for Poisson requests and fixed inter-updates $\Delta^u = 1$. (a) Hit* rate, $T = 1$; (b) Hit* rate, $T = 0.5$; (c) Hit rate, $T = 1$; (d) Hit rate, $T = 0.5$.

Explicitly, by replacing the renewal function terms in the expression for $H(T,\Delta)$ with the applicable bounds (7.6) and (7.7), we find

$$\frac{\lambda_R T + \mathbf{P}\left[R_e > T+\Delta\right] - \lambda_R^2 \mathbf{E}\left[R^2\right]}{\lambda_R(T+\Delta) + \lambda_R^2 \mathbf{E}\left[R^2\right]} \leq H(T,\Delta) \leq \frac{\lambda_R T + \lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > \Delta\right]}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]}.$$

These bounds are obviously not effective when $\lambda_R^2 \mathbf{E}\left[R^2\right]$ is large, for then it is possible for the upper bound to be greater than one and the lower bound to be negative.

Similar bounding arguments can be invoked for the hit* rate, in the process yielding the upper bound

$$\begin{aligned} H^\star(T,\Delta) \quad \leq \quad & \frac{\lambda_R(T+\Delta)+1}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]} H^\star_{Pois}(T,\Delta) \\ & + \frac{\mathbf{P}\left[U_e > \Delta\right]\left(\lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > \Delta\right]\right)}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]} \end{aligned}$$

and the associated lower bound

$$\begin{aligned} H^\star(T,\Delta) \quad \geq \quad & \frac{\lambda_R(T+\Delta)+1}{\lambda_R(T+\Delta) + \lambda_R^2 \mathbf{E}\left[R^2\right]} H^\star_{Pois}(T,\Delta) \\ & - \frac{\mathbf{P}\left[U_e > \Delta\right]\left(\lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > T+\Delta\right]\right)}{\lambda_R(T+\Delta) + \lambda_R^2 \mathbf{E}\left[R^2\right]}, \end{aligned}$$

which clearly suffer from the same deficiencies emphasized before.

8.5.3 NBUE and NWUE Requests

We now demonstrate how the NBUE and NWUE bounds described in Section 7.5.2 can be used to better evaluate the hit and hit* rates, by tightening the bounds obtained in the last section. For F_R NBUE, the hit rate satisfies

$$\frac{\lambda_R T - \mathbf{P}\left[R_e \leq T+\Delta\right]}{\lambda_R(\Delta+T)+1} \leq H(T,\Delta) \leq \frac{\lambda_R T + \mathbf{P}\left[R_e \leq T+\Delta\right]}{\lambda_R(\Delta+T) + \mathbf{P}\left[R_t > T+\Delta\right]},$$

and the bounds for the hit* rate are given by

$$H^\star(T,\Delta) \leq \frac{(\lambda_R(T+\Delta)+1) H^\star_{Pois}(T,\Delta)}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]} + \frac{\mathbf{P}\left[U_e > \Delta\right]\mathbf{P}\left[R_e \leq \Delta\right]}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]} \quad (8.13)$$

and

$$H^\star(T,\Delta) \geq H^\star_{Pois}(T,\Delta) - \frac{\mathbf{P}\left[U_e > \Delta\right]\mathbf{P}\left[R_e \leq T+\Delta\right]}{\lambda_R(T+\Delta)+1}. \tag{8.14}$$

Similarly, with F_R NWUE we have

$$\frac{\lambda_R T - \lambda_R^2\mathbf{E}\left[R^2\right]+1}{\lambda_R(\Delta+T)+\lambda_R^2\mathbf{E}\left[R^2\right]} \leq H(T,\Delta) \leq \frac{\lambda_R T + \lambda_R^2\mathbf{E}\left[R^2\right]-1}{\lambda_R(\Delta+T)+1},$$

together with

$$H^\star(T,\Delta) \leq H^\star_{Pois}(T,\Delta) + \frac{\mathbf{P}\left[U_e > \Delta\right](\lambda_R^2\mathbf{E}\left[R^2\right]-1)}{\lambda_R(T+\Delta)+1}$$

and

$$H^\star(T,\Delta) \geq \frac{(\lambda_R(T+\Delta)+1)H^\star_{Pois}(T,\Delta)}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}\left[R^2\right]} - \frac{\mathbf{P}\left[U_e > \Delta\right](\lambda_R^2\mathbf{E}\left[R^2\right]-1)}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}\left[R^2\right]}.$$

Although good bounds are already established for NBUE (and thus IFR) distributions, the NWUE bounds are still not satisfactory, for the same reasons outlined earlier. To further tune these bounds, we make use of Marshall's linear bounds, namely (7.16) and (7.17), in the expressions for the hit rate, and obtain

$$\frac{\lambda_R T - (b_u - b_\ell)}{1+\lambda_R(T+\Delta)+b_u} \leq H(T,\Delta) \leq \frac{\lambda_R T + (b_u-b_\ell)}{1+\lambda_R(T+\Delta)+b_\ell}.$$

Applying the linear bounds on the hit* rate yields

$$H^\star(T,\Delta) \geq H^\star_{Pois}(T,\Delta)\frac{1+\lambda_R(T+\Delta)}{1+\lambda_R(T+\Delta)+b_\ell} + \frac{\mathbf{P}\left[U_e > \Delta\right](b_u-b_\ell)}{1+\lambda_R(T+\Delta)+b_\ell} \tag{8.15}$$

and

$$H^\star(T,\Delta) \leq H^\star_{Pois}(T,\Delta)\frac{1+\lambda_R(T+\Delta)}{1+\lambda_R(T+\Delta)+b_u} - \frac{\mathbf{P}\left[U_e > \Delta\right](b_u-b_\ell)}{1+\lambda_R(T+\Delta)+b_u}. \tag{8.16}$$

The refined bounds are therefore achieved as we recall that $b_\ell = 0$ (equivalently, $b_u = 0$) whenever F_R is NWUE (equivalently, NBUE), respectively.[4]

[4]See (7.18) and (7.19), as well as previously made comments in Section 7.5.3.

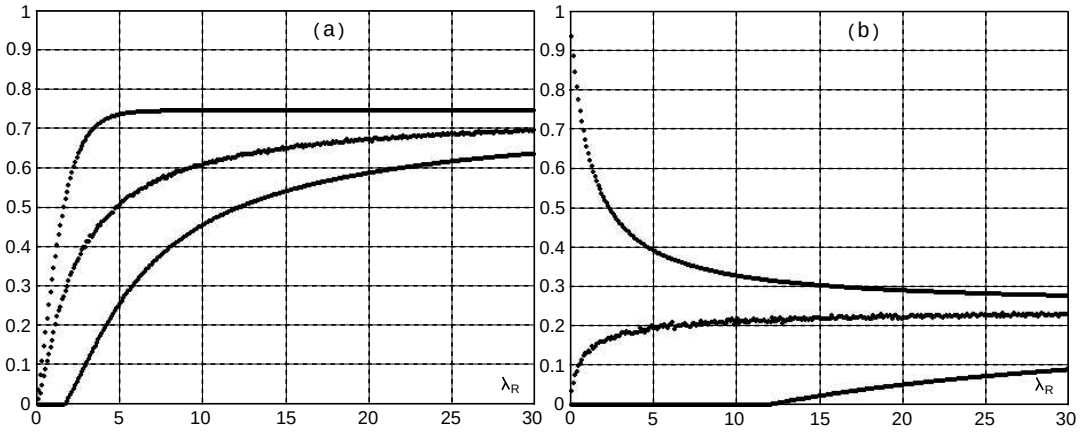

Figure 8.3: Upper bound, lower bound, and simulated hit* rate with fixed inter-updates $\Delta^u = 1$ and $T = 0.5$: (a) Weibull inter-request times, $\Delta = 0$, $\alpha = 1.3$; (b) Pareto inter-request times, $\Delta = 1/3$, $\alpha = 2.1$.

The use of these bounds is exhibited in Figure 8.3, when inter-request times are modeled by the Weibull and Pareto distributions under fixed inter-update times. In this figure, while we used the NBUE bounds (8.13) and (8.14) for the Weibull distribution, in the case of the Pareto distribution, we have relied on the results (8.15) and (8.16) with $b_\ell = 0$ and $b_u = 1$, as already concluded in (7.22).

Chapter 9

The Perfect TTL Algorithm

The perfect TTL is a non-causal consistency algorithm where the server knows at any given time when the current version of D will be updated. Each time the server receives a request for D, say at time t, it returns the current version together with the residual lifetime value $U_e(t)$ as its TTL. The downloaded document is placed in the cache at time $t + \Delta$, and is considered cache-fresh until $t + U_e(t)$.

In general, Web servers are *not* capable of predicting the next master modification time, therefore the implementation of the perfect TTL is not feasible in many caching systems in practice. In spite of this fact, the perfect TTL algorithm can be easily deployed in applications where the rollout schedule for the Web server is known in advance, a common practice in commercial servers with frequently updated pages [67].

The analysis of the perfect TTL is not only important for evaluating the consistency performance that can be attained by the systems described in [67], but it also contributes to understanding the limitations of other consistency protocols: Since the

perfect TTL is the single strong consistency algorithm in the class of TTL protocols,[1] we expect this algorithm to produce the highest hit* rate among all TTL algorithms.

9.1 Rules of engagement

When $\Delta = 0$, a document downloaded from the server in response to a cache-miss at time T_n^r for some $n = 0, 1, \ldots$ is instantenously placed in the cache, and the server-fresh object is sent to the user. All requests that arrive during the time interval $(T_n^r, T_n^r + U_e(T_n^r))$ incur a hit and are served by the cache with a server-fresh document. Consequently, each hit request produces a hit*, hence the perfect TTL is a strong consistency algorithm. In addition, since the server-freshness of D expires at the precise moment of TTL expiration at time $T_n^r + U_e(T_n^r)$, the server-freshness and cache-freshness tracking processes are identical for all $t \geq 0$, and the hit and hit* rates are equal when Δ is indeed zero.

Now pick an arbitrary $\Delta \geq 0$. The first miss request for D arrives to the cache at $T_0^r = 0$, the cache places the returned version at $T_0^r + \Delta$, and forwards the downloaded item to the user. If the master is updated prior to placement, i.e., $U_e(T_0^r) \leq \Delta$, then the cache immediately marks D as invalid. On the other hand, if $U_e(T_0^r) > \Delta$, then any subsequent request for D at the cache in the time interval $(T_0^r + \Delta, T_0^r + U_e(T_0^r))$ is served directly by the cache with a server-fresh copy.

