

Cache Optimization Models and Algorithms

Other titles in Foundations and Trends® in Communications and Information Theory

Group Testing: An Information Theory Perspective

Matthew Aldridge, Oliver Johnson and Jonathan Scarlett

ISBN: 978-1-68083-596-0

Sparse Regression Codes

Ramji Venkataramanan, Sekhar Tatikonda and Andrew Barron

ISBN: 978-1-68083-580-9

Fundamentals of Index Coding

Fatemeh Arbabjolfaei and Young-Han Kim

ISBN: 978-1-68083-492-5

Community Detection and Stochastic Block Models

Emmanuel Abbe

ISBN: 978-1-68083-476-5

Cache Optimization Models and Algorithms

Georgios S. Paschos

Amazon.com

gpaschos@gmail.com

George Iosifidis

Trinity College Dublin

george.iosifidis@tcd.ie

Giuseppe Caire

TU Berlin

caire@tu-berlin.de

now

the essence of knowledge

Boston — Delft



Documents

Document Sharing Hub

Foundations and Trends[®] in Communications and Information Theory

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

G. Paschos, G. Iosifidis and G. Caire. *Cache Optimization Models and Algorithms*. Foundations and Trends[®] in Communications and Information Theory, vol. 16, no. 3–4, pp. 156–345, 2020.

ISBN: 978-1-68083-703-2

© 2020 G. Paschos, G. Iosifidis and G. Caire

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The ‘services’ for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Foundations and Trends[®] in Communications and Information Theory

Volume 16, Issue 3–4, 2020

Editorial Board

Alexander Barg

University of Maryland
USA

Editors

Venkat Anantharam
UC Berkeley

Giuseppe Caire
TU Berlin

Daniel Costello
University of Notre Dame

Anthony Ephremides
University of Maryland

Albert Guillen i Fabregas
Pompeu Fabra University

Dongning Guo
Northwestern University

Dave Forney
MIT

Te Sun Han
University of Tokyo

Babak Hassibi
Caltech

Michael Honig
Northwestern University

Ioannis Kontoyiannis
Cambridge University

Gerhard Kramer
TU Munich

Amos Lapidoth
ETH Zurich

Muriel Medard
MIT

Neri Merhav
Technion

David Neuhoff
University of Michigan

Alon Orlitsky
UC San Diego

Yury Polyanskiy
MIT

Vincent Poor
Princeton University

Kannan Ramchandran
UC Berkeley

Igal Sason
Technion

Shlomo Shamai
Technion

Amin Shokrollahi
EPF Lausanne

Yossef Steinberg
Technion

Wojciech Szpankowski
Purdue University

David Tse
Stanford University

Antonia Tulino
Bell Labs

Rüdiger Urbanke
EPF Lausanne

Emanuele Viterbo
Monash University

Frans Willems
TU Eindhoven

Raymond Yeung
CUHK

Bin Yu
UC Berkeley



Documents
Document Sharing Hub

Editorial Scope

Topics

Foundations and Trends® in Communications and Information Theory publishes survey and tutorial articles in the following topics:

- Coded modulation
- Coding theory and practice
- Communication complexity
- Communication system design
- Cryptology and data security
- Data compression
- Data networks
- Demodulation and Equalization
- Denoising
- Detection and estimation
- Information theory and statistics
- Information theory and computer science
- Joint source/channel coding
- Modulation and signal design
- Multiuser detection
- Multiuser information theory
- Optical communication channels
- Pattern recognition and learning
- Quantization
- Quantum information processing
- Rate-distortion theory
- Shannon theory
- Signal processing for communications
- Source coding
- Storage and recording codes
- Speech and Image Compression
- Wireless Communications

Information for Librarians

Foundations and Trends® in Communications and Information Theory, 2020, Volume 16, 4 issues. ISSN paper version 1567-2190. ISSN online version 1567-2328 . Also available as a combined paper and online subscription.

Contents

1	Introduction	2
1.1	Historical Background and Scope	2
1.2	The Content Delivery Network	4
1.3	Wireless Caching and Beyond	7
1.4	Structure	8
2	Content Popularity	11
2.1	Introduction to Caching-Related Terms	11
2.2	Power Law Popularity	13
2.3	Request Sequences	25
2.4	Discussion	36
3	Cache Eviction Policies	37
3.1	Performance Under Arbitrary Requests	37
3.2	Performance Under Stationary Requests	49
3.3	Online Popularity Learning	59
3.4	Discussion of Related Work	75
4	Caching Networks	78
4.1	Model and Optimization Variables	78
4.2	Deployment of Caching Networks	83
4.3	Bipartite Caching Networks	90

4.4	Hierarchical Caching Networks	109
4.5	General Caching Networks	120
5	Online Bipartite Caching	131
5.1	Background and Model	132
5.2	Problem Statement	134
5.3	Bipartite Supergradient Caching Algorithm	139
5.4	Extensions and Numerical Evaluation	145
5.5	Discussion of Related Work	149
6	Asymptotic Laws for Caching Networks	151
6.1	Analysis of Large Caching Networks	152
6.2	Discussion of Related Work	162
	Acknowledgments	164
	References	165

Cache Optimization Models and Algorithms

Georgios Paschos¹, George Iosifidis² and Giuseppe Caire³

¹*Amazon.com; gpaschos@gmail.com*

²*Trinity College Dublin; george.iosifidis@tcd.ie*

³*TU Berlin; caire@tub.edu*

ABSTRACT

Caching refers to the act of replicating information at a faster (or closer) medium with the purpose of improving performance. This deceptively simple idea has given rise to some of the hardest optimization problems in the fields of computer systems, networking, and the Internet, many of which remain unsolved several years after their conception. While a wealth of research contributions exists from the topics of memory systems, data centers, Internet traffic, CDNs, and recently wireless networks, the literature is dispersed and overlapping at times. In this monograph, we take a unifying modeling view: by focusing on the fundamental underlying mathematical models, we re-organize the available material into a powerful framework for performing optimization of caching systems. This way, we aspire to present a solid background for the anticipated explosion in caching research, but also provide a didactic view into how engineers have managed to infuse mathematical models into the study of caching over the last 40 years.

Georgios Paschos, George Iosifidis and Giuseppe Caire (2020), “Cache Optimization Models and Algorithms”, Foundations and Trends® in Communications and Information Theory: Vol. 16, No. 3–4, pp 156–345. DOI: 10.1561/0100000104.

1

Introduction

Storage resources and caching techniques permeate almost every area of communication networks today. In the near future, caching is set to play an important role in storage-assisted Internet architectures, information-centric networks and wireless systems, reducing operating and capital expenditures and improving the offered services. In light of the remarkable data traffic growth and the increasing number of rich-media applications, the impact of caching is expected to become even more profound than it is today. Therefore, it is crucial to design these systems in an optimal fashion, ensuring the maximum possible performance and economic benefits from their deployment. To that end, this monograph presents a collection of detailed models and algorithms, which are synthesized to build a powerful analytical framework for caching optimization.

1.1 Historical Background and Scope

The term *cache* was introduced in computer systems in 1970s to describe a memory with very fast access but typically small capacity. In computer applications, memory access often exhibits locality, i.e., most requests are related to memory blocks in a specific area known as *hot spot*.

By replicating these spots on a cache, it is possible to accelerate the performance of the entire memory system. One of the most important first problems in this context was to select which memory blocks to replicate in order to maximize the expected benefits; and several key results, e.g., the oracle MIN policy [26], were developed in that early era of computer systems. Nevertheless, the design of such *caching policies* remains one of the main challenges in caching systems.

The above caching idea was later applied to the Internet. As the population of users was growing fast in 1990s, the client-server connection model became impractical since all content requests (for web pages, in particular) were routed to few central servers. This was creating server and network congestion, and motivated the idea of using *Internet caches*. The latter are deployed closer to end users and host carefully selected web pages. Given the content popularity skewness, i.e., the fact that few web pages attract the majority of requests, even small caches can bring impressive performance benefits. Indeed, it soon became clear that *web caching* can significantly alleviate network congestion and improve the content access time for users. In these interconnected caches the problem of designing optimal caching policies is more intricate, as it requires to decide which files to cache, how to route the content and how to dimension each cache.

The last few years we witness a resurgence of interest in caching in the domain of wireless networks. The expansive growth of mobile video traffic in conjunction with exciting developments — like the use of *coding techniques* — have placed caching at the forefront of research in wireless communications. There is solid theoretical and practical evidence that memory can be a game-changer in our efforts to increase the effective throughput, and there are suggestions for deploying caches at the network core, the base stations or even at the mobile devices. Similarly, novel services that involve in-network computations, require pre-stored information (e.g., machine learning services), or are bounded by low latency constraints, can greatly benefit from caching. In fact, many caching enthusiasts argue that such services can only be deployed if they are supported by intelligent caching techniques.

Amidst these developments, it is more important than ever to model, analyze and optimize the performance of caching systems. Quite

surprisingly, many existing caching solutions, albeit practical, have not been designed using rigorous mathematical tools. Hence, the question of whether they perform optimally remains open. At the same time, the caching literature spans more than 40 years, different systems and even different research communities, and there is lack of a much-needed unified view on caching models and algorithms. This monograph aspires to fill this gap by presenting the theoretical foundations of caching and the latest conceptual and mathematical advances in this area. It provides detailed technical arguments and proofs, aiming to create a stable link between the past and future of caching analysis, and offer a useful starting point for new researchers. In the remainder of this section, we set the ground by discussing key caching systems and ideas, and explain the organization of this monograph.

1.2 The Content Delivery Network

A milestone in the evolution of caching systems was the deployment of *Content Delivery Networks*. CDNs typically consist of: (i) the origin server; (ii) the dispersed caches; (iii) the backbone network; and (iv) the points of ingress user traffic. The origin server is often deployed at a remote location with enormous storage capabilities (e.g., a datacenter) and stores all content files a user of this service might request, i.e., the entire content *catalog*. The caches are smaller servers deployed near the demand points, and are connected with the backbone network. Before CDNs, the users would establish TCP connections with the origin server in order to retrieve the content. In CDNs however, these connections are redirected to caches which serve the requests for these locally replicated files. Modern CDN systems have further evolved to optimize network traffic, offer different levels of Quality of Service (QoS), and increase the robustness of these services by, for example, protecting them from Denial of Service (DoS) attacks. In this monograph we focus on the aspect of content caching, and specifically on the intelligence involved in orchestrating the caching operations.

