Maximizing Page-Level Cache Hit Ratios in Large Web Services

Justin Wang, Benjamin Berg, Daniel S. Berger
Carnegie Mellon University

Siddhartha Sen
Microsoft Research

Abstract

Large web services typically serve pages consisting of many individual *objects*. To improve the response times of page-requests, these services store a small set of popular objects in a fast caching layer. A page-request is not considered complete until *all* of its objects have either been found in the cache or retrieved from a backend system. Hence, caching only speeds up a page request if *all* of its objects are found in the cache. We seek caching policies that maximize the *page-level hit ratio*—the fraction of requests that find all of their objects in the cache.

This work analyzes page requests served by a Microsoft production system. We find that in practice there is potential for improving the page-level hit ratio over existing caching strategies, but that analytically maximizing the page-level hit ratio is NP-hard.

ACM Reference Format:
Justin Wang, Benjamin Berg, Daniel S. Berger and Siddhartha Sen. 2018. Maximizing Page-Level Cache Hit Ratios in Large Web Services. In *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS).* ACM, NY, USA, 3 pages.

1 Introduction

Providers of large user-facing web services have long faced the challenge of achieving low response times [4, 9]. These web services often employ fast caching systems to improve response times. Because response times for cache hits are generally orders of magnitude faster than for cache misses, even small increases in cache hit ratios can significantly improve mean response time [2].

However, as web services increasingly personalize content for each user, achieving high cache hit ratios becomes increasingly difficult. For example, on xbox.com, users are shown different lists of games to purchase depending on their prior purchase history. Hence, two users who make the same request may receive different page[1], results. A naive caching approach, called *full-page caching*, stores each personalized page separately. However, each additional personalizable component on a page increases the number of possible pages exponentially, making full-page caching impractical. Moreover, full-page caching is often inefficient as the content served for two users may only differ slightly (e.g., the responses could share ten game offers and differ on just one offer).

As a result, many web architectures have moved from full-page caching to *object-level caches*, where we think of a single page as being composed of several objects. Figure 1 depicts this architecture. An application server first tries to retrieve a page-request's objects

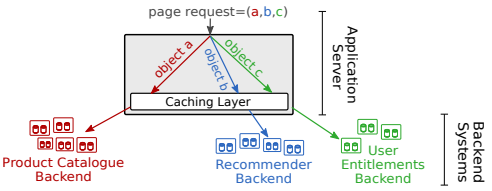


Figure 1: In large webservices, application servers assemble pages by aggregating objects from various backend systems.

from the cache and then retrieves cache misses from backend systems [12]. Once the server retrieved all necessary objects, they are assembled into a page, and the page-request is considered complete.

The challenge in object-level caching is that page-request response time will not be significantly reduced unless *all* the requested objects are found in the cache. If even a single object is not in the cache, the request must wait while a backend system is queried, which is typically much slower than a cache lookup [2]. Thus, although it may seem natural to try and maximize the *object-level hit ratio*—i.e., the fraction of object requests that are served from the cache—a high object-level hit ratio may not reduce page-request response times. Instead, we aim to maximize the *page-level hit ratio*, the fraction of pages which find *all* of their objects in the cache. For example, if every page-request consists of 100 objects and finds 99 of them in the cache, the object-level hit ratio would be 99%, but the page-level hit ratio would be 0%. If we could alternately cache all 100 objects for half of the page-requests and none of the objects for the other half, our object-level hit ratio would decrease to 50%, but the page-level hit ratio would increase from 0% to 50%!

Most existing work on building caching systems [2, 6, 7] and on analyzing caching systems [3, 5, 10, 11] focuses solely on the object-level hit ratio. Although [8] considers objects of different sizes and costs, the cost of a cache miss for objects in our setting is hard to describe, since it depends on the state of the cache and thus varies over time. Furthermore, while [1] considers caching to maximize page-level hit ratio, this work does not consider pages which can share common objects. The goal of our work is to explore caching policies that maximize the page-level hit ratio for pages composed of shared objects. Specifically, we study the example of the OneRF page rendering framework at Microsoft, which serves several major websites (including microsoft.com and xbox.com) and currently relies on an object-level caching architecture.

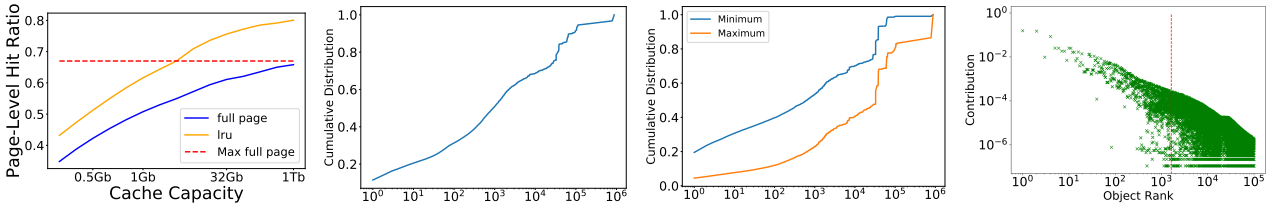
2 Evaluation of Object-Level Caching

We first evaluate the performance of object-level caching on a OneRF production trace and examine the extent to which page-requests share objects in common. We show that maximizing object-level hit ratio for this trace is not a good proxy for maximizing page-level hit ratio. This analysis uses a OneRF production trace from a full day in July 2017 which contains several hundred million object-queries and corresponding page-request-identifiers.