As before, if a request at time T_n^r for some $n = 1, 2, \ldots$ is the first one presented after the TTL has expired, then it is forwarded to the server. Requests presented

[1] See additional arguments below.

during the time interval $(T_n^r, T_n^r + \Delta)$ that find an invalid cached copy are forwarded to the server as well. If a document returned from the server finds a zero TTL value at the cache, then the TTL counter is reinitialized upon the placement of the newly retrieved document.[2]

9.2 Quality of data under the perfect TTL

For the consistency algorithms considered in this dissertation, the cache-freshness and server-freshness tracking processes, $\{L_\Delta(t),\ t \geq 0\}$ and $\{L_\Delta^\star(t),\ t \geq 0\}$, respectively, are related through $L_\Delta(t) \geq L_\Delta^\star(t)$, for all $t \geq 0$ and $\Delta \geq 0$. These two processes are obviously not equal in general, as previously demonstrated in the analysis of the fixed TTL [8].

We have already established that if $\Delta = 0$, then under the perfect TTL the cache and server freshness tracking processes coincide. This fact remains true even when Δ is non-zero, as later explained in the hit* rate analysis in Section 9.4. The perfect TTL is therefore a special case where $L_\Delta(t) = L_\Delta^\star(t)$ for all $t \geq 0$ and $\Delta \geq 0$, whence the hit and hit* rates are always equal. As a result, the QoD given by the ratio (7.4) achieves its maximal value of one.

In spite of maximizing the QoD measure, it is erroneous to conclude that the perfect TTL achieves a higher hit* rate than all other (including non-TTL) cache

[2]These operational constraints coincide with the rules of the fixed TTL algorithm discussed in Section 8.1, where requests arriving after a cache miss at time T_n^r in the interval $(T_n^r, T_n^r + \Delta)$ for $n = 0, 1, \ldots$, are also forwarded to the server. However, under the assumption $T \geq \Delta$ for the fixed TTL, miss requests presented in this time interval never find a zero TTL upon their return to the cache, hence do not affect the TTL of the cached copy.

consistency protocols: Consider the replication server-invalidation algorithm. Under this algorithm, the server publishes master changes to the cache upon each update. As a consequence, documents stored at the cache are always cache-fresh, therefore user requests are always served by the cache. When $\Delta = 0$, master changes are replicated to the cache *instantenously*, and every request is served with a cache-fresh* document. In other words, when $\Delta = 0$, the hit and hit* rates attained by the replication algorithm are both equal to one, whereas under the perfect TTL the hit* rate (9.3) is clearly less than one.

9.3 Zero delays

In the notation of the modeling framework, let $\{L(t),\ t \geq 0\}$ denote the cache (and server) freshness tracking process of D at the cache.[3] This process has right-continuous sample paths with left limits, and is defined by

$$L(t) = \left(L(T_n^r) - (t - T_n^r)\right)^+, \quad T_n^r \leq t < T_{n+1}^r$$

for each $n = 0, 1, \ldots$, with the update rule

$$L(T_n^r) = \begin{cases} U_e(T_n^r) & \text{if } L(T_n^r-) = 0 \\ \\ L(T_n^r-) & \text{if } L(T_n^r-) > 0. \end{cases} \tag{9.1}$$

The initial value of the process is taken to be $L(0-) = 0$ so that $L(0) = U_e(0) = U_1$ by (9.1). The TTL expires at time $t > 0$ if and only if $L(t-) = 0$, and therefore the n^{th} request at time T_n^r produces both a hit and hit* if and only if $L(T_n^r-) > 0$.

[3]Since $\{L_\Delta(t),\ t \geq 0\} = \{L_\Delta^\star(t),\ t \geq 0\}$ for $\Delta \geq 0$ from previous arguments, we only refer to the process $\{L_\Delta(t),\ t \geq 0\}$ throughout this chapter. Furthermore, in this section $\Delta = 0$ and is omitted from the notation.

The tracking process $\{L(t),\ t \geq 0\}$ is related to the residual lifetime process $\{U_e(t),\ t \geq 0\}$, but is *not* identical to it. To appreciate this fact, introduce the sequence $\{\eta_m, m = 0, 1, \ldots\}$ of $\mathbb{N}$-valued rvs through

$$\eta_m = \inf\{n = 0, 1, \ldots : T^u_m \leq T^r_n\}. \tag{9.2}$$

As the rv η_m identifies the first request made after the m^{th} update, $m = 0, 1, \ldots$, the rv $T^r_{\eta_m}$ represents the first request epoch after that update taking place at T^u_m. It is easy to check that the two processes coincide everywhere except in each interval $[T^u_m, T^r_{\eta_m})$ where the tracking process vanishes. In particular, $L(T^r_{\eta_m}-) = 0$ and the request occuring at time $T^r_{\eta_m}$ incurs a miss. However, all the requests made in the interval $(T^r_{\eta_m}, T^r_{\eta_m} + U_e(T^r_{\eta_m}))$ result in a hit. This fact allows for an equivalent definition for the TTL tracking process, namely

$$L(t) = \left(L(T^r_{\eta_m}) - (t - T^r_{\eta_m})\right)^+$$

whenever $T^r_{\eta_m} \leq t < T^r_{\eta_m+1}$ for each $m = 0, 1, \ldots$, with the update rule

$$L(T^r_{\eta_m}) = U_e(T^r_{\eta_m}).$$

We use the initial condition $L(0-) = 0$ so that $L(T^r_{\eta_0}) = L(0) = U_1$.

The corresponding hit rate H (and thus the hit* rate $H^\star$ as well) is defined at (7.2) with $\{L(t),\ t \geq 0\}$.

Proposition 9.1 *If the point processes $\{T^r_n,\ n = 0, 1, \ldots\}$ and $\{T^u_m,\ m = 0, 1, \ldots\}$ are renewal processes, the common value of the hit and hit* rates is given by*

$$H = H^\star = 1 - \frac{\mathbf{P}\left[R_e < U\right]}{\lambda_R \mathbf{E}\left[U-\right]}. \tag{9.3}$$

The proof of this proposition is omitted as it is available by assigning $\Delta = 0$ in Proposition 9.2.

In the expression (9.3) the stationary forward recurrence time R_e is taken to be independent of the counting process $\{U(t),\ t \geq 0\}$, and its distribution is given by (7.5). An alternative expression for H can therefore be written as

$$H = H^{\star} = 1 - \lambda_U \int_0^{\infty} \mathbf{P}\left[R > z\right] \mathbf{P}\left[U > z\right]\, dz \tag{9.4}$$

and is a direct consequence of (9.6) with $\Delta = 0$.

9.4 Non-zero delays

A request presented at time $T^r_{\eta_m}$ finds an invalid cached document and incurs a miss. The downloaded copy is placed in the cache at $T^r_{\eta_m} + \Delta$ and expires at $T^r_{\eta_m} + U_e(T^r_{\eta_m})$, possibly prior to placement. All requests that arrive in the interval $(T^r_{\eta_m}, T^r_{\eta_m} + \Delta)$ also find an expired cached copy and are forwarded to the server as well, to increase the consistency of D at the cache: If $T^r_{\eta_m} < T^u_{m+1} < T^r_{\eta_m} + \Delta$ for some $m = 0, 1, \ldots,$ then the document placed at time $T^r_{\eta_m} + \Delta$ is immediately stale, and requests made during $[T^u_{m+1}, T^r_{\eta_m} + \Delta)$ retrieve the latest version of D and replace the stale replica. These dynamics are illustrated in Figure 9.1.

Under these rules of engagement, the validator tracking process $\{L_{\Delta}(t),\ t \geq 0\}$ can be defined as

$$L_{\Delta}(t) = \left(L_{\Delta}(T^r_n + \Delta) - (t - (T^r_n + \Delta))\right)^+$$

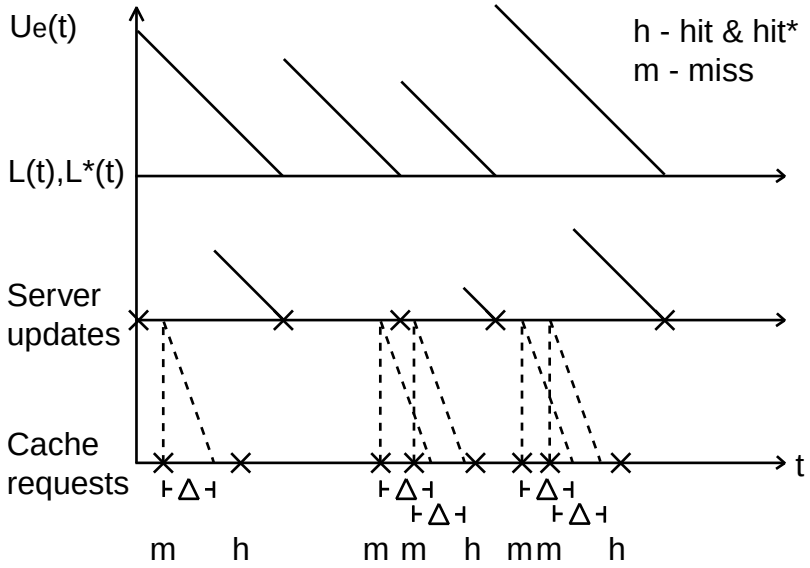

Figure 9.1: Time line diagram of requests, updates, and the tracking process of the perfect TTL algorithm for $\Delta > 0$.

whenever $T_n^r + \Delta \leq t < T_{n+1}^r + \Delta$, and adopts the update rule

$$L_\Delta(T_n^r + \Delta) = \begin{cases} (U_e(T_n^r) - \Delta)^+ & \text{if } L_\Delta((T_n^r + \Delta)-) = 0 \\ \\ L_\Delta((T_n^r + \Delta)-) & \text{if } L_\Delta((T_n^r + \Delta)-) > 0\ . \end{cases}$$

This tracking process can again be related to the residual lifetime process $\{U_e(t),\ t \geq 0\}$ as follows: If $T_{\eta_m}^r + \Delta < T_{m+1}^u$ (equivalently, $U_e(T_{\eta_m}^r) - \Delta > 0$) for some $m = 0, 1, \ldots$, then the two processes coincide on each interval $T_{\eta_m}^r + \Delta \leq t < T_{m+1}^u$, whereas $L_\Delta(t) = 0$ in each interval $[T_m^u, T_{\eta_m}^r + \Delta)$. In all other cases where $T_{\eta_m}^r + \Delta \geq T_{m+1}^u$, the tracking process is zero in the interval $[T_m^u, T_{m+1}^u]$. In other words, the two processes coincide wherever the tracking process is non-zero. These observations lead to the alternative definition of the tracking process given by

$$L_\Delta(t) = (L_\Delta(T_{\eta_m}^r + \Delta) - (t - (T_{\eta_m}^r + \Delta)))^+$$

on each time interval $T^r_{\eta_m} + \Delta \leq t < T^r_{\eta_{m+1}} + \Delta,\ m = 0, 1, \ldots$, updated by the rule

$$L_\Delta(T^r_{\eta_m} + \Delta) = (U_e(T^r_{\eta_m}) - \Delta)^+,$$

with the initial value $L_\Delta(t) = 0$ at $t < \Delta$. Moreover, by the fact that $L_\Delta(t) = U_e(t)$ on the set $\{t \geq 0 \ :\ L_\Delta(t) > 0\}$ it is plain that each hit request incurs a hit*, whence the processes $\{L_\Delta(t),\ t \geq 0\}$ and $\{L^\star_\Delta(t),\ t \geq 0\}$ coincide, as in the case of zero delays.