An iconic CDN system is the “Akamai Intelligent Platform”, see [181] for a detailed description. Akamai was one of the most prominent CDN providers in the booming Internet era of 2000s, and by 2019 was

responsible for delivering 20% approximately of the Internet traffic. Its 216 K caching servers are dispersed at network edges offering low-latency content access around the globe. The Akamai model was designed to intercept HTTP traffic using a DNS redirect: when a user wants to open a website with the HTTP protocol, it would first contact the local DNS server to retrieve the IP of the origin server. The intelligent platform replaces the DNS entry with the IP of an Akamai cache containing the requested content, and hence the HTTP request is eventually served by that cache. The intelligent operations are handled by the mapping system, which decides where to cache each content file, and accordingly maps DNS entries to caches. Although the mapping system is effectively deciding the placement of content, the local caches are also operated with reactive policies such as the famous LRU and its variants.

Benefits of Caching

The replication of few popular contents can significantly *reduce the traffic* at the backbone network. When a content is available at a nearby cache, the respective requests are redirected to that cache instead of the origin server (an event called *hit*). Therefore, caches are often scattered around the network to minimize the geodesic distance, or network hops, from the users. Previous research has investigated solutions for the optimal placement of servers, e.g., [201], and the sizing of cache storage, called cache dimensioning [135]. Since more hits mean less network traffic, an important criterion for deploying caches is the increase of the cache *hit ratio*. Pertinent optimization problems in this context include the choice of eviction policies, i.e., the dynamic selection of the contents that are evicted from an overflowing cache, and the strategic content placement for enabling cache collaboration [24].

Another benefit of caching is *latency reduction*, i.e., the decrease of elapsed time between the initiation of a request and content delivery. Typically, the latency improvement is attributed to cutting down propagation latency. Packets traversing a transcontinental link, for example, experience latency up to 250 msec due to speed-of-light limitations [218]. Given that each TCP connection involves the exchange of several messages, it might actually take seconds before a requested content is

delivered over such links. These large latencies are very harmful for e-commerce and other real-time applications, and their improvement has been one of the main market-entry advantages of CDNs. Indeed, when users retrieve contents from a nearby cache, the geodesic distance is greatly reduced and so is the propagation time that hinders the content delivery. Nevertheless, latency optimization in caching systems is an intricate task and there are some notable misconceptions.

Firstly, in most applications latency effects smaller than 30 msec do not impact the user experience. Hence, one needs to be cautious in increasing the infrastructure costs in order to deliver content faster than this threshold. In other words, when it comes to latency, a single local cache often suffices to serve a large metropolitan area. Secondly, regarding video content delivery, the latency requirements apply only to the first video chunks and not to the entire file. Delivering fast the first chunks and then exploiting the buffer capacity at the user side is often adequate for ensuring smooth reproduction, even if the later video segments are delivered with higher latency. Finally, several low latency applications, such as reactive virtual reality, vehicular control, or industrial automation, cannot typically benefit from caching since their traffic is not reusable. However, we stress that there are scenarios where one can exploit caching (e.g., using proactive caching policies) in order to boost the performance of such demanding services.

Another important effect of web caching is that it *balances the server load*. For example, the Facebook Photo CDN leverages web browser caches on user devices, edge regional servers, and other caches in order to reduce the traffic reaching the origin servers. Notably, browser caches serve almost 60% of traffic requests, due to the fact that users view the same content multiple times. Edge caches serve 20% of the traffic (i.e., approximately 50% of traffic not served by browser caches), and hence offer important off-network bandwidth savings by serving locally the user sessions. The remaining 20% of content requests are served at the origin, using a combination of slow back-end storage and a fast origin-cache [116]. This CDN functionality shields the main servers from high load and increases the scalability of the architecture. Note that the server load is minimized when the cache hits are maximized, and hence the problem of server load minimization is equivalent to

cache hit maximization. Therefore, in the remaining of this monograph we will focus on hit maximization, as well as bandwidth and latency minimization.

1.3 Wireless Caching and Beyond

Caching has also been considered for improving content delivery in wireless networks, see [191] and references therein. There is consensus that network capacity enhancements by means of improving physical layer access rates or through denser deployment of base stations is a costly approach and outpaced by the fast-increasing data traffic [60]. Caching techniques promise to fill this gap, and several interesting ideas have been suggested to this end: (i) deep caching at the evolved packet core (EPC) in order to reduce content delivery delay [242]; (ii) caching at the base stations to alleviate congestion in their throughput-limited backhaul links [103]; (iii) caching at the mobile devices to leverage device-to-device communications [101]; and (iv) coded caching for accelerating transmissions over a broadcast medium [163].

Furthermore, techniques that combine caching with coding demonstrate revolutionary *goodput* scaling in bandwidth-limited cache-aided networks. This has motivated researchers to revisit the fundamental question of how memory “interacts” with other resources. The topic of *coded caching* started as a powerful tool for broadcast mediums, and led towards establishing an information theory for memory. Similarly, an interesting connection between memory and processing has been identified [154], creating fresh opportunities for improving the performance of distributed and parallel computing systems. These lines of research have re-stirred the interest in joint consideration of bandwidth, processing and memory resources.

At the same time, the advent of technologies such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV) create new opportunities for leveraging caching in wired and wireless networks. Namely, they enable the fine-grained and unified control of storage, computing and network bandwidth, and support the flexible deployment of in-network caching services. This gives rise to the new concept of content-centric network architectures that aim to use storage

and caching as a means to rethink the Internet operation. Similarly, new business models are emerging as new players are entering the content delivery market. Service providers like Facebook are acquiring their own CDNs, network operators deploy in-network cache servers to reduce their bandwidth expenditures, and content providers like Google, Netflix, and Amazon use caches to replicate their content world-wide. Interestingly, smaller content providers can buy caching resources on the cloud market to instantiate their service *just in time and space*. These novel concepts create, unavoidably, new research questions for caching architectures and the caching economic ecosystem, and one of our goals is to provide the fundamental underlying mathematical theories that can support research in these exciting directions.

1.4 Structure

In this subsection we provide a quick summary of the monograph, serving both as a warm-up for reading it, as well as a map with directions to specific information.

We begin in Section 2 with a detailed treatment of content popularity models, a crucial factor shaping the performance of caching policies. We first explain the power-law popularity model and how we can infer its parameters from a dataset, and then use it for the purpose of cache dimensioning. We then define the *Independence Reference Model* (IRM) for describing a request generation process. IRM is a widely used model for caching analysis because of its tractability, but it has limited accuracy since it fails to capture two observed correlation effects of request processes, namely *temporal and spatial locality*. We discuss state-of-the-art mathematical models which are more accurate than IRM in that respect, but also more difficult to analyze. For the case of temporal correlations, we provide the optimal rule for popular/unpopular content classification that maximizes cache performance.

In Section 3 we explore the class of *online eviction policies*, also known as replacement policies. A cache receives requests and a rule must be employed for evicting a content when the cache overflows. The design of such policies is an equally challenging and important problem, and we present the main pertinent results. We begin with the case

of arbitrary requests, for which an oracle policy, known as “Belady”, achieves the maximum number of hits under any request sequence. This policy requires knowledge of future requests, and therefore is useful only as a benchmark. Using the Belady policy we prove that the “Least Recently Used” (LRU) rule provides the best competitive performance among all online policies, i.e., those that do not have information about future requests. Then, for stationary IRM requests the “Least Frequently Used” (LFU) rule is the optimal, as it estimates the (assumed static) content popularity using the observed frequencies. We also study the *characteristic time* approximation, with which we obtain the performance of LRU for stationary requests, as well that of *Time To Live* (TTL) caches which allow to optimally tune different content hit probabilities. Last, we depart from the stationary assumption and take a model-free approach inspired by the Machine Learning framework of *Online Convex Optimization* (OCO). We present an adaptation of the Zinkevich’s online gradient policy to the caching problem, and show that it achieves the optimal *regret*, i.e., the smallest possible losses with respect to the best static cache configuration with future knowledge.

In Section 4 we study caching networks (CNs), i.e., systems where multiple caches are interconnected via a network. We focus on *proactive caching* policies which populate the caches based on estimated demand. In CNs, the designer needs to decide where to cache each content file (caching policy), how to route the contents from caches to the requesters (routing policy), and also the capacity of the caches and network links. Therefore, we start the section by explaining this general *CN design and management problem*. This is a notoriously hard problem, that cannot be solved optimally for large CNs and content catalogs. To gain a better understanding into the available solution methodologies, we survey a number of important subproblems: (i) the *cache dimensioning* problem where we decide where to place storage in the network; (ii) the *content caching* in bipartite and tree graphs; and (iii) the *joint content caching and routing* problem in general graphs. Although these are all special cases of the general CN problem, they are governed by significantly different mathematical theories. Therefore, our exposition in this section serves to clarify where each mathematical theory applies best, and how to get a good approximate guarantee for each scenario.

In the following Section 5 we take an approach that combines the two previous sections. We study a CN where the content popularity is unknown, and therefore the goal is to design a joint caching and routing policy which at the same time learns the content popularity, decides in an online manner which files to cache, and in a reactive fashion how to route the contents to requesters. These results generalize the OCO-based policy that was introduced in Section 3 for a single cache. We show that a policy that takes a step in the direction of a subgradient of the previous slot's utility function can provide “no regret” in the CN scenario as well. This means that the designed policy gradually learns to match the performance of the optimal static policy, a theoretical result that is validated through trace-driven numerical experiments. The analysis in this section is general enough to account for changes in the network structure and cache reconfiguration costs.

In the last Section 6 we examine a very large (scaling) network of caches, arranged in a square grid. For this special case, we show that it is possible to relax the original combinatorial problem and obtain a relaxed solution via convex optimization. Then it can be shown that the relaxed solution is of the same order of performance with the actual integral, hence we can use it to understand the scaling performance of a large caching network. The section includes detailed results about the sustainability of networks aiming to deliver content in different regimes of: (i) network size; (ii) catalog size; and (iii) cache size.

2

Content Popularity

Managing a cache effectively requires deep understanding of content popularity and the principles underlying the content request sequences. This section introduces basic concepts, presents the important IRM request model with power law popularities, and then proceeds with more detailed models that capture time and space correlations.

2.1 Introduction to Caching-Related Terms

We begin with a few terms that will be useful across the entire monograph. A piece of reusable information is called a *content*. Examples of contents in the Internet include YouTube videos, Netflix movies, music files, Facebook pictures, HTTP web pages, documents, news feeds, etc. On the other hand, examples of non-reusable information include VoIP calls, teleconference, sms, and control signaling which are not relevant to caching. Fortunately, reusable contents account approximately for the 80% of total Internet traffic [60]. This makes caching extremely relevant for communication networks and content delivery platforms.

Definition 2.1 (Content Catalog). We consider a set of contents $\mathcal{N} = \{1, 2, \dots, N\}$ called the catalog, with size $N = |\mathcal{N}|$.

We will also be using the term *file* instead of content. For simplicity we assume that all contents are of equal size, unless otherwise

Table 2.1: Catalog and cache sizes (TB) of popular applications (retrieved in 2019)

Application	Catalog Size N	Cache Size M	$\gamma = M/N$
Small CDN	10	1	0.1
Netflix	314	>200	0.63
P2P file sharing	1.5×10^3	40	2.6×10^{-2}
Youtube	3×10^5	>200	0.5×10^{-3}
Facebook (photos)	1.5×10^3	300	0.2
Facebook (total)	3×10^5	300	10^{-3}
The Internet	3×10^8	—	—

stated.¹ Sometimes we measure the catalog size in Bytes (B), with the understanding that N can be recovered if we divide with the file size. The catalog and its size depend on the application. For instance, the catalog of videos in a Video-on-Demand (VoD) application is in the order of TBs (10^{12} Bs), while it is estimated that the total catalog of the Internet has surpassed 10^{24} Bs [113, 187].

Due to memory limitations, a cache can typically store only a subset of the catalog $\mathcal{M} \subset \mathcal{N}$, and we denote with $M = |\mathcal{M}|$ the total number of contents that fit in the cache. The *relative cache size* $\gamma = M/N \in [0, 1]$ is the fraction of catalog that can be cached, and its value affects significantly the cache performance. Ideally, we would prefer γ to be as large as possible, since with $\gamma = 1$ we can cache the entire catalog and thus satisfy all requests.

Nonetheless, parameter γ is very small in most applications. For example, YouTube’s catalog grows by roughly 30 PB/year, and the standing estimate is 300 PB. On the other hand, the Global Google Cache system — used to cache YouTube videos — is based on disk arrays with estimated size 200 TB. The pertinent figures for Facebook catalog,² Facebook photos [116], and Netflix³ are shown in Table 2.1. We see that the cache size is in most cases much smaller than the catalog. Nevertheless, even a small cache may be useful since some contents are

¹Capturing varying sizes in the analysis is possible and interesting, see [45, 86]; and we present some examples in Section 4.

²<https://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/>.

³<https://medium.com/netflix-techblog/content-popularity-for-open-connect-b86d56f613b>.

very popular and thus more frequently requested than others. To shed more light into this possibility, this section studies the properties of content popularity that is defined as follows.

Definition 2.2 (Content Popularity). Consider the probability distribution $\mathbf{p} = (p_n, n \in \mathcal{N})$ over the set $\mathcal{N}$, where p_n expresses how likely it is that a request is issued for content n . The probability p_n is called the popularity of content n .

2.2 Power Law Popularity

In this subsection we ask the question: what is a “good model” for the distribution $\mathbf{p}$? Where a good model should capture reality and facilitate mathematical analysis and optimization.

2.2.1 Zipf’s Law

The frequency of words appearing in text is a well-studied topic in the field of statistical linguistics [166]. Zipf’s law specifies that given a text of natural language, the probability we encounter a word is inversely proportional to its rank in the frequency table. We write:

$$p_n \propto n^{-1}, \quad n = 1, \dots, N,$$

which means that p_1 is the popularity of the most frequent word, p_2 that of the second most frequent, and so on; and the Zipf law governs the relation between all popularities, i.e., the distribution $\mathbf{p}$. For example, the popularity ratio of the most frequent word “the” over the second most frequent “be” is exactly $p_1/p_2 = 2$.

The reason why power laws appear in natural systems has been a topic of extensive study. Links have been made to: mixtures of exponential distributions [78], scale-invariance [53] and invariance under aggregation [78], random division of elements into groups [15], and others mentioned in [194]. Sometimes the connection between the rules of the underlying system and the formation of power laws remains unexplained [172, 194]. However, the frequency with which power laws appear in experimental data is too striking to be ignored. Thus, it is not surprising that Zipf models are used to emulate content requests

and perform mathematical optimization. Besides, as statistician George Box once said, “All models are wrong, but some models are useful” [42].

2.2.2 Evidence of Zipf’s Law in the Internet

Extensive traffic measurements have revealed that the popularity of requests for Internet content (web pages, video, etc.) exhibit a Zipf behavior [43]. A significant difference from word frequencies in texts is that the content popularity $\mathbf{p}$ is not always inversely proportional to rank, but a different exponent may be observed: the elements of $\mathbf{p}$ are $p_n \propto n^{-\tau}$, where τ sets the rate of popularity decline. For $\tau \rightarrow 0$, we recover the discrete uniform distribution, where all contents are equally popular. For $\tau = 1$, we recover the Zipf’s law. Taking the logarithm in both sides produces $\log p_n \propto -\tau \log n$, hence a log–log popularity/rank plot is a line with slope equal to $-\tau$. Larger τ values yield steeper slopes, and thus more skewed distributions. One can observe evidence of power law in a dataset by looking at the log–log plot of the observed empirical frequencies, as in Figure 2.1.

Since $\mathbf{p}$ is a probability distribution, the power law distribution is:

$$p_n = \frac{n^{-\tau}}{\sum_{j=1}^N j^{-\tau}}, \quad n = 1, \dots, N, \quad (2.1)$$

where the denominator is the truncated zeta function $H_\tau(N) \triangleq \sum_{j=1}^N j^{-\tau}$ evaluated at τ , also called the N th τ -order generalized harmonic number. Although this sum does not have a closed form, we provide below a useful approximation.

Approximation of generalized harmonic number. For $n \geq m \geq 0$ we bound $\sum_{j=m}^n j^{-\tau} = H_\tau(n) - H_\tau(m)$ using integrals:

$$\begin{aligned} \int_m^n (x+1)^{-\tau} dx &\leq H_\tau(n) - H_\tau(m) \leq 1 + \int_{m+1}^n x^{-\tau} dx, \Rightarrow \\ \left\{ \begin{array}{ll} \frac{(n+1)^{1-\tau} - (m+1)^{1-\tau}}{1-\tau} \leq H_\tau(n) - H_\tau(m) \leq \frac{n^{1-\tau} - (m+1)^{1-\tau}}{1-\tau} + 1, & \text{if } \tau \neq 1, \\ \ln \frac{n+1}{m+1} \leq H_\tau(n) - H_\tau(m) \leq \ln \frac{n+1}{m+2}, & \text{if } \tau = 1. \end{array} \right. \end{aligned}$$

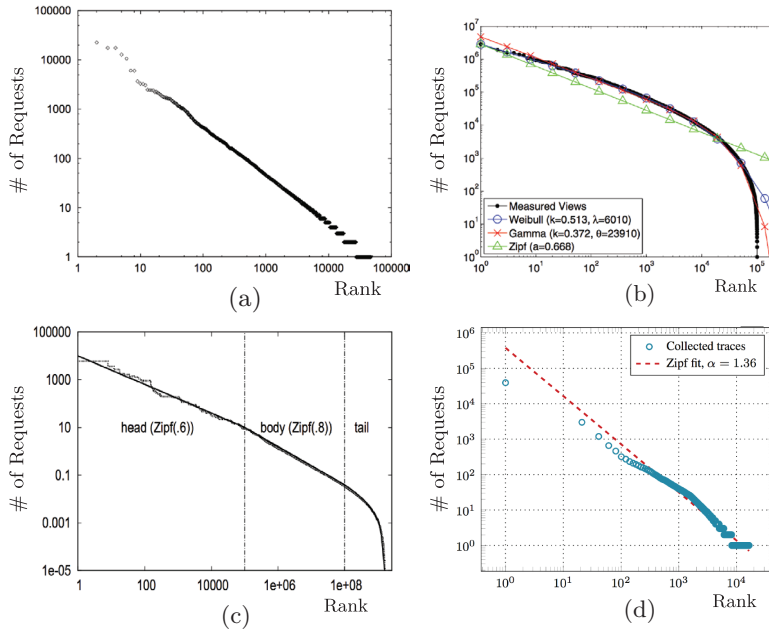


Figure 2.1: Datasets with power-law popularity. Log-log plots of number of requests versus content rank, showing evidence of power law popularity in: (a) WWW documents [43] (1999); (b) YouTube videos [56] (2008); (c) P2P torrents [206] (2013); and (d) HTTP requests in a cellular network [18] (2015).

Also, setting $m = 0, n = N$ we obtain:

$$\frac{(N+1)^{1-\tau} - 1}{1-\tau} \leq H_\tau(N) \leq \frac{N^{1-\tau} - 1}{1-\tau} + 1, \quad \text{if } \tau \neq 1. \quad (2.2)$$

If N is large, we then have the following approximation:

$$H_\tau(N) \approx \begin{cases} \frac{N^{1-\tau}}{1-\tau}, & \text{if } \tau < 1, \\ \log N, & \text{if } \tau = 1, \\ \frac{1}{\tau-1}, & \text{if } \tau > 1. \end{cases} \quad (2.3)$$

The approximation error is very small for all τ and large N , and is larger near $\tau = 1$; also, as N increases this error diminishes, Figure 2.2.

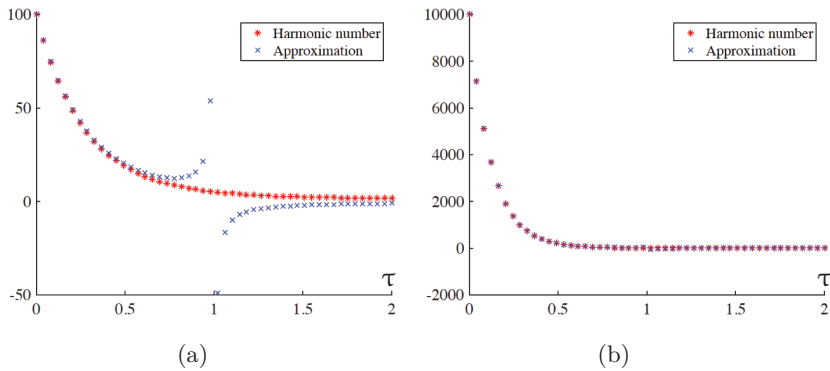


Figure 2.2: Examples of approximation (2.3). Plots of the approximation versus the actual generalized harmonic number, with parameters: $\gamma = 0.1$ and (a) $M = 10^2, N = 10^3$; (b) $M = 10^4, N = 10^5$.

2.2.3 Hit Probability Maximization

We present here a toy single cache optimization problem and its solution, in order to understand how the popularity model affects the analysis of caching systems. We draw one request for a content randomly from the catalog, and define the hit probability as follows.

Definition 2.3 (Hit Probability). The random event A = “the requested content is cached” is called a hit. The probability of A is called hit probability.

The hit probability is indicative of the fraction of requests that is served by the cache. Since we would like to serve as many requests from the cache as possible, one way to optimize the system performance is to decide which contents should be cached in order to maximize hit probability. This procedure is described next.