The hit rate $H(\Delta)$ is given in (7.2) with the tracking process $\{L_\Delta(t),\ t \geq 0\}$, and an expression for it is available in the following proposition.

Proposition 9.2 *With the assumptions of Proposition 9.1, the common value of the hit and hit* rates is given by*

$$\begin{aligned} H(\Delta) = H^\star(\Delta) &= 1 - \frac{\mathbf{E}\left[R(\min(R_e + \Delta, U))\right]}{\lambda_R \mathbf{E}\left[U-\right]} \\ &= 1 - \frac{\mathbf{E}\left[\mathbf{1}\left[R_e + \Delta > U\right] R(U)\right]}{\lambda_R \mathbf{E}\left[U-\right]} \\ &\quad - \frac{\mathbf{P}\left[R_e + \Delta \leq U\right] (1 + \mathbf{E}\left[R(\Delta)\right])}{\lambda_R \mathbf{E}\left[U-\right]}. \end{aligned} \tag{9.5}$$

As expected, when $\Delta = 0$ this result specializes to the one stated in Proposition 9.1. A proof of Proposition 9.2 is available in Appendix C.

Here again, the rv R_e in (9.5) is taken to be independent of the counting process $\{U(t),\ t \geq 0\}$. This fact allows us to rewrite the hit rate as

$$\begin{aligned} H(\Delta) &= 1 - \lambda_U \int_0^\infty \int_{(u-\Delta)^+}^u (1 + \mathbf{E}\left[R(u - r)\right]) \mathbf{P}\left[R > r\right]\, dr\, dF_U(u) \\ &\quad - (1 + \mathbf{E}\left[R(\Delta)\right]) \cdot \lambda_U \int_0^\infty \mathbf{P}\left[U > r + \Delta\right] \mathbf{P}\left[R > r\right]\, dr, \end{aligned} \tag{9.6}$$

which reduces to (9.4) for $\Delta = 0$. A proof of the alternative expression (9.6) is also provided in Appendix C.

9.5 Bounds and properties

Bounds on the hit rate produced by the perfect TTL can be derived from the expression (9.5), for any inter-request time and inter-update time distributions. Explicitly, under the renewal assumption on the point process of requests, we have[4]

$$\frac{\mathbf{E}\left[\mathbf{1}\left[U < R_e + \Delta\right] R(U)\right]}{\lambda_R \mathbf{E}\left[U-\right]} \leq \frac{\mathbf{P}\left[U < R_e + \Delta\right] (1 + \mathbf{E}\left[R(\Delta)\right])}{\lambda_R \mathbf{E}\left[U-\right]}, \tag{9.7}$$

and we obtain the lower bound

$$H(\Delta) \geq 1 - \frac{1 + \mathbf{E}\left[R(\Delta)\right]}{\lambda_R \mathbf{E}\left[U-\right]}. \tag{9.8}$$

An upper bound can be developed by removing the negative term in the left hand side of the expression (9.7) for the hit rate. This yields

$$H(\Delta) \leq 1 - \frac{\mathbf{P}\left[R_e + \Delta \leq U\right] (1 + \mathbf{E}\left[R(\Delta)\right])}{\lambda_R \mathbf{E}\left[U-\right]}, \tag{9.9}$$

and equality is achieved whenever $\Delta = 0$.

The single parameter that affects the consistency performance of the perfect TTL is the download delay Δ. Under the operational rules of this algorithm, we (intuitively) anticipate that increased communication delays would degrade the cache consistency outcome. This is indeed the case, as concluded in the following proposition.

Proposition 9.3 *The hit rate produced by the perfect TTL algorithm is a non-increasing function of* Δ*, with* $\lim_{\Delta \to \infty} H(\Delta) = 0$.

Proof. Take $\Delta_2 \geq \Delta_1 \geq 0$. From the expression (9.6) for the hit rate, it is simple

[4] A proof for (9.7) is provided in Appendix C.

matter to check that

$$\begin{aligned}
&\frac{1}{\lambda_U}(H(\Delta_1) - H(\Delta_2)) \\
&= \int_0^\infty \int_{\Delta_1}^{\Delta_2} (1 + \mathbf{E}\left[R(s)\right])\mathbf{P}\left[R > u - s\right]\, ds\, dF_U(u) \\
&\quad - (1 + \mathbf{E}\left[R(\Delta_1)\right]) \int_0^\infty \int_{(u-\Delta_2)^+}^{(u-\Delta_1)^+} \mathbf{P}\left[R > r\right]\, dr\, dF_U(u) \\
&\quad + (\mathbf{E}\left[R(\Delta_2)\right] - \mathbf{E}\left[R(\Delta_1)\right]) \int_0^\infty \mathbf{P}\left[U > r + \Delta_2\right] \mathbf{P}\left[R > r\right]\, dr.
\end{aligned}$$

The desired result $H(\Delta_1) - H(\Delta_2) \geq 0$ follows by replacing $\mathbf{E}\left[R(s)\right]$ with $\mathbf{E}\left[R(\Delta_1)\right]$ (clearly not greater than $\mathbf{E}\left[R(s)\right]$ for s in the interval $[\Delta_1, \Delta_2]$), cancelling the first two terms in this last expression, and recalling that $\mathbf{E}\left[R(\Delta_2)\right] - \mathbf{E}\left[R(\Delta_1)\right] \geq 0$.

The asymptotic behavior of $H(\Delta)$ can be derived from (9.6) as well: First, the hit rate can be rewritten as

$$\begin{aligned}
H(\Delta) &= 1 - \frac{\lambda_U}{\lambda_R} \int_0^\infty \int_{(u-\Delta)^+}^{u} (1 + \mathbf{E}\left[R(u - r)\right])\, dF_{R_e}(r)\, dF_U(u) \\
&\quad - (1 + \mathbf{E}\left[R(\Delta)\right]) \cdot \lambda_U \int_0^\infty \mathbf{P}\left[U > r + \Delta\right] \mathbf{P}\left[R > r\right]\, dr.
\end{aligned}$$

Letting Δ go to infinity, we find

$$\lim_{\Delta\to\infty} \frac{\lambda_U}{\lambda_R} \int_0^\infty \int_{(u-\Delta)^+}^{u} (1 + \mathbf{E}\left[R(u - r)\right])\, dF_{R_e}(r)\, dF_U(u) = \lambda_U \int_0^\infty u \cdot\, dF_U(u) = 1.$$

This result is a simple consequence of the Renewal Equation [69] according to which

$$\lambda_R \cdot t = \int_0^t (1 + \mathbf{E}\left[R(t - r)\right])\, dF_{R_e}(r), \quad t \geq 0.$$

Then, by applying the Basic Renewal Theorem (8.10) and recalling $\lim_{u\to\infty} u\mathbf{P}\left[U > u\right] = 0$ (implied by the integrability of F_U), we have

$$\lim_{\Delta\to\infty} (1 + \mathbf{E}\left[R(\Delta)\right]) \cdot \lambda_U \int_0^\infty \mathbf{P}\left[U > r + \Delta\right] \mathbf{P}\left[R > r\right]\, dr$$

$$
\begin{aligned}
&= \lim_{\Delta\to\infty} \lambda_R\lambda_U \int_0^\infty \Delta\mathbf{P}\left[U > r+\Delta\right]\mathbf{P}\left[R > r\right]\, dr \\
&\leq \lim_{\Delta\to\infty} \lambda_R\lambda_U \int_0^\infty \Delta\mathbf{P}\left[U > \Delta\right]\mathbf{P}\left[R > r\right]\, dr \\
&= 0,
\end{aligned}
$$

which completes the proof of the proposition. ■

9.6 Evaluation of the hit and hit* rates

9.6.1 Poisson requests

When inter-request times are exponentially distributed with rate λ_R, the hit rate of the perfect TTL can be written as

$$
\begin{aligned}
H_{Pois}(\Delta) &= \lambda_U \mathbf{E}\left[(U-\Delta)^+ - \lambda_R^{-1}\left(1 - e^{-\lambda_R(U-\Delta)^+}\right)\right] \\
&= \frac{\lambda_U}{\lambda_R}\mathbf{E}\left[\varphi(\lambda_R(U-\Delta)^+)\right], \qquad (9.10)
\end{aligned}
$$

with mapping $\varphi : \mathbb{R}_+ \to \mathbb{R}_+$ given by

$$
\varphi(x) = x - (1 - e^{-x}), \quad x \geq 0.
$$

This mapping is monotone increasing, thereby confirming the validity of Proposition 9.3 for the special case of Poisson requests.

A closed form expression for the hit rate can be found in some scenarios. For instance, under periodic updates where D is altered every Δ^u time units, we find that

$$
H_{Pois}(\Delta) = 0, \quad \Delta^u \leq \Delta,
$$

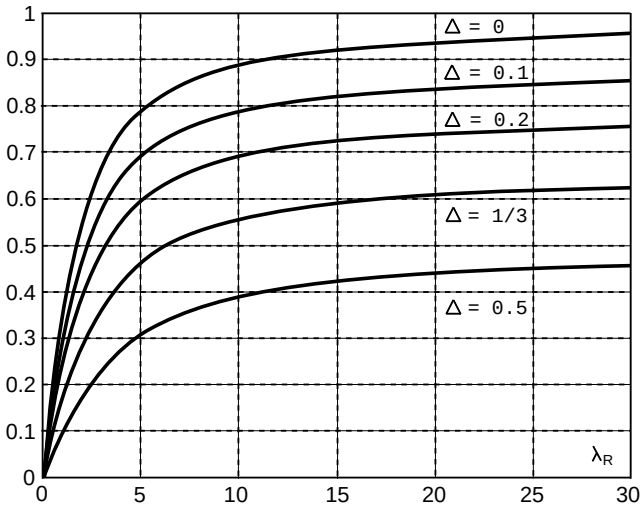

Figure 9.2: The hit and hit* rate of the perfect TTL with Poisson requests and fixed inter-updates $\Delta^u = 1$, for several values of Δ.

while with $\Delta^u > \Delta$ the expression (9.10) becomes

$$H_{Pois}(\Delta) = \frac{\Delta^u - \Delta}{\Delta^u} - \frac{1}{\lambda_R \Delta^u} \left(1 - e^{-\lambda_R(\Delta^u - \Delta)}\right).$$

This hit rate is presented in Figure 9.2 for several values of the delay Δ, where we can now visualize the degradation of the hit rate as Δ increases.