We introduce decision variables $\mathbf{y} = (y_n \in \{0, 1\}: n = 1, \dots, N)$, where $y_n = 1$ if we decide to cache content n , and $y_n = 0$ otherwise. Due to the limited cache size $M < N$, a *feasible* decision must satisfy the cache size constraint $\sum_{n=1}^N y_n \leq M$. Conditioning on the request of content n , the hit probability is computed by $h(\mathbf{y}, M) = \sum_{n=1}^N y_n p_n$,

when the decision $\mathbf{y}$ is taken. Overall, we are interested in the question: which feasible decision $\mathbf{y}$ will maximize hit probability? This is answered by solving the following optimization problem.

Hit Probability Maximization (HPM)

$$\begin{aligned} h(\mathbf{y}^*, M) = \max_{\mathbf{y} \in \{0,1\}^N} \quad & \sum_{n=1}^N y_n p_n \\ \text{s.t.} \quad & \sum_{n=1}^N y_n \leq M. \end{aligned}$$

We may solve (HPM) by inspection: set $y_n = 1$ for the M contents with greatest probability, and zero otherwise. This solution is also known as “cache the most popular” contents. Assuming w.l.o.g. that the files are ordered in non-increasing popularity, i.e., $p_1 \geq \dots \geq p_N$, the maximum hit probability is $h(\mathbf{y}^*, M) = \sum_{n=1}^M p_n$. When popularities are power-law with exponent τ , it is:

$$h(\mathbf{y}^*, M) = \sum_{n=1}^M p_n = \sum_{n=1}^M \frac{n^{-\tau}}{\sum_{j=1}^N j^{-\tau}} = \frac{\sum_{n=1}^M n^{-\tau}}{\sum_{n=1}^N n^{-\tau}} = \frac{H_\tau(M)}{H_\tau(N)},$$

and we may employ the approximation of $H_\tau(\cdot)$ in (2.3) to obtain an approximation of the maximum hit probability:

$$h(\mathbf{y}^*, M) \approx \begin{cases} \left(\frac{M}{N}\right)^{1-\tau} = \gamma^{1-\tau} & \text{if } 0 \leq \tau < 1 \\ \frac{\ln M}{\ln N} = 1 - \frac{\ln \gamma^{-1}}{\ln N} & \text{if } \tau = 1 \\ 1 & \text{if } \tau > 1. \end{cases} \quad (2.4)$$

Figure 2.3 shows the maximum hit probability comparing the actual value (found numerically by evaluating the sums) with the closed-form approximation of (2.4). We see that the approximation is very accurate, while the error introduced when $\tau \approx 1$ decreases as M or N increase.

Observe that in the typical case $\tau < 1$, the maximum hit probability can be approximated by the expression $\gamma^{1-\tau}$. The cache performance depends on the relative cache size γ and the power law exponent τ . We discuss next how these parameters can be estimated in real systems.

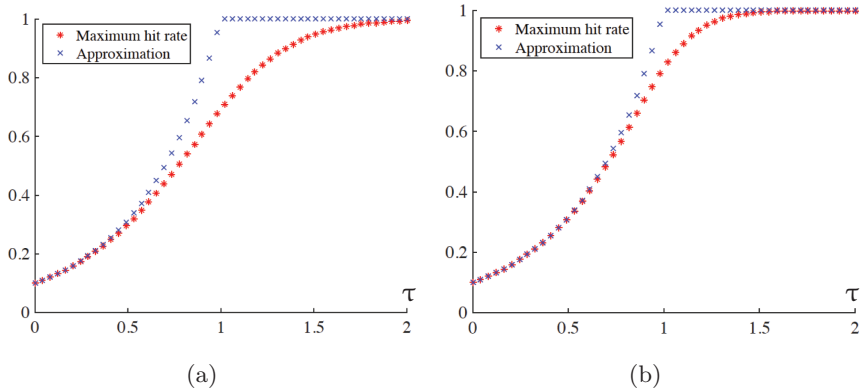


Figure 2.3: Examples of approximation (2.4). Approximation versus actual value of maximum hit probability, with $\gamma = 0.1$, (a) $M = 100$; (b) $M = 10000$.

2.2.4 Fitting the Power Law Model to Data

One of the most important design problems in caching is the *cache dimensioning*, where we must decide the cache size M for a specific system. For given storage and data transfer costs, and expected traffic, the cache dimensioning problem aims to tune the cache size for cost minimization. In the example of one cache, this can be done via $h(\mathbf{y}^*, M)$ which provides an estimate of the cache efficiency for a certain value of M . As we saw above, this estimation requires knowledge of the Zipf law exponent. Namely, we would like to determine the value τ^* that reproduces accurately the observed system behavior in terms of requests. Inferring the parameters of a distribution from data is known as *statistical inference*, and in this subsection we focus on highlighting some subtle points that appear in the inference of the power law exponent.

In practice, engineers sample a dataset with requests from the said system during peak-demand and then analyze it statistically. We discuss here how τ^* can be extracted from such a dataset. Throughout the subsection we assume that the samples are drawn independently from a distribution $\mathbf{p} = (p_n \propto n^{-\tau}, n \in \mathcal{N})$ with unknown τ , and we explain how to compute the parameter τ_{MLE}^* that best fits the samples in a maximum likelihood sense.

A few remarks are in order. First, we note that in general the popularity distribution changes over time. Our assumption of static distribution is meaningful only when the dataset is collected in a reasonably small time interval. If we knew there are time correlations, we could use this knowledge to improve our estimate; this approach is explained in Subsection 2.3.2. Second, in practice it is common to fit data to a power law using a graphical method, where one simply plots the log-log requests versus rank and sets a value for τ that approximates the curve. Unfortunately, such graphical analysis can be grossly erroneous [62, 100], and wrong estimates for τ may lead to wrong decisions about cache dimensioning. For this reason, we present a rigorous approach of fitting the data to a power law by means of maximizing the log-likelihood function [100]. We mention that similar conclusions can be reached using the linear least squares method, albeit at a higher error [22].

We begin with a dataset $X = (x_i \in \mathcal{N}: i = 1, \dots, K)$, where the i th datum x_i is the id of one of the contents in catalog; for instance we have in mind the YouTube dataset from [251]. For a single datum x_i we denote with $\ell(x_i; \tau)$ the likelihood that x_i is generated from a power law model with exponent τ , and

$$\ell(x_i; \tau) = n^{-\tau} / H_\tau(N), \quad \text{iff } x_i = n.$$

The likelihood function is multiplicative over the data in the dataset. And for X with K samples, we can write:

$$\ell(X; \tau) = \prod_{i=1}^K \ell(x_i; \tau).$$

Definition 2.4 (Empirical Frequency). For a sequence of requests, the empirical frequency of content n is the ratio:

$$f_n = \frac{\text{Number of requests for content } n}{\text{Total number of requests}}.$$

Disregarding the constant of total number of requests, we can write:

$$\ell(X; \tau) = p_1^{f_1} p_2^{f_2} \dots p_N^{f_N}, \quad \text{where } p_n = n^{-\tau} / H_\tau(N).$$

The log-likelihood function $\Lambda(\tau) = \log \ell(X; \tau)$ is:

$$\begin{aligned} \Lambda(\tau) &= \sum_{n=1}^N f_n \log(n^{-\tau}/H_{\tau}(N)) = \sum_{n=1}^N f_n (\log n^{-\tau} - \log H_{\tau}(N)) \\ &= -\tau \sum_{n=1}^N f_n \log n - \log H_{\tau}(N). \end{aligned}$$

The log-likelihood function is preferred from likelihood because it simplifies the calculations. Since the logarithm is a strictly increasing function, the maximization of either function leads to the same solution. The MLE we are seeking can be recovered by solving the optimization problem:

$$\tau_{\text{MLE}}^* = \arg \max_{\tau \geq 0} \Lambda(\tau). \quad (2.5)$$

The maximum likelihood estimator is *efficient*, meaning that as the sample size increases, $K \rightarrow \infty$, it achieves the Cramér–Rao bound [203], and therefore has the smallest mean squared error among all unbiased estimators. Problem (2.5) can be solved in closed-form using the approximation (2.3) and computing the stationary point of $\Lambda(\tau)$. Alternatively, we may solve it numerically. Figure 2.4 presents the results of the described method applied to the YouTube dataset of [251].

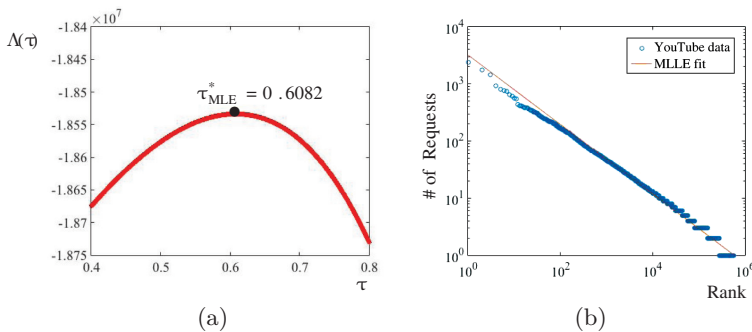


Figure 2.4: Example of power-law fitting with MLE. Experiments on the 2008 YouTube dataset [251]: (a) log-likelihood function $\Lambda(\tau)$; (b) log–log plot of ranked requests and MLE power-law vs rank.

Unknown Labels

A typical complication that arises in caching is the unavailability of content labels. Here, we use the term label to refer to the actual index of a content in the underlying popularity distribution $\mathbf{p}$. For instance, the content with popularity p_n has label n . When working with real traces we observe how many times a specific content is requested, but its actual label remains unknown. The content with label 100 and popularity p_{100} might appear fewer times than the content with label 99 and popularity p_{99} , due to limited size of our sample. Therefore, although $p_{99} > p_{100}$, it may be that $f_{99} < f_{100}$. When the dataset is unlabeled with respect to the underlying popularity distribution one possibility is to adapt the maximum likelihood estimation as follows: (i) we rank the empirical frequencies in decreasing order such that $f_{\phi(1)} \geq \dots \geq f_{\phi(N)}$; (ii) we *assume* that the ranked frequency $f_{\phi(n)}$ corresponds to label n ; and (iii) we infer the power law exponent with the above MLE method. Nevertheless, the implicit assumption $\phi(n) = n$ does not always hold in practice, and so the known guarantees for maximum likelihood do not apply. We provide an example below.

Fitting unlabeled data with MLE. We create $1.46 \cdot 10^6$ samples from a power law distribution with exponent $\tau = 0.6082$, and catalog size $N = 566 \cdot 10^3$ (same parameters as in the YouTube dataset but fictitiously generated data). Then we extract from the samples three different empirical frequency vectors:

- (f_n) is ranked according to the actual labels of p_n ,
- (f'_n) is permuted so that $f'_1 \geq \dots \geq f'_N$, and
- (f''_n) is same as (f'_n) but limited to the first 10^3 elements (head of ranked frequencies).

The vectors are then used to produce the maximum log-likelihood estimates of the power law exponent; we find $\tau_{\text{MLE}}^1 = 0.6078$ (0.06% relative error), $\tau_{\text{MLE}}^2 = 0.6406$ (5.32% rel. error), $\tau_{\text{MLE}}^3 = 0.6050$ (0.52% rel. error). Figure 2.5 shows log-log requests vs rank plots in all cases.

While the labeled MLE has a very small relative error (0.06%) as expected, the example shows that the ranking of frequencies leads to

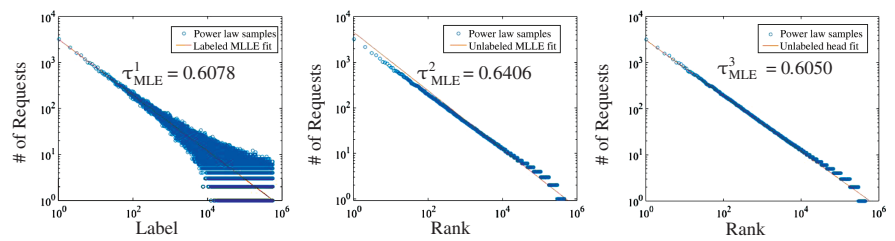


Figure 2.5: MLE fitting of unlabeled data. Generated power law data with $\tau = 0.6082$, catalog $N = 566 \cdot 10^3$ and $1.46 \cdot 10^6$ samples. (left) MLE fit with the correct labels, (middle) MLE fit with ranked frequencies, (right) MLE fit of the head with ranked frequencies.

erroneous estimation (5.32% error). This is due to the permutation of the labels at the distribution tail. We can improve the accuracy if we restrict the fitting to the head of the ranked empirical frequencies (0.52% error), an approach known as “trust your body” in [174]. Indeed, while the tail is very noisy, the head and body of the labeled frequencies and the ranked frequencies are similar. A different approach to inference of unlabeled data is given in [183], which considers the popularity of each file as a random variable with the same power-law distribution.

Observed Values of τ

Table 2.2 lists publicly available datasets for experimentation with request sequences and power-law popularity, and we summarize available measurements of τ in Table 2.3. Evidently, the power law model is fitted

Table 2.2: Publicly available datasets for caching experimentation

Dataset	Ref.	Number of Requests	Time Span
Web proxy accesses	[45]	24 M accesses	8/96–9/96
Netflix movie ratings	[29]	100 M movie ratings	10/98–12/05
Anonymized URL requests	[170]	25 B requests	9/06–3/08
YouTube traces, UMass	[251]	1.5 M requests	6/07–3/08
Top-1000 Wikipedia pages	[241]	34 M requests	10/15–11/15

Table 2.3: List of τ values from datasets

Reference	Value of τ	Year	Application
[66]	1	1995	Web traffic
[43]	0.64–0.83	1999	Web traffic
[205]	0.64–0.91	1999	Web traffic (univ. proxy)
[186]	1.39–1.81	2000	Web traffic (MSNBC news)
[8]	1.16	2000	Web traffic (1998 world cup)
[165]	0.74–0.84	2000	Web traffic
[3]	1	2002	Web traffic (AOL)
[59]	0.58–0.64	2003	P2P (Gnutella & Napster)
[49]	1	2004	Web pages (Japanese)
[240]	1	2004	P2P (FastTrack)
[138]	0.92–1.09	2006	Web traffic (Russian)
[244]	1–1.5	2006	Mobile web
[247]	0.6	2006	VoD (China Telecom)
[94]	0.56	2007	Video (YouTube)
[57]	0.54	2007	Video (YouTube)
[112]	0.6–0.78	2008	P2P
[230]	0.53	2009	Wikipedia
[131]	0.85	2009	Video (Yahoo)
[69]	0.6–0.86	2010	P2P (BitTorrent)
[228]	0.7–0.85	2012	Video (YouTube)
[116]	0.7–1	2013	Photos (Facebook)
[117]	0.83	2014	Web traffic (French)
[253]	1.09–1.87	2015	Web traffic (Spanish)
[18]	1.36	2015	Mobile web (Turkish)
[109]	0.5–0.75	2015	Wikipedia

in many different scenarios over the years, but each application (or, even each dataset) fits a different value of τ . Therefore, our mathematical analysis should be parametric to the power law exponent, and adjustable to the dataset at hand. An important question is: what should be our choice for τ when data are not available? For visualizing results, it is customary to take $\tau \in \{0.6, 0.8, 1\}$, which are the most depictive values.

2.2.5 Catalog Estimation

The problem of cache dimensioning requires additionally the knowledge of catalog size N . While estimation of τ is commonly encountered in

the literature, the catalog size estimation is often overlooked.⁴ For a dataset X with K samples, the number of different observed contents is called the *visible catalog* and has size:

$$N_K = |\{n \in \mathcal{N} \mid f_n(X) > 0\}|$$

which clearly depends on X and its length. Using this notation, the true catalog size is N_∞ , and we have $N_\infty \geq N_K$.

Let E_j be the number of contents observed exactly j times in X , and let e_j denote its mean. We have:

$$E_j = \sum_{n=1}^{N_K} \mathbb{1}_{\{\text{content } n \text{ is requested } j \text{ times}\}},$$

with $\mathbb{1}_{\{\alpha\}}$ being the indicator of event α . Taking the expectations, yields:

$$\begin{aligned} e_j(X) &= \sum_{n=1}^{N_K} \Pr[\text{content } n \text{ is requested } j \text{ times}] \\ &\stackrel{(a)}{=} \sum_{n=1}^{N_K} \binom{K}{j} p_n^j (1 - p_n)^{K-j}, \end{aligned}$$

where in (a) we have assumed that the samples are i.i.d. A mean estimator $\hat{N}_\infty$ of the actual catalog N_∞ can be thus obtained by

$$\hat{N}_\infty = N_K + e_0(X),$$

where recall that N_K is the size of the visible catalog and therefore easy to find by enumerating the unique ids in the dataset, and e_0 is the mean number of contents that are never requested:

$$e_0(X) = \sum_{n=1}^{N_K} \binom{K}{0} p_n^0 (1 - p_n)^{K-0} = \sum_{n=1}^{N_K} (1 - p_n)^K.$$

In summary, combining the calculation of $e_0(X)$ with the observable N_K , one can obtain an unbiased estimate of the true catalog size $\hat{N}_\infty$, which subsequently determines our estimate of γ and allows for correct cache dimensioning via $h(\mathbf{y}^*, M)$.

⁴We would like to acknowledge discussions with Dr. J. Roberts for this subsection.

2.3 Request Sequences

So far we have considered a single isolated request, drawn randomly from distribution $\mathbf{p}$. This allowed us to discuss content popularity and introduce the optimization of hit probability. In this subsection, we introduce the concept of *request sequence*: a succession of content requests that arrive at the cache. Based on that, we investigate more intricate aspects of content popularity, such as temporal and spatial correlations.

Definition 2.5 (Request Sequence). The request sequence $\mathbf{R} = (R_1, \dots, R_T)$ is a sequence of integers, where $R_t \in \mathcal{N}$ denotes the content id of the t th request.

We present next different models of request sequences.

2.3.1 Independent Reference Model

Definition 2.6 (IRM Model). A request sequence $\mathbf{R}$ is called IRM if $R_t, \forall t$ are i.i.d. random variables.

The term “independent” reflects the property of an IRM sequence that each random variable is drawn with distribution $\mathbf{p}$ independently of all others. The term “reference” is identical to the term “request” in our context. As a sequence of i.i.d. random variables, IRM represents a special case where each element of the sequence repeats the same random experiment, in which case the definition of hit probability continues to apply. In the next section we provide the hit probability analysis for several online caching policies under IRM.

Sometimes we define the request sequence in continuous time. The simplest extension in this direction is to consider a Poisson process whose points denote the requests. Each point of the Poisson process is *marked* with a value from $\{1, \dots, N\}$ denoting which content is requested. The marks are chosen randomly and i.i.d. from distribution $\mathbf{p}$.

Definition 2.7 (Poisson IRM Model). A request sequence is called Poisson IRM if it is determined by the marks of an associated homogeneous Poisson process with intensity λ and random i.i.d. marks drawn from $\mathbf{p}$.

The intensity parameter λ defines how densely the different requests are spaced apart in time. We remark that the standard IRM model can be obtained from the Poisson procession by discarding the inter-request time information and keeping only the (ordered) marks at the points of the process. When restricting attention to the marks, we may still use the notion of hit probability. For the associated continuous time process, we will later define the notion of *hit rate*, see discussion in Subsection 3.2.

IRM sequences are the simplest and most established models. They are often combined with a power law distribution, thus yielding a one-parameter model (or two-parameter in case of a Poisson IRM), which is easy to fit. Previous work validated IRM models, showing that they fit well real request traces obtained at peak-demand hour [43]. On the other hand, IRM sequences by design have no temporal or spatial correlations. This means that if the actual data do exhibit such correlations, an IRM model will not capture them. We discuss next alternative models that can handle temporal and spatial correlations.

2.3.2 Temporal Locality

Several measurement studies have shown that content popularity exhibits temporal correlations. Specifically, it has been shown that the popularity of recent requests is higher than the popularity of past requests, when these have the same empirical frequency. This phenomenon is called “temporal locality” and it is particularly important for caching. Namely, it suggests that we should not cache contents that are popular on average, but rather those being popular locally in time.

Evidence of temporal locality dates back to web page requests in the 90s. For example, [6] attributed to temporal locality the hit probability discrepancy between dataset and generated data from a distribution with the same power-law exponent; [43] observed that the popularity of highly popular contents changes less often than that of the unpopular ones; and [228] explained how decaying popularity in YouTube videos affects cache hits. In sum, content popularity is often time-varying, and this has a profound effect on hit probability and cache dimensioning.

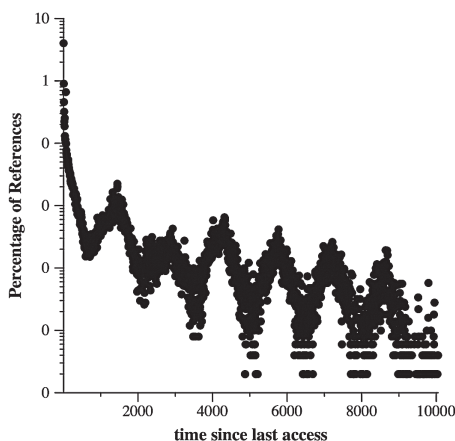


Figure 2.6: Example of temporal locality. Percentage of references as a function of the time since last access to a document by the same user, from [45]. The time is in minutes and y -axis is in log scale.