9.6.2 Distribution-free results

The bounds presented in (9.8) and (9.9) are not very useful in general applications, as the value of the renewal function $\mathbf{E}\left[R(t)\right]$ may not be known in closed form for every $t \geq 0$. To circumvent this difficulty, additional bounds can be derived by utilizing the bounds on the renewal function discussed in Chapter 7.

A new upper bound can be developed by replacing the term $\mathbf{E}\left[R(\Delta)\right]$ in the upper

bound expression (9.9) with the lower bound (7.7), so that

$$H(\Delta) \leq 1 - \frac{\lambda_U}{\lambda_R}\mathbf{P}\left[R_e + \Delta \leq U\right](\lambda_R\Delta + \mathbf{P}\left[R_e > \Delta\right]). \tag{9.11}$$

In this last term we have taken $\mathbf{E}\left[R(U-)\right] = \lambda_R\lambda_U^{-1}$, by virtue of the fact that the rv U is independent of the stationary point process of user requests.

A lower bound on the hit rate can be derived in a similar manner, by assigning (7.6) in the lower bound (9.8). This action results in a smaller lower bound than the one reported in (9.8), which is given by the expression

$$H(\Delta) \geq 1 - \frac{\lambda_U}{\lambda_R}\left(\lambda_R\Delta + \lambda_R^2\mathbf{E}\left[R^2\right]\right). \tag{9.12}$$

The new lower bound is inefficient when $\lambda_R^2\mathbf{E}\left[R^2\right]$ is large, and therefore better bounds are still required.

To conclude this section, note that additional bounds can be obtained by using the Marshall bounds. Replacing the term $\mathbf{E}\left[R(\Delta)\right]$ with the corresponding lower bound (7.17) in the upper bound expression (9.9) yields

$$H(\Delta) \leq 1 - \frac{\lambda_U}{\lambda_R}\mathbf{P}\left[R_e + \Delta \leq U\right](1 + \lambda_R\Delta + b_\ell). \tag{9.13}$$

Similar calculations provide us with the lower bound

$$H(\Delta) \geq 1 - \frac{\lambda_U}{\lambda_R}(1 + \lambda_R\Delta + b_u), \tag{9.14}$$

with b_ℓ and b_u given in (7.14) and (7.15), respectively.

9.6.3 NBUE and NWUE Requests

The hit rate under the perfect TTL algorithm can be better estimated by tightening the bounds listed thus far. This goal is achieved in this section when the distribution F_R is either NBUE or NWUE.

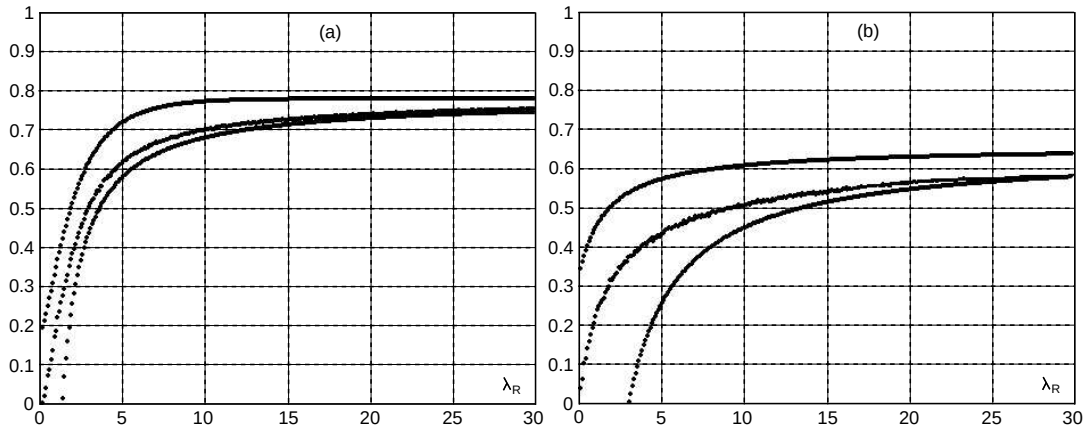

Figure 9.3: Upper bound, lower bound, and hit rate simulation results under fixed inter-update times Δ^u = 1: (a) Weibull inter-request times, $\Delta = 0.2$, $\alpha = 1.3$; (b) Pareto inter-request times, $\Delta = 1/3$, $\alpha = 2.1$.

If the inter-request time distribution is NBUE, the previous lower bounds can be revisited to write

$$H(\Delta) \geq 1 - \frac{\lambda_U}{\lambda_R}(1 + \lambda_R \Delta)$$

as we recall that $b_u = 0$ in (9.14) for all NBUE distributions. Combining this last inequality with (9.11) we find that the distance between the upper and lower bounds for NBUE inter-request time distributions is given by

$$\frac{\lambda_U}{\lambda_R}\left\{(1 + \lambda_R \Delta)\,\mathbf{P}\left[R_e + \Delta > U\right] + \mathbf{P}\left[R_e + \Delta \leq U\right]\mathbf{P}\left[R_e \leq \Delta\right]\right\},$$

so that the hit rate can already be well-estimated when Δ is small and $\lambda_R >> \lambda_U$, as is the case in most Web applications.

Similarly, if F_R is NWUE, then the bound

$$H(\Delta) \leq 1 - \frac{\lambda_U}{\lambda_R}\mathbf{P}\left[R_e + \Delta \leq U\right](1 + \lambda_R \Delta) \tag{9.15}$$

follows from (9.13), since now $b_l = 0$. Here, the lower bound (9.12) that suffers from the disadvantages mentioned earlier remains the best lower bound for general NWUE distributions. However, once the distribution F_R is given, improved bounds can still be derived. For example, when F_R is the Pareto distribution, then $b_u = 1$, and the lower bound (9.14) is given by

$$H(\Delta) \geq 1 - \frac{\lambda_U}{\lambda_R} (2 + \lambda_R \Delta) .$$

These bounds, together with the NBUE bounds, are illustrated in Figure 9.3 for the Pareto and Weibull inter-request time distributions, respectively.

Part III

Proofs

Appendix A

A Proof of Theorem 4.1

Theorem 4.1 is a direct consequence of the following fact:

Proposition A.1 *For each $T = 0, 1, \ldots,$ it holds that*

$$\begin{aligned} &\arg\min\{u \in S + r : \mathbf{E}\left[V_T(S + r - u, R)\right]\} \\ &= \arg\min\{u \in S + r : p(u)c(u)\} \end{aligned} \tag{A.1}$$

for any (S, r) in $\mathcal{X}$ with r not in S.

Equality (A.1) is understood to mean that

$$\mathbf{E}\left[V_T(S + r - v, R')\right] = \min_{u \in S+r} \mathbf{E}\left[V_T(S + r - u, R')\right] \tag{A.2}$$

holds with v given by

$$v = \arg\min\{j \in S + r : p(j)c(j)\}. \tag{A.3}$$

The proof proceeds by induction on $T = 0, 1, \ldots$.

The basic step - Fix (S, r) in $\mathcal{X}$ and note that

$$V_0(S, r) = \mathbf{1}\left[r \notin S\right] c(r). \tag{A.4}$$

Thus, for u in $S + r$ distinct from v (also in $S + r$ by virtue of its definition (A.3)), we have

$$\begin{aligned}
&\mathbf{E}\left[V_0(S + r - u, R')\right] \\
&= \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r - u\right] c(R')\right] \\
&= \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r\right] c(R')\right] - \mathbf{E}\left[\mathbf{1}\left[R' = u\right] c(R')\right]
\end{aligned}$$

with a similar expression for $\mathbf{E}\left[V_0(S + r - v, R')\right]$. Hence,

$$\begin{aligned}
&\mathbf{E}\left[V_0(S + r - u, R')\right] - \mathbf{E}\left[V_0(S + r - v, R')\right] \\
&= p(u)c(u) - p(v)c(v) \qquad \text{(A.5)}
\end{aligned}$$

and (A.1) does hold for $T = 0$.

The induction step - Assume (A.1) to hold for some $T = 0, 1, \ldots$. Fix (S, r) in $\mathcal{X}$ with r not in S. We need to show that for u in $S + r$, we have

$$\mathbf{E}\left[V_T(S + r - u, R') - V_T(S + r - v, R')\right] \geq 0 \qquad \text{(A.6)}$$

with v given by (A.3).

Fix u in $S + r$ and note that R'' is distributed like R' and is independent of it, by the IRM. Using the DPE (4.3) we can write

$$\begin{aligned}
&\mathbf{E}\left[V_{T+1}(S + r - u, R')\right] \\
&= \mathbf{P}\left[R' \in S + r - u\right] \mathbf{E}\left[V_T(S + r - u, R'')\right] \\
&\quad + \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r - u\right] c(R')\right] \qquad \text{(A.7)} \\
&\quad + \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r - u\right] \min_{u^\star \in S + r - u + R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right]\right].
\end{aligned}$$

Note that

$$\mathbf{P}\left[R' \in S + r - u\right] \mathbf{E}\left[V_T(S + r - u, R'')\right]$$

$$= \mathbf{P}\left[R' \in S + r - (u, v)\right] \mathbf{E}\left[V_T(S + r - u, R'')\right]$$
$$+ p(v)\mathbf{E}\left[V_T(S + r - u, R'')\right] \tag{A.8}$$

with v defined by (A.3), and that

$$\mathbf{E}\left[\mathbf{1}\left[R' \notin S + r - u\right] c(R')\right]$$
$$= \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r\right] c(R')\right] + p(u)c(u). \tag{A.9}$$

Finally,

$$\mathbf{E}\left[\mathbf{1}\left[R' \notin S + r - u\right] \min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right]\right]$$
$$= \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r\right] \min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right]\right]$$
$$+ p(u) \min_{u^\star \in S+r} \mathbf{E}\left[V_T(S + r - u^\star, R'')\right]. \tag{A.10}$$

Reporting (A.8), (A.9) and (A.10) into (A.7), we conclude that

$$\mathbf{E}\left[V_{T+1}(S + r - u, R')\right]$$
$$= \mathbf{P}\left[R' \in S + r - (u, v)\right] \mathbf{E}\left[V_T(S + r - u, R'')\right]$$
$$+ \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r\right] c(R')\right]$$
$$+ p(u)c(u) + p(u) \min_{u^\star \in S+r} \mathbf{E}\left[V_T(S + r - u^\star, R'')\right]$$
$$+ p(v)\mathbf{E}\left[V_T(S + r - u, R'')\right] \tag{A.11}$$
$$+ \mathbf{E}\left[\mathbf{1}\left[R' \notin S + r\right] \min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right]\right].$$