Regarding the span of temporal locality, a YouTube study from [226] confirms that correlations on timescale of few hours can be ignored, as their impact on cache performance is negligible. Instead, the timescale of few days up to weeks is deemed most significant. For example, Figure 2.6 shows how the web proxy accesses evolve over time, and it is indicative of strong ephemerality. Recent accesses are 10^2 times more probable than those requested one day ago, and 10^3 times than those requested one week ago. Similarly, [109] demonstrated that requests of Wikipedia monographs have a rapid day-to-day change in popularity rank. In particular, measuring the fraction of contents in top- x popularities of a day and the previous, they find 76% for $x = 10^3$ and only 54% for $x = 25$, meaning that half of the top 25 contents change from day to day. In conclusion, a caching analysis that extends beyond a few hours must take into account temporal correlations. This finding is particularly important for systems with few requests per unit time, where learning the evolving popularity might be very challenging.

In order to model temporal locality, we consider request models with time-varying popularity distribution $\mathbf{p}(t) = (p_n(t), n \in \mathcal{N})$. When designing such models we aim to fit accurately the observed correlations while ensuring mathematical tractability. There exist several

such models, e.g.: [99] uses the theory of variations, [75] makes random replacements of contents in the catalog, and [182] uses a dependent Poisson model. We elaborate next on the Shot Noise Model (SNM) [228], which is similar to the one independently proposed in [182].

The Shot Noise Model

The main idea in SNM is to define the request sequence as a superposition of many independent *inhomogeneous* Poisson processes called “shots”, where each shot is associated to an individual content and describes the temporal profile of its popularity [228]. This is a natural generalization of the Poisson IRM model in the following sense. Recall that the latter is represented by a homogeneous Poisson process with intensity λ , where a Poisson point corresponds to a request for content n with probability p_n . Using properties of the Poisson process, this can be shown to be equivalent to superposing N thinned homogeneous Poisson processes, one for each content, where the process for n has intensity λp_n . In SNM, we replace each thinned homogeneous Poisson process with an inhomogeneous Poisson process of a time-varying intensity $\lambda_n(t)$, like the one in Figure 2.7.

If the shape of all shots is horizontal $\lambda_n(t) = \lambda p_n, \forall n$ then we recover the Poisson IRM. Typically though, we use a time-varying shape to reflect the changing nature of popularity. The temporal locality phenomenon can be modeled using a rapid growth towards maximum popularity (highest intensity) followed by a phase of slow decrease [249],

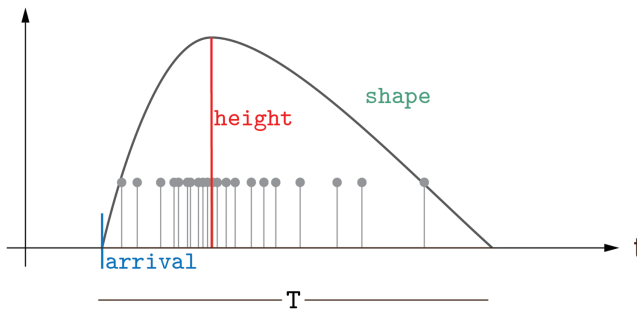


Figure 2.7: Characteristics of a shot in SNM.

similar to Figure 2.7. In general, the shot is characterized by its: (i) shape; (ii) duration; (iii) arrival instance; and (iv) height.

It is reported that different choices of (i)–(ii) result in similar hit probability performance [228], therefore as a general recipe we may fix the shape to rectangular pulses and the duration to a fixed T in order to maintain tractability. The shot arrival instances (feature (iii)) are points of another Poisson process with constant rate ν . Denote with t_n the arrival time of shot n . At time t the active content catalog is:

$$\mathcal{N}(t) = \{n: t - T \leq t_n \leq t\},$$

and the expected size of the catalog at any t will be $\mathbb{E}[|\mathcal{N}(t)|] = \nu T$. Hence, parameter ν is used to tune the model to the estimated catalog. The shot height (iv) is often taken to be power-law distributed, which ensures a good fit to the power law model when we inspect the instantaneous popularity. Specifically, we set the height of n , while it is alive, to the random variable p_n constructed in the following way. First, for any n , let ξ_n be an i.i.d. random variable drawn uniformly from $[0, 1]$. Then

$$p_n = \frac{\xi_n^{-\tau}}{\int_0^1 z^{-\tau} dz} \bar{p} = \xi_n^{-\tau} (1 - \tau) \bar{p}, \quad \text{for all } n,$$

where $\bar{p}$ is the mean popularity over all contents, and τ the power law exponent. Let $f(x)$ denote the density of p_n at $x \in [\bar{p}(1 - \tau), +\infty)$;⁵ $f(x)$ is in fact a Pareto distribution with $\alpha_{\text{Par}} = 1/\tau$ and $x_{\text{Par}} = \bar{p}(1 - \tau)$ [179]. Note that Pareto is the limit of Zipf distributions for large catalogs.

Definition 2.8 (SNM Model). The SNM is the superposition of N Poisson inhomogeneous processes, where process $n = 1, \dots, N$ corresponds to content n and has a time-varying intensity $\lambda_n(t)$, characterized by shape, duration, arrival instance determined by a Poisson point process with intensity ν and height p_n Pareto distributed. Furthermore, a rectangular-SNM has rectangular shape and fixed duration T .

[228] employed an SNM with 4 shot shapes (all with rectangular shape but different shot duration) to accurately fit a dataset with

⁵Note that the density is not defined for smaller values of x .

YouTube video requests. In parallel, [182] observed that the shots duration is correlated with the height: highly popular contents tend to stay alive longer. This result implies that the SNM model accuracy can be improved if we correlate the duration T_n of file n to its shot height.

Classifying Contents Based on Instantaneous Popularity

When content popularity changes over time, we must follow a different approach than solving (HPM). The focus is now on the instantaneous hit probability, which is random, and hence we wish to maximize its expectation. An equivalent — and instructive — approach is to formulate the maximization of instantaneous hit probability as a binary classification task. Given a cache with relative size γ , we would like to observe a request sequence up to t and classify each content into the $\gamma N(t)$ most popular contents (or not), where recall that $N(t)$ is the number of “alive” contents. The classification is performed by means of solving the following optimization problem.

Popularity Classification (PC)

$$\begin{aligned} \min_{\mathbf{y} \in \{0,1\}^N} \quad & \sum_{n=1}^N y_n \mathbb{E}[p_n(t) - \hat{p}_n(t)] \\ \text{s.t.} \quad & \sum_{n=1}^N y_n = \gamma N(t). \end{aligned}$$

Parameter $p_n(t)$ is the instantaneous popularity of n at time t , and $\hat{p}_n(t)$ is our respective estimation. Hence, the objective function expresses the difference between the maximum possible hit probability and the one we would obtain by caching the most popular contents according to our estimation. The constraint ensures that we only classify $\gamma N(t)$ contents as cacheable. Note that $\sum_{n=1}^N y_n \mathbb{E}[p_n(t)]$ is in fact equal to $\bar{p} \gamma N(t)$, thus fixed and independent of the optimization variable y_n . Furthermore, the solution $\mathbf{y}^*$ of problem (PC) is obtained as before by ordering the contents in decreasing estimated popularity $\mathbb{E}[\hat{p}_n(t)]$ and then setting $y_n^* = 1$ for the top $\gamma N(t)$ of them. Therefore, below we will

focus on finding a good popularity estimate $\hat{p}_n(t)$ in order to decrease the hit rate loss $\sum_{n=1}^N y_n^* \mathbb{E}[p_n(t) - \hat{p}_n(t)]$.

Hereinafter, we consider the example of rectangular-SNM model. To arrive at a popularity estimate, we assume that at time t we know the total number of requests for content n up to t , $K_n(t) = \sum_{i=1}^t \mathbb{1}_{\{R_i=n\}}$; the time t_n of the first request for content n , from which we can infer the age of the content⁶ $a_n(t) = t - t_n$; and the model of p_n which is power law if n active, and zero if inactive. Accordingly, $\hat{p}_n(t)$ is the popularity estimate based on measurements $(K_n(t), a_n(t))$. It is tempting to use as estimates the frequencies $f_n(t) = K_n(t)/a_n(t)$. However, such a frequentist approach would disregard the prior model. Instead, we follow the Bayesian approach and use the *a posteriori* estimate of the instantaneous popularity of n at time t , given by:

$$\hat{p}_n \triangleq \mathbb{E}[p_n \mid K_n, a_n] = \frac{\int_{p_n} p_n \Pr[K_n \mid p_n, a_n] f(p_n) dp_n}{\int_{p_n} \Pr[K_n \mid p_n, a_n] f(p_n) dp_n}, \quad (2.6)$$

where for notation simplicity we omitted the dependence on t from all terms. The term $\Pr[K_n \mid p_n, a_n]$ is a Poisson probability, i.e.,

$$\Pr[K_n \mid p_n, t_n] = (p_n a_n)^{K_n} \frac{e^{-p_n a_n}}{K_n!},$$

and $f(x)$ is the Pareto probability density function.

Now let $F(x)$ be the distribution of $\hat{p}_n$ and θ be a threshold defined as follows: $\theta = F^{-1}(1 - \gamma)$, hence θ is in fact the γ -upper quantile of $F(x)$, or more simply θ is the smallest value for which $\Pr[p_n > \theta] = \gamma$ is true. We may also think of θ as a popularity threshold after which contents are classified as “popular”. Then we define the corresponding threshold on the number of observed requests as a function of age a :

$$\tilde{K}(a) = \min\{k = 0, 1, \dots \mid \mathbb{E}[p_n \mid K_n = k, a_n = a] \geq \theta\}, \quad (2.7)$$

From [144] we have the following result.

⁶This is defined as the time duration for which the content has been active.

Theorem 2.1 (Popularity classification for rectangular SNM): Consider a cache with relative size γ , and let $\tilde{K}(a)$ be the threshold defined in (2.7). Then if content n has K_n requests and age a_n classify it according to a posteriori estimate rule:

$$y_n^* = \begin{cases} 1 & \text{if } K_n \geq \tilde{K}(a_n) \\ 0 & \text{otherwise.} \end{cases}$$

As $\nu \rightarrow \infty$, y_n^* is a solution to (PC) with probability 1.

Proof. We focus on time instance t and drop all mention of t , and let $\mathcal{N}$ denote the set of alive shots. As $\nu \rightarrow \infty$, the mean number of alive shots $\mathbb{E}[|\mathcal{N}|] = \nu T$ scales to infinity as well, and the random variable $|\mathcal{N}| \rightarrow \infty$ w.p.1. Each alive shot's popularity is Pareto distributed, and therefore we are observing a scaling number of independent Pareto experiments. In the following, we use this fact to obtain the proof.

Recall that $\hat{p}_n \triangleq E[p_n | K_n, a_n]$ is the expected popularity of a shot conditioned on observing K_n requests in a_n time (and factoring the prior model of p_n which is Pareto). Then, consider the empirical distribution of $\hat{p}_n$ of all alive shots:

$$F^\nu(x) = \frac{1}{|\mathcal{N}|} \sum_{n=1}^{|\mathcal{N}|} \mathbb{1}_{\{\hat{p}_n < x\}}.$$

Since $|\mathcal{N}| \rightarrow \infty$ w.p.1, we have that $\lim_{\nu \rightarrow \infty} F^\nu(x) = F(x)$, where F is the distribution of $\hat{p}_n$. Let now θ^* denote the γ -upper quantile of F , such that $\Pr[\hat{p}_n \leq \theta^*] = 1 - \gamma$. This quantile is very convenient because it represents the threshold on $\hat{p}_n$ above which we should cache a shot in order to meet exactly the cache constraint of (PC). Indeed, we note that this is the rule with which we decide y_n^* . By the convergence of distributions, it follows that the optimal empirical threshold also converges to θ^* . As a result, classifying with y_n^* ensures that: (i) the cache constraint is exactly met (by the property of quantile); and (ii) the shots with $y_n^* = 1$ have higher $\hat{p}_n$ than those with $y_n^* = 0$, hence the objective of (PC) is minimized. $\square$

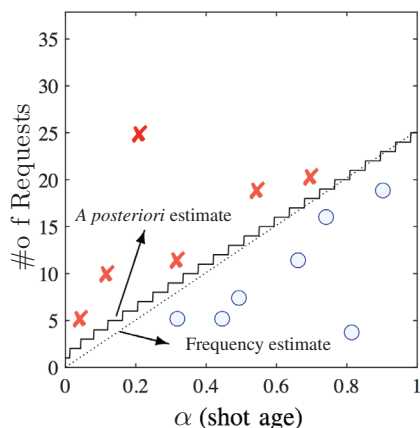


Figure 2.8: Comparison of classifiers. The *a posteriori* estimate (solid line) and the frequency estimate (dotted line).

In Figure 2.8 we compare the above (optimal) classifier for the rectangular-SNM to the frequency estimates, and we see that they are indeed different. The differences are greater in the general SNM, without the assumption of equal shot duration T .

To summarize, we have discussed the common phenomenon of temporal locality, which amounts to time correlations in the request sequence. Such correlations can be captured accurately by non-stationary models, such as the SNM. The latter, allows for a simple derivation of the optimal classifier (to popular/unpopular) by exploiting the knowledge of the prior model to outperform the standard frequency estimator. Future research is needed along these lines to solidify our understanding of what is a good non-stationary model, and derive Bayesian estimators for time-varying popularity distributions.

2.3.3 Spatial Locality

The requests exhibit correlations across space as well. For example, crossing the borders of two countries with different language has a severe impact on the content popularity. Furthermore, it has been found that the popularity distribution changes rapidly even within a city [213, 251]. And similarly, studying traffic volumes at base stations

in European cities, [88] showed that the traffic fluctuates based on a location-dependent pattern, called “signature”. Metro station areas, residential, commercial, touristic, academic, and other type of areas have their own distinctive signature. Similarly, it is believed that the content preference may vary depending on the culture and activity at each location. In a multi-cache installation spanning a large geographical area, such spatial correlations result in certain contents being very popular in some caches, and unpopular in the rest. This also suggests that the popularity model has location-dependent “features”, and clustering together locations with similar features can improve popularity learning.

We mention, however, that identifying spatial correlations in popularity is challenging since it requires collection of data at different points in the network, simultaneous availability of content id (application information) and location information (user-specific information), and a large number of data in a short time interval from a sizable geographical area. To the best of our knowledge, no such data are published to date. Our discussion is therefore restrained here to the proposition of mathematical models for generating spatially correlated requests, without the ability to validate the models with real data. We propose next the stochastic block model that captures the geographical correlations of content requests, and allows easy inference and optimization.

The Stochastic Block Model

In this subsection we present the *Stochastic Block Model* (SBM). We consider $V + 1$ caches, consisting of a set $\mathcal{V} = \{1, 2, \dots, V\}$ edge caches and a parent cache that we index with 0, Figure 2.9. The edge caches receive local requests from different geographical locations, while the parent cache receives unsatisfied requests from all areas. This is a typical model of an hierarchical caching network; further details and examples are presented in Section 4. Each cache $v \in \mathcal{V} \cup \{0\}$ “sees” a popularity distribution $\mathbf{p}_v = (p_{nv}, n \in \mathcal{N})$, that can be vary across the caches.

We propose a model for correlated local popularities. The global popularities $\mathbf{p}_0$ can follow any law, for example we may assume that they are described by an SNM with mean popularity $\bar{p}$. To model correlations

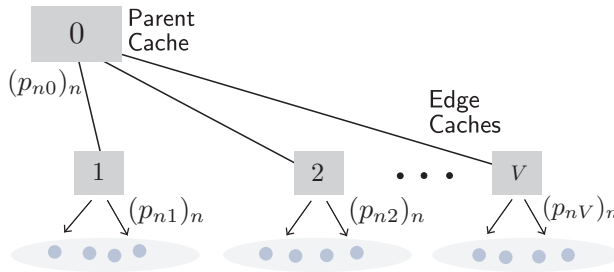


Figure 2.9: Stochastic block model. Modeling correlated popularities in a system with V edge caches and 1 parent cache. Each cache v receives requests that are drawn from a possibly different distribution probability $\mathbf{p}_v = (p_{nv}, n \in \mathcal{N})$.

between the local cache popularities, we draw inspiration from the field of community detection [147]:

- Each content n is associated with a feature vector (X_n) . To simplify the model we let (X_n) be independent uniform random variables, each taking values in $[0, 1]$.
- Each location v is associated with feature vector (Y_v) , which are again chosen independently and uniformly in $[0, 1]$.
- We define a kernel $K(x, y) = g(|x - y|)$, where g is a continuous function, strictly decreasing on $[0, 1/2]$, symmetric and 1-periodic, with $\int_{[0,1]} g(|x - y|) dy = 1$ for all $x \in [0, 1]$. Since the kernel K is periodic, we can think of the feature vectors $(X_n), (Y_v)$ as lying on the torus $[0, 1]$ rather than the interval.

The local popularity of content n at cache v is defined as

$$p_{nv} = p_{n0} \frac{K(X_n, Y_v)}{\sum_{v' \in V} K(X_n, Y_{v'})}, \quad \forall n, v = 1, \dots, V.$$

One can trivially verify that by averaging the popularity of locations $1, \dots, V$, we obtain p_{n0} (the popularity of location 0). As such, SBM is completely determined by the model of p_{n0} and the chosen kernel $K(x, y)$, where the latter is responsible for inducing the correlations. We can further simplify the model by choosing the kernel to have a

specific form, for example [144] uses:

$$K(x, y) = 5(1 - 2|x - y|)^4.$$

The SBM is an analytical model where popularity exhibits spatial correlations. In this setting, we note an inherent tradeoff of learning. The global location 0 aggregates the requests from all locations, and therefore learns quickly the average popularity distribution (p_{n0}), owing to the large number of request samples per unit time. On the other hand, each individual location learns slower (due to smaller sample), but it learns the local model (p_{nv}), which is more accurate than the average of all models (p_{n0}). An interim approach is also possible, where the distribution is learned from aggregating requests from a subset of locations with similar popularity. Such a cluster-based learning would be faster than local learning and more accurate than global learning. As an example, suppose half of the locations share the same popularity distribution, and they are negatively correlated with the other half such that $p_1 \geq p_2 \geq \dots \geq p_N$ in the first, and $p_1 \leq p_2 \leq \dots \leq p_N$ in the second. Then learning in two clusters would greatly outperform both global and local learning. This can be achieved with standard spectral clustering techniques from the literature, for example [235].

2.4 Discussion

In this section we surveyed the key concepts of content popularity, explained spatio-temporal correlations in the request process, and presented the most prominent and widely used content popularity models. Predicting a time-varying popularity is a key challenge in caching, and we provide more information in Subsection 3.4. In the next section we study policies that decide which content files to cache in reaction to an unknown (and potentially changing) popularity.

3

Cache Eviction Policies

More often than not, the cache is full and hence in order to store a new content, another one must be evicted to make room in the cache. This section focuses on cache eviction policies that decide which content should be evicted. Broadly speaking, we seek to find dynamic eviction policies that maximize the hit performance. In light of this, we present offline and online cache eviction policies with associated performance guarantees under various settings. We examine different assumptions for the underlying request sequence, emphasizing fundamental properties of caching policies, and providing intuition for designing new policies to match the needs of new systems.

3.1 Performance Under Arbitrary Requests

In this subsection we make no assumptions about requests, and analyze the performance of caching policies under an arbitrary request sequence.

3.1.1 The Paging Problem

Consider a catalog $\mathcal{N} = \{1, 2, \dots, N\}$ of N equal-sized contents, and a cache that fits at most M of them. The time is slotted, $t = 1, 2, \dots, T$,

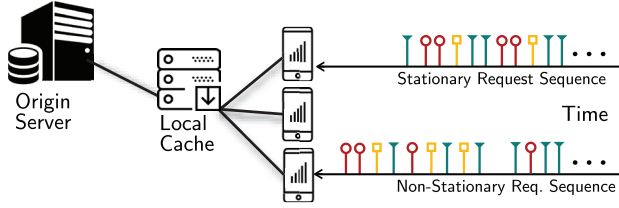


Figure 3.1: Eviction policies. Content requests must be served by a local cache (hit) or the origin server (miss). An eviction policy observes past requests and decides which contents to cache in order to increase the hit probability (or hit rate).

and there is a request sequence $\mathbf{R} = (R_1, R_2, \dots, R_T)$, where request $R_t \in \mathcal{N}$ arrives at the cache in the beginning of slot t , see Figure 3.1. The cached files at slot t are denoted with the set $\mathcal{M}_t \subseteq \mathcal{N}$, and there is a hit if $R_t \in \mathcal{M}_t$ and a miss if $R_t \notin \mathcal{M}_t$. In case of a miss, we typically add the newly requested content to the cache, and when $|\mathcal{M}_t \cup R_t| > M$ we *evict* one of the cached files, denoted E_t . The evolution of the *cache state* is then:

$$\mathcal{M}_{t+1} = \begin{cases} \mathcal{M}_t \cup R_t - E_t & \text{if we have a miss} \\ \mathcal{M}_t & \text{if we have a hit} \end{cases} \quad t = 1, \dots, T-1.$$

Definition 3.1 (Eviction Policy). The online eviction policy π is a rule that at slot t maps the cache state $\mathcal{M}_t$ and the request sequence $(R_1, R_2, \dots, R_t)$ to an evicted content with the constraint that $E_t^\pi \in \mathcal{M}_t \cup R_t$ when $R_t \notin \mathcal{M}_t$ and $E_t^\pi = \emptyset$ otherwise.