We can now write the corresponding expression (A.11) with u replaced by v, and the difference in (A.6) takes the form

$$\mathbf{E}\left[V_{T+1}(S + r - u, R') - V_{T+1}(S + r - v, R')\right]$$
$$= p(u)c(u) - p(v)c(v)$$
$$+ \mathbf{P}\left[R' \in S + r - (u, v)\right] \Delta_1 + p(u)\Delta_2 + p(v)\Delta_3 + \Delta_4 \tag{A.12}$$

with

$$\begin{aligned}\Delta_1 &:= \mathbf{E}\left[V_T(S+r-u,R'')\right] - \mathbf{E}\left[V_T(S+r-v,R'')\right] \\ \Delta_2 &:= \min_{u^\star \in S+r} \mathbf{E}\left[V_T(S+r-u^\star,R'')\right] - \mathbf{E}\left[V_T(S+r-v,R'')\right] \\ \Delta_3 &:= \mathbf{E}\left[V_T(S+r-u,R'')\right] - \min_{v^\star \in S+r} \mathbf{E}\left[V_T(S+r-v^\star,R'')\right] \quad \text{(A.13)}\end{aligned}$$

and

$$\begin{aligned}\Delta_4 \;=\; & \mathbf{E}\left[\mathbf{1}\left[R' \notin S+r\right] \min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S+r-u+R'-u^\star,R'')\right]\right] \\ & - \mathbf{E}\left[\mathbf{1}\left[R' \notin S+r\right] \min_{v^\star \in S+r-v+R'} \mathbf{E}\left[V_T(S+r-v+R'-v^\star,R'')\right]\right].\end{aligned}$$

Observe that $p(u)c(u) - p(v)c(v) \geq 0$ by the definition of v and that the condition $\Delta_1 \geq 0$, being equivalent to (A.1), holds true under the induction hypothesis. Again, by invoking the hypothesis we have that

$$\min_{u^\star \in S+r} \mathbf{E}\left[V_T(S+r-u^\star,R'')\right] = \mathbf{E}\left[V_T(S+r-v,R'')\right]$$

and therefore concludes $\Delta_2 = 0$. Similarly, for Δ_3 we get

$$\begin{aligned}&\mathbf{E}\left[V_T(S+r-u,R'')\right] - \min_{u^\star \in S+r} \mathbf{E}\left[V_T(S+r-u^\star,R'')\right] \\ &= \mathbf{E}\left[V_T(S+r-u,R'')\right] - \mathbf{E}\left[V_T(S+r-v,R'')\right] = \Delta_1,\end{aligned}$$

which is non-negative as argued earlier in the context of Δ_1.

Consequently, it is already the case that

$$\mathbf{E}\left[V_{T+1}(S+r-u,R') - V_{T+1}(S+r-v,R')\right] \geq 0 \quad \text{(A.14)}$$

directly from the induction hypothesis, and (A.3)-(A.6) will hold if we can show that $\Delta_4 \geq 0$. Inspection of Δ_4 reveals that $\Delta_4 \geq 0$ provided

$$\begin{aligned}&\min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S+r-u+R'-u^\star,R'')\right] \\ &\quad - \min_{v^\star \in S+r-v+R'} \mathbf{E}\left[V_T(S+r-v+R'-v^\star,R'')\right] \geq 0 \quad \text{(A.15)}\end{aligned}$$

whenever R' is not in $S + r$.

To establish (A.15) we find it useful to order the set of documents $\{1, \ldots, M\}$ according to their expected cost: For u and v in $\{1, \ldots, M\}$ we write $u < v$ (respectively, $v \leq u$) if $p(u)c(u) < p(v)c(v)$ (respectively, $p(v)c(v) \leq p(u)c(u)$), with equality $u = v$ if $p(u)c(u) = p(v)c(v)$. With this terminology we can now interpret v as the smallest element in $S+r$ according to this order. Two cases emerge, depending on whether $v < R'$ or $R' \leq v$:

Case 1 - Assume that $R' \leq v$ with R' not in $S + r$. Then, consider

$$\min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right] \tag{A.16}$$

and note that R' is not in $S + r - u$ and that R' is the smallest element in $S + r + R'$ (thus in $S + R' - u$ which contains it). By the induction hypothesis applied in the state $(S + r - u, R')$, the minimization (A.16) is achieved by the selection $u^\star = R'$, so that

$$\min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right] = \mathbf{E}\left[V_T(S + r - u, R'')\right].$$

The same argument shows that

$$\min_{v^\star \in S+r-v+R'} \mathbf{E}\left[V_T(S + r - v + R' - v^\star, R'')\right] = \mathbf{E}\left[V_T(S + r - v, R'')\right]$$

by applying the hypothesis on the state $(S + r - v, R')$. Combining these facts, we get

$$\begin{aligned}
&\min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right] \\
&\quad - \min_{v^\star \in S+r-v+R'} \mathbf{E}\left[V_T(S + r - v + R' - v^\star, R'')\right] \\
&= \mathbf{E}\left[V_T(S + r - u, R'')\right] - \mathbf{E}\left[V_T(S + r - v, R'')\right]
\end{aligned} \tag{A.17}$$

and (A.15) follows by invoking the induction hypothesis once more, this time in state (S, r).

Case 2 - Assume $v < R'$ with R' not in $S + r$. Then, going back to the expression (A.16), by virtue of the induction hypothesis applied to the state $(S + r - v, R')$, we find that

$$\begin{aligned}\min_{u^\star \in S+r-u+R'} \mathbf{E}\left[V_T(S + r - u + R' - u^\star, R'')\right] \\ = \mathbf{E}\left[V_T(S + r - u + R' - v, R'')\right]\end{aligned}$$

and (A.15) now follows by invoking the induction hypothesis once more, this time in state $(S + r - v, R')$, as we note that any element $u^\star$ in $S + r$ with $u^\star \neq v$ is necessarily in $S + r - v$, hence in $S + r - v + R'$. This completes the proof of Theorem 4.1.

Appendix B

Proofs of Propositions 8.3 and 8.4

In the next sections B.1 and B.2, we make use of the $\mathbb{N}$-valued rvs $\{\mu_k,\ k = 0, 1, \ldots\}$ defined recursively through (8.5). Note that $R(T^r_{\mu_k}) = \mu_k$ for all $k = 0, 1, \ldots$, and that under the renewal assumptions on the request process, the rvs $\{R(T^r_{\mu_{\ell+1}}) - R(T^r_{\mu_\ell}),\ \ell = 0, 1, \ldots\}$ are i.i.d., each distributed according to $R(T^r_{\mu_1})$. Therefore, by the Strong Law of Large Numbers we find

$$\lim_{k\to\infty} \frac{\mu_k}{k} = \lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R(T^r_{\mu_{\ell+1}}) - R(T^r_{\mu_\ell}) \right) = \mathbf{E}\left[R(T^r_{\mu_1}) \right] \quad a.s. \tag{B.1}$$

By the very definition of $T^r_{\mu_1}$, we have that

$$\mathbf{E}\left[R(T^r_{\mu_1}) \right] = 1 + \mathbf{E}\left[R((T+\Delta)-) \right], \tag{B.2}$$

as explained through arguments below.

B.1 A proof of Proposition 8.3

Since the rvs $\{\mu_k,\ k = 0, 1, \ldots\}$ monotonically exhaust $\mathbb{N}$ a.s., it is plain that

$$H(T, \Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T^r_n -) > 0 \right] \quad a.s.$$

In order to make use of this fact, fix $k = 0, 1, \ldots$ and consider the dynamics of the freshness tracking process on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$: With $T^r_{\mu_k} < t \leq T^r_{\mu_k} + \Delta$, we have $L_\Delta(t-) = 0$ since $L_\Delta(T^r_{\mu_k}) = 0$. The validator process is reinitialized at time $T^r_{\mu_k} + \Delta$ to the value $L_\Delta(T^r_{\mu_k} + \Delta) = T$, returning to zero with the expiration of the TTL at time $T^r_{\mu_k} + \Delta + T$, i.e., $L_\Delta((T^r_{\mu_k} + \Delta + T)-) = 0$.

Thus, the requests made at the cache in the interval $(T^r_{\mu_k}, T^r_{\mu_k} + \Delta]$ incur misses, while those occurring in $(T^r_{\mu_k} + \Delta, T^r_{\mu_k} + \Delta + T)$ are necessarily hits. Also, there is exactly one request made in the interval $[T^r_{\mu_k} + \Delta + T, T^r_{\mu_{k+1}}]$, and it is necessarily a miss. In summary, we see that there are exactly $1 + \left(R(T^r_{\mu_k} + \Delta) - R(T^r_{\mu_k})\right)$ misses in the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$. Therefore, for each $k = 1, 2, \ldots$, we get

$$\begin{aligned}\sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] &= \sum_{\ell=0}^{k-1} \sum_{\mu_\ell < n \leq \mu_{\ell+1}} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] \\ &= \sum_{\ell=0}^{k-1} \left(1 + \left(R(T^r_{\mu_\ell} + \Delta) - R(T^r_{\mu_\ell})\right)\right). \qquad \text{(B.3)}\end{aligned}$$

Again, under the renewal assumption on the request process, the rvs $\{R(T^r_{\mu_\ell} + \Delta) - R(T^r_{\mu_\ell}),\ \ell = 0, 1, \ldots\}$ are i.i.d. rvs, each distributed according to $R(\Delta)$, and the Strong Law of Large Numbers now gives

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R(T^r_{\mu_k} + \Delta) - R(T^r_{\mu_k})\right) = \mathbf{E}\left[R(\Delta)\right] \quad a.s. \qquad \text{(B.4)}$$

Since

$$1 - H(T, \Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] \quad a.s.,$$

it is plain from (B.3) and (B.4) that

$$1 - H(T, \Delta) = \frac{1 + \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]}$$

by the usual arguments, as we recall (B.1) with (B.2). The desired expression (8.6) is finally obtained. ■

B.2 A proof of Proposition 8.4

The arguments are similar to those given in the proof of Proposition 8.3. We begin by noting that

$$H^{\star}(T,\Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L^*_{\Delta}(T^r_n-) > 0\right] \quad a.s. \tag{B.5}$$

In order to use (B.5), fix $k = 0, 1, \ldots$ and consider the cache-fresh* tracking process on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$: A download is requested at time $T^r_{\mu_k}$, resulting in a cache placement at time $T^r_{\mu_k} + \Delta$. If no update occurs before this cache placement, then the validation process is reinitialized at time $T^r_{\mu_k} + \Delta$ to the value $\min(T, U_e(T^r_{\mu_k}) - \Delta)$, and returns to zero with the expiration of this TTL. In other words, $L^*_{\Delta}((T^r_{\mu_k} + \Delta + \min(T, U_e(T^r_{\mu_k}) - \Delta))-) = 0$. On the other hand, if an update occurs before the placement, i.e., $U_e(T^r_{\mu_k}) \leq \Delta$,[1] then $L^*_{\Delta}(t) = 0$ on the entire interval $(T^r_{\mu_k}+\Delta, T^r_{\mu_{k+1}}]$.

As already discussed in the proof of Proposition 8.3, the hits on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$ can occur only in the subinterval $(T^r_{\mu_k}+\Delta, T^r_{\mu_k}+\Delta+T)$. If $U_e(T^r_{\mu_k}) \leq \Delta$, none of these requests can be hits*. However, if $\Delta < U_e(T^r_{\mu_k})$, then all the requests made in the interval $(T^r_{\mu_k} + \Delta, T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})))$ are hits*, while none of those made in the interval $[T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})), T^r_{\mu_k} + \Delta + T]$ are as they are all hit but miss* requests. Summarizing, we conclude that the number H^*_k of hits* in

[1] Only applies when $\Delta > 0$.

the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$ is given by the difference

$$H^*_k = R((T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})))-) - R(T^r_{\mu_k} + \min(\Delta, U_e(T^r_{\mu_k}))).$$

Consequently, for each $k = 0, 1, \ldots$, we can write

$$\sum_{n=1}^{\mu_k} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right] = \sum_{\ell=0}^{k-1} \sum_{\mu_\ell < n \le \mu_{\ell+1}} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right] = \sum_{\ell=0}^{k-1} H^\star_\ell. \qquad \text{(B.6)}$$

Under the renewal assumptions on the independent processes $\{T^r_n,\ n = 0, 1, \ldots\}$ and $\{T^u_m,\ m = 0, 1, \ldots\}$, we can easily verify the following: First, we have the equalities

$$\begin{aligned} &\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R((T^r_{\mu_\ell} + \min(\Delta + T, U_e(T^r_{\mu_\ell})))-) - R(T^r_{\mu_\ell})\right) \\ = \quad &\mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] \quad a.s., \end{aligned}$$

and

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R(T^r_{\mu_\ell} + \min(\Delta, U_e(T^r_{\mu_\ell}))) - R(T^r_{\mu_\ell})\right) = \mathbf{E}\left[R(\min(\Delta, U_e))\right] \quad a.s.$$

In both cases the rv U_e is taken to be independent of the counting process $\{R(t),\ t \ge 0\}$. Combining, we get

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} H^\star_\ell = \mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right] \quad a.s. \qquad \text{(B.7)}$$

To conclude, we report (B.6) and (B.7) into (B.5), and this yields

$$H^\star(T, \Delta) = \frac{\mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]}$$

as we recall (B.1) with (B.2).

In order to derive the alternative expression (8.8) we expand the expression (8.7), and by taking the rv U_e to be independent of the point process of requests we get

$$\begin{aligned} H^\star(T, \Delta) \quad = \quad &\frac{\mathbf{P}\left[U_e \ge T + \Delta\right] \mathbf{E}\left[R(T + \Delta)\right] - \mathbf{P}\left[U_e > \Delta\right] \mathbf{E}\left[R(\Delta)\right]}{\mathbf{E}\left[R(T + \Delta)\right] + 1} \\ &+ \frac{\mathbf{E}\left[\mathbf{1}\left[\Delta < U_e \le T + \Delta\right] R(U_e)\right]}{\mathbf{E}\left[R(T + \Delta)\right] + 1}. \end{aligned}$$

Then, simple manipulations of this last term lead to

$$\mathbf{E}\left[\mathbf{1}\left[\Delta < U_e \leq T + \Delta\right] R(U_e)\right] = \lambda_U \int_{\Delta}^{\Delta+T} \mathbf{E}\left[R(u)\right] \mathbf{P}\left[U > u\right] \, du$$

and the desired result (8.8) follows from the fact that

$$\begin{aligned} & \frac{\mathbf{P}\left[U_e \geq T + \Delta\right] \mathbf{E}\left[R(T + \Delta)\right] - \mathbf{P}\left[U_e > \Delta\right] \mathbf{E}\left[R(\Delta)\right]}{\mathbf{E}\left[R(T + \Delta)\right] + 1} \\ = \; & \mathbf{P}\left[U_e > T + \Delta\right] H(T, \Delta) - \frac{\mathbf{P}\left[\Delta < U_e \leq T + \Delta\right] \mathbf{E}\left[R(\Delta)\right]}{\mathbf{E}\left[R(T + \Delta)\right] + 1} \\ = \; & \mathbf{P}\left[U_e > \Delta\right] H(T, \Delta) - \frac{\mathbf{P}\left[\Delta < U_e \leq T + \Delta\right] \mathbf{E}\left[R(T + \Delta)\right]}{\mathbf{E}\left[R(T + \Delta)\right] + 1}, \end{aligned}$$

as we recall that U_e is distributed according to (8.3). ■

Appendix C

A Proof of Proposition 9.2

Throughout this proof we shall make use of the $\mathbb{N}$-valued rvs $\{\eta_m,\ m = 0, 1, \ldots\}$ defined recursively through (9.2). In addition, we find it useful to introduce the sequence of $\mathbb{N}$-valued rvs $\{\zeta_k,\ k = 0, 1, \ldots\}$ defined through

$$\zeta_k = \sup\{m = 0, 1, \ldots : T^u_m \leq T^r_{\eta_k}\}, \tag{C.1}$$

so that $T^u_{\zeta_k}$ identifies the most recent document update time taking place prior to the request at time $T^r_{\eta_k}$. With (C.1), the requests and updates processes are related through

$$T^r_{\eta_k} = T^u_{\zeta_k} + R_e(T^u_{\zeta_k}), \tag{C.2}$$

or alternatively via

$$T^u_{\zeta_k+1} = T^u_{\zeta_k} + U_{\zeta_k+1} = T^r_{\eta_k} + U_e(T^r_{\eta_k}), \tag{C.3}$$

for each $k = 0, 1, \ldots$.

Since the rvs $\{\eta_m,\ m = 0, 1, \ldots\}$ monotonically exhaust $\mathbb{N}$ a.s., it is plain that

$$H(\Delta) = 1 - \lim_{m\to\infty} \frac{1}{\eta_m} \sum_{n=1}^{\eta_m} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] \quad a.s. \tag{C.4}$$

In order to make use of (C.4), fix $m = 0, 1, \ldots$ and consider the dynamics of the freshness tracking process on the interval $[T^r_{\eta_m}, T^r_{\eta_{m+1}})$: If $T^r_{\eta_m} + \Delta > T^r_{\eta_{m+1}}$, the miss requests in this interval[1] are those that arrive to the cache during $[T^r_{\eta_m}, T^r_{\eta_m} + U_e(T^r_{\eta_m}))$, as we recall that the request at time $T^r_{\eta_{m+1}}$ is the first one presented after the update at time $T^r_{\eta_m} + U_e(T^r_{\eta_m}) = T^u_{\zeta_m+1}$. In fact, there are exactly $1 + R(T^r_{\eta_m} + U_e(T^r_{\eta_m})) - R(T^r_{\eta_m})$ such requests. On the other hand, when $T^r_{\eta_m} + \Delta \leq T^r_{\eta_{m+1}}$, miss requests are those presented in the time interval $[T^r_{\eta_m}, T^r_{\eta_m} + \Delta)$, and there are exactly $1 + R(T^r_{\eta_m} + \Delta) - R(T^r_{\eta_m})$ such requests. To summarize, denote by M_m the number of miss occurrences on each interval $[T^r_{\eta_m}, T^r_{\eta_{m+1}})$, therefore

$$M_m \;\; = \;\; 1 + R(T^r_{\eta_m} + \min(\Delta, U_e(T^r_{\eta_m}))) - R(T^r_{\eta_m}).$$

Combining all facts and observations, we get that for each $m = 0, 1, \ldots$, it holds that

$$\sum_{n=1}^{\eta_m} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] = \sum_{\ell=0}^{m-1} \sum_{\eta_\ell < n \leq \eta_{\ell+1}} \mathbf{1}\left[L_\Delta(T^r_n-) = 0\right] = \sum_{\ell=0}^{m-1} M_\ell.$$

Now, the processes $\{T^r_n,\ n = 0, 1, \ldots\}$ and $\{T^u_m,\ m = 0, 1, \ldots\}$ are assumed mutually independent renewal processes, and by the Strong Law of Large Numbers we can write

$$\lim_{m\to\infty} \frac{1}{m} \sum_{\ell=0}^{m-1} M_\ell = 1 + \mathbf{E}\left[R(\min(\Delta, U_e(T^r_{\eta_0})))\right] \quad a.s. \tag{C.5}$$

By the fact that the processes $\{R(R_e(0) + t),\ t \geq 0\}$ and $\{1 + R(t),\ t \geq 0\}$ are stochastically equivalent, an alternative expression for (C.5) is given by[2]

$$\lim_{m\to\infty} \frac{1}{m} \sum_{\ell=0}^{m-1} M_\ell = \mathbf{E}\left[R(R_e(T^u_{\zeta_0}) + \min(\Delta, U_e(T^r_{\eta_0})))\right] \quad a.s.$$

[1] This only applies when $\Delta > 0$.

[2] Here, we make use of the fact that $T^u_{\zeta_0} = 0$, which follows directly from the modeling framework; see Section 7.1.1 and Section 7.1.2 for details.

Applying the relationships (C.2) and (C.3) in this last result we conclude that

$$U_{\zeta_k+1} = R_e(T^u_{\zeta_k}) + U_e(T^r_{\eta_k}), \quad k = 0, 1, \ldots,$$

and the result (C.5) can again be rewritten to yield

$$\lim_{m\to\infty} \frac{1}{m} \sum_{\ell=0}^{m-1} M_\ell = \mathbf{E}\left[R(\min(R_e(T^u_{\zeta_0}) + \Delta, U))\right] \quad a.s.$$

Next, it is simple to check that

$$\begin{aligned} & \mathbf{E}\left[R(\min(R_e(T^u_{\zeta_0}) + \Delta, U))\right] \\ = \; & \mathbf{E}\left[\mathbf{1}\left[U < R_e(T^u_{\zeta_0}) + \Delta\right] R(U)\right] \\ & + \mathbf{E}\left[\mathbf{1}\left[U \geq R_e(T^u_{\zeta_0}) + \Delta\right] R(R_e(T^u_{\zeta_0}) + \Delta)\right]. \end{aligned} \tag{C.6}$$

By the fact that

$$R(R_e(T^u_{\zeta_0}) + \Delta) = 1 + \sum_{\ell=2}^{\infty} \mathbf{1}\left[\sum_{k=2}^{\ell} R_k \leq \Delta\right] = 1 + R(\Delta)$$

we get that the rvs $R(R_e(T^u_{\zeta_0}) + \Delta)$ and $\mathbf{1}\left[U \geq R_e(T^u_{\zeta_0}) + \Delta\right]$ are independent. This holds true since $R_e(T^u_{\zeta_0})$ depends on the rv R_1, which under the renewal assumption on the process of requests is independent of the rvs $R_k,\ k = 2, 3, \ldots$. Then, (C.6) becomes

$$\begin{aligned} & \mathbf{E}\left[R(\min(R_e(T^u_{\zeta_0}) + \Delta, U))\right] \\ = \; & \mathbf{E}\left[\mathbf{1}\left[U < R_e(T^u_{\zeta_0}) + \Delta\right] R(U)\right] \\ & + \mathbf{P}\left[U \geq R_e(T^u_{\zeta_0}) + \Delta\right] (1 + \mathbf{E}\left[R(\Delta)\right]), \end{aligned} \tag{C.7}$$

and we replace $R_e(T^u_{\zeta_0})$ with R_e as we recall that the two rvs are identically distributed.