We allow the possibility for a policy π to be aware of the entire sequence $(R_1, R_2, \dots, R_T)$ at each slot t (*offline policy*), instead of only the requests $(R_1, R_2, \dots, R_t)$ (*online policy*). We will denote the set of online and offline policies with Π , which includes all the T -vectors of feasible eviction decisions. We will consider two eviction models:

- (I1) $E_t^\pi \in \mathcal{M}_t \cup R_t$, where the requested item at t is eligible for eviction.
- (I2) $E_t^\pi \in \mathcal{M}_t$, where the requested item at t is *not* eligible for eviction.

Model I2 is often encountered in the literature of databases and computing, while model I1 is a reasonable option for Internet caching

applications. Although their differences are minor, for clarity, we will make explicit reference to the employed model.

A note on function h : Cache hits are central to this monograph, and we will encounter different manifestations of function h , sometimes abusing the notation to reflect the hit probability or the actual number of hits. To help the reader, we present the notation collectively here:

- $h^\pi(\mathbf{R})$: are the number of hits using policy π under sequence $\mathbf{R}$,
- $m^\pi(\mathbf{R})$: the number of misses (complementary of hits),
- χ^π : hit rate of π , the limit of the empirical probability of hit,
- $h(\mathbf{y})$: hit probability when caching contents $\mathbf{y}$,
- $h^\pi(\infty)$: hit probability when reaching a stationary state,
- $h_n^\pi(M; t_n(M))$: content hit probability for characteristic time $t_n(M)$,
- $h_n^\pi(T_n)$: content hit probability for choice of TTL timer T_n .

An archetypical form of hit probability function is $h_n^\pi(\mathbf{y}, M, T_n; \cdot)$, where π is the eviction policy, $\mathbf{y}$ a vector of cached contents, M the size of the cache, T_n the TTL timer, index n refers to conditioning to the hits for content n , and $\cdot$ a placeholder for other conditions. However, arguments will be dropped at will if they are irrelevant to the context of the discussion.

We denote with $h^\pi(\mathbf{R})$ the *number of hits* under policy $\pi \in \Pi$ and request sequence $\mathbf{R}$. We can now formally state the *paging problem*: find an eviction policy that maximizes the number of hits for the worst-case request sequence.

Paging problem

$$\max_{\pi \in \Pi} \min_{\mathbf{R} \in \mathcal{N}^T} h^\pi(\mathbf{R})$$

The problem is called paging since it was originally formulated in the scope of paging memory of computing systems [219]. It is often used as a didactic example of competitive analysis for deterministic algorithms.

Sequence R	7	7	9	1	1	2	2	5	3	7	8	10	1	2	2	5	3	3
Stack distance of																		
1	3	2	1	1	8	7	6	5	4	3	2	1	∞	∞	∞	∞	∞	∞
2	5	4	3	2	1	1	8	7	6	5	4	3	2	1	∞	∞	∞	∞
3	8	7	6	5	4	3	2	1	9	8	7	6	5	4	3	2	1	∞

Figure 3.2: Showcased stack distances $D(1, t)$, $D(2, t)$, $D(3, t)$.

3.1.2 Belady's Offline Paging

We first describe an offline policy that maximizes $h^\pi(\mathbf{R})$ for any $\mathbf{R}$, and therefore solves the paging problem. The algorithm was proposed in [26] by Belady, and it is also known as the *max stack distance*, or the “*MIN*” replacement policy (minimizes the misses), or caching oracle.

We define the *stack distance* $D(n, t)$ of content n at slot t as the number of slots until the next request for n . Formally:

$$D(n, t) = \inf\{\tau > t, R_\tau = n\} - t.$$

If there is no other occurrence of n in the remaining sequence, then we trivially set $D(n, t) = \infty$. An example is presented in Figure 3.2.

Belady (*MIN*) policy: evicts the content with the largest stack distance:

$$E_t^{MIN} = \arg \max_{n \in \mathcal{M}_t \cup R_t} D(n, t).$$

This policy works by evicting the content that will be requested further in the future; which means that it is impractical as it requires information about all future requests. Nevertheless, Belady plays an important role in the design of online eviction policies. First, since it achieves the maximum possible hits (Theorem 3.1), it provides the performance upper bound for others policies on known datasets of requests. It can be thus used to test whether the hits achieved by another policy are satisfactory. Second, as we will show in the following, it can be used to characterize the performance of practical online eviction policies. We state and prove next the optimality of Belady.

Theorem 3.1 (Belady achieves optimal paging): Fix T , and choose any sequence $\mathbf{R} \in \mathcal{N}^T$. Belady achieves the highest number of hits in class Π under both I1 and I2.

$$MIN \in \arg \max_{\pi \in \Pi} h^\pi(\mathbf{R}).$$

The classical proofs [145, 168, 209, 234] prove the statement under model I2. Here we provide a proof for I1.

Proof of Theorem 3.1 (case I1). Assume any eviction policy $\pi \in \Pi$. We will show that it yields less or equal hits to Belady, i.e., $h^\pi(\mathbf{R}) \leq h^{MIN}(\mathbf{R})$. The proof is based on showing that starting from a certain policy we can produce a new policy that takes one more decision like Belady and performs no worse than the former. Then we use induction on the number of steps. Below we drop the notation $(\mathbf{R})$ with the understanding that $\mathbf{R}$ is fixed.

Lemma 3.2 (One more Belady step): Fix any $\mathbf{R}$. Consider policy $\pi_t \in \Pi$ with $h^{\pi_t}(\mathbf{R})$ hits, which takes the same eviction decisions with MIN up to slot t , where $t \in \{0, \dots, T-1\}$, that is

$$E_\tau^{\pi_t} = E_\tau^{MIN}, \quad \forall \tau = 0, \dots, t.$$

Then, there exists a policy $\pi_{t+1} \in \Pi$ satisfying:

$$E_\tau^{\pi_{t+1}} = E_\tau^{MIN}, \quad \tau = 0, \dots, t+1. \quad (3.1)$$

$$h^{\pi_{t+1}}(\mathbf{R}) \geq h^{\pi_t}(\mathbf{R}). \quad (3.2)$$

Lemma 3.2 states that if we extend policy π_t to π_{t+1} which takes one more decision like Belady, we get greater or equal hits. Assuming w.l.o.g. that π from Theorem 3.1 performs the same steps with Belady up to slot $t \leq T$, by Lemma 3.2 and induction on t , we get $h^\pi(\mathbf{R}) = h^{\pi_t}(\mathbf{R}) \leq h^{\pi_{t+1}}(\mathbf{R}) \leq \dots \leq h^{MIN}(\mathbf{R})$, which proves the theorem. $\square$

Proof of Lemma 3.2. The proof is by construction. Let $a \in \mathcal{N}$ denote the max stack distance content evicted by Belady in slot $t+1$ (hence

also by π_{t+1}) and $\mathcal{M}_{t+1}^{\pi_{t+1}} - \mathcal{M}_{t+1}^{\pi_t} \neq \emptyset$ denote the content evicted by π_t , where $\mathcal{M}_\tau^\pi$ is the set of files that are cached at slot τ by policy π . We design π_{t+1} as follows: (i) for slots $0, \dots, t+1$ it takes a Belady decision; (by the premise of Lemma) (ii) for slots $\tau = t+2, \dots, T$ we have:

$$E_\tau^{\pi_{t+1}} = \begin{cases} E_\tau^{\pi_t} & \text{if miss and } E_\tau^{\pi_t} \neq \{a\} \\ \mathcal{M}_\tau^{\pi_{t+1}} - \mathcal{M}_\tau^{\pi_t} & \text{if miss and } E_\tau^{\pi_t} = \{a\} \\ \emptyset & \text{if hit.} \end{cases}$$

We immediately obtain that $\pi_{t+1} \in \Pi$ and (3.1) holds. It remains to show (3.2). Since π_t and π_{t+1} take the same decisions up to time t , they yield the same hits up to and including $t+1$. Hence we will compare the hits in slots $\{t+2, \dots, T\}$.

Consider two useful slot events in the interval $\{t+2, \dots, T\}$:

Ω : in a slot π_t gets a hit while π_{t+1} not

$\hat{\Omega}$: in a slot π_{t+1} gets a hit while π_t not.

Denote by $\mathcal{I}_t$ the intersection of the caching states of the two policies at slot t . Due to the different decision at $t+1$ we obtain:

$$\begin{aligned} \mathcal{M}_{t+1}^{\pi_t} &= \mathcal{I}_{t+1} \cup \{a\} \\ \mathcal{M}_{t+1}^{\pi_{t+1}} &= \mathcal{I}_{t+1} \cup \{b\} \\ \mathcal{M}_{t+1}^{\pi_t} - \mathcal{M}_{t+1}^{\pi_{t+1}} &= \{a\} \\ \text{Uncached contents: } \mathcal{N} - \mathcal{I}_{t+1} &= \{a, b\}, \end{aligned}$$

where b is the content evicted by π_t . Now, let us observe what are the possible outcomes in slot $\tau = t+2$ and onwards:

$$\begin{array}{lll} R_\tau = \{a\} & \Rightarrow \Omega & \text{and } \mathcal{M}_{\tau+1}^{\pi_t} - \mathcal{M}_{\tau+1}^{\pi_{t+1}} = \{\emptyset\} \\ R_\tau = \{b\} & \Rightarrow \hat{\Omega} & \text{and } \mathcal{M}_{\tau+1}^{\pi_t} - \mathcal{M}_{\tau+1}^{\pi_{t+1}} \subseteq \{a\} \\ R_\tau \in \mathcal{I}_{t+1} & \Rightarrow \text{hit for both} & \text{and } \mathcal{M}_{\tau+1}^{\pi_t} - \mathcal{M}_{\tau+1}^{\pi_{t+1}} = \{a\} \\ R_\tau \in \mathcal{N} - \mathcal{I}_{t+1} - \{a, b\} & \Rightarrow \text{miss for both} & \text{and } \mathcal{M}_{\tau+1}^{\pi_t} - \mathcal{M}_{\tau+1}^{\pi_{t+1}} \subseteq \{a\} \end{array}$$

which is explained as follows. Following a , π_{t+1} will store a and make the caching states identical. Following b , the cache of π_{t+1} will remain the same, but π_t can evict a to make caching states identical, b to keep the difference the same, or any other content $x \in \mathcal{I}_\tau$ to result in $\mathcal{M}_{\tau+1}^{\pi_{t+1}} - \mathcal{M}_{\tau+1}^{\pi_t} = \{x\}$. In any case, we always get $\mathcal{M}_{\tau+1}^{\pi_{t+1}} - \mathcal{M}_{\tau+1}^{\pi_t} = \{a\}$