To complete the proof, it is left to calculate $\lim_{m\to\infty} \frac{\eta_m}{m}$. In order to do so, we must first define the delayed renewal process associated with the requests: Consider the point process $\{T_n^{\hat{r}}, n = 0, 1, \ldots\}$ with the understanding that the n^{th} request occurs at time $T_n^{\hat{r}}$, and $T_0^{\hat{r}} = T_1^u = U_1$. Thus, $T_n^{\hat{r}} \leq T_{n+1}^{\hat{r}}$ for each $n = 0, 1, \ldots$. Let $\{\hat{R}_{n+1},\ n = 0, 1, \ldots\}$ denote the sequence of inter-request times with $\hat{R}_{n+1} = T_{n+1}^{\hat{r}} - T_n^{\hat{r}}$ for each $n = 0, 1, 2, \ldots$. This point process is assumed to be a renewal process, and it is related to the point process of requests defined in Section 7.1.1 as follows: $\{\hat{R}_{n+1}, n = 1, 2, \ldots\}$ form a sequence of i.i.d rvs, each distributed according to the common cdf F_R; the rv $\hat{R}_1 = R_e(T_1^u)$ is distributed according to the cdf F_{R_e}, and $\hat{R}_1, \hat{R}_{n+1}, n = 1, 2, \ldots$ are mutually independent rvs.

The counting process $\{\hat{R}(t),\ t \geq 0\}$ associated with the point process $\{T_n^{\hat{r}}, n = 0, 1, \ldots\}$ is given by

$$\hat{R}(t) = \sup\{n = 0, 1, \ldots :\ T_n^{\hat{r}} \leq t\}, \quad t \geq 0 \tag{C.8}$$

so that $\hat{R}(t)$ counts the number of requests in the interval $(0, t]$. Here, $\{T_n^{\hat{r}}, n = 0, 1, \ldots\}$ is a delayed renewal process, therefore there exists a rate λ_R that satisfies

$$\lambda_R = \lim_{t\to\infty} \frac{\hat{R}(t)}{t} \quad a.s.$$

In this case, the Renewal Function is known in closed form [69], and is given by

$$\mathbf{E}\left[\hat{R}(t)\right] = \lambda_R t, \quad t \geq 0. \tag{C.9}$$

To finalize the proof, we have

$$\begin{aligned}
\lim_{m\to\infty} \frac{\eta_m}{m} &= \lim_{m\to\infty} \frac{1}{m} \sum_{\ell=0}^{m-1} \left(R(T_{m+1}^u -) - R(T_m^u)\right) \\
&= \lim_{m\to\infty} \frac{1}{m} \sum_{\ell=0}^{m-1} \left(R((T_m^u + U_{m+1}) -) - R(T_m^u)\right)
\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{E}\left[\hat{R}(U-)\right] \quad a.s. \\
&= \lambda_R \mathbf{E}\left[U-\right] \quad a.s. \\
&= \lambda_R \lambda_U^{-1} \quad a.s. \qquad \text{(C.10)}
\end{aligned}
$$

By substituting (C.7) into (C.5), and applying the results (C.5) and (C.10) in the hit rate definition (C.4), we obtain the desired result (9.5). ■

We conclude this appendix with the derivation of the alternate expression (9.6) and the inequality (9.7). First, observe that $R(U)$ can be expressed as

$$
R(U) = \mathbf{1}\left[R_e(0) < U\right] + \sum_{\ell=2}^{\infty} \mathbf{1}\left[R_e(0) + \sum_{k=2}^{\ell} R_k \leq U\right].
$$

Under the enforced assumptions on the point processes of user requests and document updates (see Section 7.1.1 and Section 7.1.2, respectively), we have the identification $T_{\zeta_0}^u = 0$. We can therefore write the first term in (C.7) as

$$
\begin{aligned}
\mathbf{E}\left[\mathbf{1}\left[U < R_e(0) + \Delta\right] R(U)\right] &= \mathbf{E}\left[\mathbf{1}\left[0 < U - R_e(0) < \Delta\right]\right] \\
&\quad + \mathbf{E}\left[\sum_{\ell=2}^{\infty} \mathbf{1}\left[\sum_{k=2}^{\ell} R_k \leq U - R_e(0) < \Delta\right]\right],
\end{aligned}
$$

and under the renewal assumption on the request process this expression becomes

$$
\begin{aligned}
&\mathbf{E}\left[\mathbf{1}\left[U < R_e(0) + \Delta\right] R(U)\right] \\
&= \lambda_R \int_0^{\infty} \int_{(u-\Delta)^+}^{u} (1 + \mathbf{E}\left[R(u-r)\right]) \mathbf{P}\left[R > r\right] \, dr \, dF_U(u).
\end{aligned}
$$

By applying similar arguments on the second term in (C.7), we find that

$$
\begin{aligned}
\mathbf{P}\left[U \geq R_e(0) + \Delta\right] &= \mathbf{E}\left[\lambda_R \int_0^{(u-\Delta)^+} \mathbf{P}\left[R > r\right] \, dr\right] \\
&= \mathbf{E}\left[\lambda_R \int_0^{\infty} \mathbf{1}\left[U > r + \Delta\right] \mathbf{P}\left[R > r\right] \, dr\right] \\
&= \lambda_R \int_0^{\infty} \mathbf{P}\left[U > r + \Delta\right] \mathbf{P}\left[R > r\right] \, dr,
\end{aligned}
$$

and the expression (9.6) for the hit rate follows by reporting the last two results into (9.5), and replacing the denominator with (C.10).

It is now left to prove the upper bound (9.7). Since the counting process $\{R(t),\ t \geq 0\}$ is non-decreasing in t, we have that

$$\mathbf{E}\left[\mathbf{1}\left[U < R_e(0) + \Delta\right] R(U)\right] \ \leq \ \mathbf{E}\left[\mathbf{1}\left[U < R_e(0) + \Delta\right] R(R_e(0) + \Delta)\right].$$

Under the renewal assumption on the point process of requests, the rv $R(R_e(0) + \Delta)$ and the rv $1+R(\Delta)$ are identically distributed. Using the fact that the rv $R(R_e(0)+\Delta)$ is independent of $R_e(0)$ (shown earlier in this appendix), we get

$$\mathbf{E}\left[\mathbf{1}\left[U < R_e(0) + \Delta\right] R(U)\right] \ \leq \ \mathbf{P}\left[U < R_e(0) + \Delta\right] \mathbf{E}\left[1 + R(\Delta)\right],$$

which leads directly to (9.7).

Bibliography

[1] S. V. Adve, *Designing Memory Consistency Models For Shared-Memory Multiprocessors*, Ph.D. Thesis, Department of Computer Sciences, University of Wisconsin-Madison, Madison (WI), 1993.

[2] A. Aho, P. Denning and D. Ullman, “Principles of optimal page peplacement,” Journal of the ACM, Vol.18, no. 1, January 1971.

[3] V. Almeida, A. Bestavros, M. Crovella and A. Oliveira, “Characterizing reference locality in the WWW,” In Proceedings of the 4th International Conference on Parallel and Distributed Information Systems (PDIS), pp. 92-107, 1996.

[4] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, “Evaluating content management techniques for Web proxy caches,” In ACM Performance Evaluation Review, Vol. 27, no. 4, pp. 3–11, March 2000.

[5] S. Asmussen and M. Bladt, “Renewal theory and queueing algorithms for matrix-exponential distributions,” in: *Matrix-Analytic Methods in Stochastic Models*, S. R. Chakravarthy and A. A. Alfa, Eds., Marcel Dekker, New-York (NY), 1996.

[6] O. I. Aven, E. G. Coffman and Y. A. Kogan, *Stochastic Analysis of Computer Storage*, D. Reidel Publishing Company, Dordrecht (Holland), 1987.

[7] O. Bahat and A. M. Makowski, "Optimal replacement policies for non-uniform cache objects with optional eviction," In Proceedings of INFOCOM 2003, San Franscisco (CA), April 2003.

[8] O. Bahat and A. M. Makowski, "Measuring consistency in TTL-based caches," In Proceedings of Performance 2005, Juan-les-Pins (France), October 2005.

[9] A. Balamash and M. Krunz, "An overview of Web caching replacement algorithms," In the IEEE Communications Surveys and Tutorials, Vol. 6, no. 2, 2004.

[10] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web client access patterns: Characteristics and caching implications," World Wide Web, Vol. 2., no. 1-2, pp. 15 - 28, 1999.

[11] G. Barish and K. Obraczka, "World Wide Web caching: Trends and techniques," IEEE Communications Magazine, Vol. 38, no. 5, pp. 178-185, 2000.

[12] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*. Holt, Rinehart and Winston, New York, 1975.

[13] F. Beutler and K. Ross, "Optimal policies for controlled markov chains with a constraint," Journal of Mathematical Analysis and Applications, Vol. 112, pp. 236-252, 1985.

[14] M. A. Blaze, *Caching in large-scale distributed file systems*, Ph.D. Thesis, Princeton University, January 1993.

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and Zipf-like distributions: Evidence and implications,” In Proceedings of IEEE INFOCOM 1999, New York (NY), March 1999.

[16] M. Brown, “Bounds, inequalities, and monotonicity properties for some specialized renewal processes,” in Ann. Prob. 8, pg. 227-240, 1980.

[17] P. Cao and S. Irani, “Cost-aware WWW proxy caching algorithms,” In Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, pp. 193-206, Dec 1997.

[18] P. Cao and C. Liu, “Maintaining strong cache consistency in the World-Wide Web,” IEEE Transactions on Computers (COMP-47), pp. 445–457, 1998.

[19] I. Cidon, S. Kutten, and R. Soffer, “Optimal allocation of electronic content,” in Proceedings of IEEE INFOCOM 2001, Anchorage (AL), April 2001.

[20] E. Coffman and P. Denning, *Operating Systems Theory*, Prentice-Hall, NJ, 1973.

[21] E. Cohen and H. Kaplan, “Aging through cascaded caches: Performance issues in the distribution of web content,” in Proceedings of SIGCOMM 2001 (Computer Communication Review) Vol. 31, no. 4, pp. 41-54, October 2001.

[22] E. Cohen and H. Kaplan, “Refreshment policies for web content caches,” In Proceedings of INFOCOM 2001, Anchorage (AL), April 2001.

[23] E. Cohen and H. Kaplan, “The age penalty and its effect of cache performance,” In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS), 2001.

[24] E. Cohen, E. Halperin and H. Kaplan, “Performance aspects of distributed caches using TTL-based consistency,” In Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP’01), London (England), 2001.

[25] J. Dilley, M. Arlitt and S. Perret, “Enhancement and validation of Squid’s cache replacement policy,” HP Labs Technical Report HPL-1999-69.

[26] K. Elhardt and R. Bayer, “A database cache for high performance and fast restart in database systems,” ACM Transactions on Database Systems, Vol. 9, pp. 503-525, December 1984.

[27] A. Feldman, “Characteristics of TCP connections arrivals,” Technical Report, AT&T Labs-Research, Florham Park (NJ), 1998.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*, The Internet Society, June 1999.

[29] R. Fonseca, V. Almeida, M. Crovella and B. Abrahão, “On the intrinsic locality properties of Web reference streams*,” In Proceedings of INFOCOM 2003, San Franscisco (CA), April 2003.

[30] M. J. Franklin, M. J. Carey, and M. Livny, “Transactional client-server cache consistency: Alternatives and performance,” in ACM Transactions on Database Systems, Vol. 22, no. 3, pp. 315-363, September 1997.

[31] J. Gwertzman and M. Seltzer, “World-Wide Web cache consistency,” in Proceedings of the 1996 USENIX Technical Conference, San Diego (CA), January 1996.

[32] D. Heyman and M. Sobel, *Stochastic Models in Operations Research, Volume II: Stochastic Optimization*, McGraw-Hill, New York (NY), 1984.

[33] S. Hosseini-Khayat, “On optimal replacement of nonuniform cache objects,” In IEEE Transactions on Computers, Vol. 49, no. 8, August 2000.

[34] Y. Hou, J. Pan, B. Li, X. Tang, and S. Panwar, “Modeling and analysis of an expiration-based hierarchical caching system,” in Proceedings of GLOBECOM 2002, November 2002.

[35] K. Gharachorloo, D. Lenoski, J. Laudon, and P. Gibbons, A. Gupta and J. Hennessy, “Memory consistency and event ordering in scalable shared-memory multiprocessors,” in Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 15-26, May 1990.

[36] S. Jin and A. Bestavros, “GreedyDual* Web caching algorithm: Exploiting the two sources of temporal locality in Web request streams,” In Proceedings of the 5th International Web Caching and Content Delivery Workshop, Lisbon (Portugal), May 2000.

[37] S. Jin and A. Bestavros, “Popularity-aware GreedyDual-Size Web proxy caching algorithms,” In Proceedings of ICDCS’2000: The IEEE International Conference on Distributed Computing Systems, Taiwan, May 2000.

[38] S. Jin and A. Bestavros, “Sources and characteristics of Web temporal locality,” In Proceedings of MASCOTS’2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Fransisco, CA, August 2000.

[39] S. Jin and A. Bestavros, “Temporal locality in Web request streams: Sources, characteristics, and caching implications,” In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS’2000), Santa Clara, CA, June 2000.

[40] J. Jung, A.W. Berger, and H. Balakrishnan, “Modeling TTL-based internet caches,” in Proceedings of INFOCOM 2003, San Francisco (CA), March 2003.

[41] A. Karlin, S. Phillips and P. Raghavan, “Markov paging,” In Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, pp. 208–217, October 1992.

[42] J. Kawash, “Consistency models for internet caching,” in Proceedings of the Winter International Symposium on Information and Communication Technology, January 2004.

[43] Y. A. Kogan, “Analytic investigation of replacement algorithms for Markov and Semi-Markov program behavior models,” In Automation and Remote Control, Vol. 38, no. 2, pp. 109-111, 1977.

[44] B. Krishnamurthy and C. E. Wills, “Study of piggyback cache validation for proxy caches in the World Wide Web,” in Proceedings of USENIX Symposium on Internet Technologies and Systems, Monterey (CA), December 1997.

[45] B. Krishnamurthy and C. E. Wills, “Piggyback server invalidation for proxy cache coherency,” in Proceedings of the 7th WWW Conference, Brisbane (Australia), April 1998.

[46] P. Krishnan, D. Raz and Y. Shavitt, “The cache location problem,” IEEE/ ACM Transactions on Networking, Vol. 8, no. 5, pp. 568-582, October 2000.

[47] L. Lamport, “How to make a multiprocessor computer that correctly executes multiprocess programs,” in IEEE Transactions on Computers, Vol. C-28, no. 9, September 1979.

[48] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, “Consistency maintenance in peer-to-peer file sharing networks,” in Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP’03), San Jose (CA), pp. 76–85, June 2003.

[49] B. Li, M. J. Golin, G. F. Italiano, and X. Deng, “On the optimal placement of Web proxies in the Internet,” In Proceedings of IEEE INFOCOM 1999, New York (NY), March 1999.

[50] G. Lorden, “On excess over the boundary,” Annals of Mathematical Statistics Vol. 41, pp. 521–527, 1970.

[51] A. Mahanti, D. Eager and C. Williamson, “Temporal locality and its impact on Web proxy cache performance,” In Performance Evaluation, Vol. 42, no. 2-3, October 2000.

[52] D.-J. Ma, A.M. Makowski and A. Shwartz, “Stochastic approximations for finite–state Markov chains,” Stochastic Processes and Their Applications, vol. 35, pp. 27-45, 1990.

[53] K.T. Marshall, “Linear bounds on the renewal function,” SIAM Journal on Applied Mathematics Vol. 24, pp. 245–250, 1973.

[54] M. Mikhailov and C. E. Wills, “Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content,” in Proceedings of the 12th International Conference on World Wide Web, Budapest, Hungary, 2003.

[55] P. V. Mockapetris, *RFC 1034: Domain Names - Concepts and Facilities*, USC/Information Sciences Institute, November, 1987.

[56] P. V. Mockapetris, *RFC 1035: Domain Names - Implementation and Specification*, USC/Information Sciences Institute, November, 1987.

[57] B. Nam and K. Koh, “Periodic polling for Web cache consistency,” In Proceedings of the Internation Conference on the World Wide Web (WebNet 99), Honolulu (HI), pp. 800 - 804, October 1999.

[58] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, “Caching in the Sprite network file system,” ACM Transactions on Computer Systems, Vol. 6, no. 1, pp. 134-154, February 1988.

[59] P. Nuggehalli, V. Srinivasan and C. F. Chiasserini, “Energy-efficient caching strategies in ad hoc wireless networks,” In the Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Annapolis (MD), 2003.

[60] V. N. Padmanabhan and L. Qiu, “The content and access dynamics of a busy Web site: Findings and implications,” in Proceedings of the ACM SIGCOMM

2000 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm (Sweden), August 2000.

[61] A. E. Papathanasio and M. L. Scott, “Energy efficient prefetching and caching,” In Proceedings of the 2004 USENIX Annual Technical Conference (USENIX’04), Boston (MA), 2004.

[62] V. Paxson and S. Floyd, “Wide area traffic: The failure of Poisson modeling,” IEEE/ACM Transactions on Networking, Vol. TON-3, pp. 226–244, 1995.

[63] K. Peterson and K. Li, “An evaluation of multiprocessor cache coherence based on virtual memory support,” in Proceedings of the 8th International Parallel Processing Symposium (IPPS’94), pp. 158–164, 1994.

[64] V. Phalke and B. Gopinath, “An inter-reference gap model for temporal locality in program behavior,” In Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 291 - 300, Canada 1995.

[65] S. Potlibnig and L. Böszörmenyi, “A survey of Web cache replacement strategies,” In ACM Computing Surveys (CSUR), Vol. 35, no. 4, pp. 374-398, December 2003.

[66] K. Psounis, A. Zhu, B. Prabhakar and R. Motwani, “Modeling correlations in Web traces and implications for designing replacement policies,” Computer Networks, Vol. 45, no. 4, pp. 379-398, July 2004.

[67] J. Rajendiran, J. Patwardhan, V. Abhijit, R. Lakhotia, and A. Vadhat, “Exploring the benefits of a continuous consistency model for wireless Web por-

tals," in Proceedings of the Second IEEE Workshop on Internet Applications (WIAPP2001), San Jose (CA), July 2001.

[68] S.M. Ross, *Introduction to stochastic dynamic programming*, Academic Press, New York (NY), 1984.

[69] S. Ross, *Stochastic Processes*, Second Edition, Wiley, 1996.

[70] A. J. Smith, "Cache memories," ACM Computing Surveys, Vol. 14, no. 3, pp. 470-530, September 1982.

[71] D. Starobinski and D. Tse, "Probabilistic methods for Web caching," Performance Evaluation, Vol. 46, no. 2-3, pp. 125-137, 2001.

[72] K. L. Tan, J. Cai and B. C. Ooi, "An evaluation of cache invalidation strategies in wireless environments," In IEEE Transactions on Parallel and Distributed Systems (PDIS), Vol. 12, no. 8, pp. 789 - 807, August 2001.

[73] F. J. Torres-Rojas, M. Ahamad, and M. Raynal, "Timed consistency for shared distributed objects," in Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'99), Atlanta (GA), 1999.

[74] S. Vanichpun, *Comparing Strength of Locality of Reference: Popularity, Temporal Correlations, and Some Folk Theorems for the miss Miss Rates and Outputs of Caches,* Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, January 2005.

[75] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin, "Potential costs and benefits of long-term prefetching for content-distribution," Elsevier Computer Communications, Vol. 25, no. 4, pp. 367-375, March 2002.

[76] J. Wang, “A survey of Web caching schemes for the Internet,” ACM Computing Communication Review, Vol. 29, no. 5, 1999.

[77] D. Wessels, “Squid Internet object cache,” available at http://squid.nlanr.net/Squid/, August 1998.

[78] K. L. Yeung, and H. T. Wong, “Caching policy design and cache allocation in active reliable multicast,” Computer Networks, Vol. 43, no. 2, pp. 177-193, 2003.

[79] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, “Engineering Web cache consistency,” ACM Transactions on Internet Technology, Vol. 2, pp. 224–259, 2002.

[80] L. Yin, G. Cao, and Y. Cai, “A generalized target driven cache replacement policy for mobile environments*,” in Proceedings of the 2003 International Symposium on Applications and the Internet (SAINT-2003), Orlando (FL), January 2003.

[81] N.E. Young, “On-line caching as cache size varies,” In Proceedings of Symposium on Discrete Algorithms, January 1991.