To evaluate object-level caching, we simulate an object-level LRU cache and a full-page LRU cache processing requests from the production trace. The results of this simulation are shown in Figure 2a. We find that the object-level cache significantly outperforms the

[1]Note that unlike [1], we use "page" to refer to web pages, which are composed of several "objects" such as ads, news, products, images, and so on.

SIGMETRICS, June, 2018, Irvine, California, USA



(a) Simulation results **(b) Object degree** **(c) Sharing degree** **(d) Contribution graph**

Figure 2: Analysis of a Microsoft production trace. In (a), our simulations show that object-level caches lead to significantly higher page-level hit ratios than full-page caching. In (b), we analyze the degree of objects and find that more than 50% of objects are shared between more than 10^3 page-requests. In (c), we calculate the minimum and maximum object degree for a given page-request and find that about 80% of requests share at least on object with over 10^3 distinct requests. In (d), we calculate the contribution to the page-level hit ratio for the 10^5 most popular objects. We rank the objects by popularity. For ranks above 10^3, the contribution varies wildly.

full-page cache with respect to page-level hit ratio. One might guess that this is due to space savings since, unlike the full-page cache, the object-level cache will not cache the same object multiple times. However, even as the cache size grows, the page-level hit rates do not converge. This is due to another advantage of the object-level cache: it reduces compulsory misses by allowing previously unseen pages to find all of their component objects in the cache due to previous, similar page-requests. Hence, the degree to which page-requests share objects in common will determine the ability of object-level caching to reduce both capacity misses and compulsory misses at the page level.

To understand how objects are shared between requests, we introduce the notion of *object degree*, which counts how many distinct requests a particular object belongs to. The cumulative distribution function in Figure 2b grows logarithmically with the object degree, showing that there is an even mixture of highly shared and mostly unique objects.

We can also compute sharing statistics at a request level. For each request, we compute the minimum and maximum object degrees over the objects in the request. We call these the minimum and maximum *sharing degrees* of a request. Figure 2c shows the cumulative distribution functions for these two metrics. We see that about 80% of requests share at least one object with over 10^3 distinct requests and roughly 50% of requests share each of their objects with over 10^3 other requests. Thus, nearly all requests are composed of very commonly used objects.

Finally, we show that maximizing object-level hit ratio is not sufficient to provide a high page-level hit ratio. We begin by considering an infinite-size cache that contains every object. For each object o, we calculate by how much the page-level hit ratio decreases when o is removed from this infinite cache. We call this difference o's *contribution* to the page-level hit ratio.

Figure 2d is a scatter plot of popularity rank vs contribution. The red line in this figure indicates the number of objects required to achieve a .5 object-level hit ratio. For objects with a rank higher than this threshold, making caching decisions is hard because objects with similar rank have wildly different contributions.

3 Maximing Page-Level Hits is Hard

We now consider the following simplified, offline version of our caching problem: *Given fixed page request probabilities and a fixed cache size, find the subset of objects that maximizes the expected long-term page-level hit ratio and fits into the cache.*

We show that this optimization problem is NP-Hard by using a reduction from the densest k-subgraph problem. We further observe this optimization problem is closely related to constrained minimization of a submodular function for which there is no known constant factor approximation polynomial time algorithm [13].

4 Conclusion

Large web services depend on effective object-level caching layers to reduce the mean response time of page-requests. However, maximizing the page-level hit ratio is challenging.

In future work, we seek to develop caching policies that incorporate how different objects contribute to the page-level hit ratio when making cache replacement decisions. We also seek to quantify more rigorously when request-aware caching is necessary by analyzing a more diverse set of traces including other web services.

References

[1] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. Pacman: Coordinated memory caching for parallel jobs. In *USENIX NSDI*, pages 20–20, 2012.
[2] N. Beckmann, H. Chen, and A. Cidon. LHD: Improving cache hit rate by maximizing hit density. In *USENIX NSDI*, pages 389–403, 2018.
[3] D. S. Berger, N. Beckmann, and M. Harchol-Balter. Practical bounds on optimal caching with variable object sizes. *POMACS*, 2(2):32, 2018.
[4] D. S. Berger, B. Berg, T. Zhu, M. Harchol-Balter, and S. Sen. RobinHood: Tail latency-aware caching - dynamically reallocating from cache-rich to cache-poor. In *USENIX OSDI*, 2018.
[5] D. S. Berger, P. Gland, S. Singla, and F. Ciucu. Exact analysis of TTL cache networks. *Perform. Eval.*, 79:2 – 23, 2014. Special Issue: Performance 2014.
[6] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. AdaptSize: Orchestrating the hot object memory cache in a content delivery network. In *USENIX NSDI*, pages 483–498, Berkeley, CA, USA, 2017. USENIX Association.
[7] A. Blankstein, S. Sen, and M. J. Freedman. Hyperbolic caching: Flexible caching for web applications. In *USENIX ATC*, pages 499–511, 2017.
[8] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *USENIX SITS*, pages 193–206, 1997.
[9] J. Dean and L. A. Barroso. The tail at scale. *CACM*, 56(2):74–80, 2013.
[10] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for LRU cache performance. In *ITC*, page 8, 2012.
[11] N. Gast and B. Van Houdt. Transient and steady-state regime of a family of list-based cache algorithms. In *ACM SIGMETRICS*, pages 123–136, 2015.
[12] C. Li, D. G. Andersen, Q. Fu, S. Elnikety, and Y. He. Better caching in search advertising systems with rapid refresh predictions. In *WWW*, 2018.
[13] K. Nagano, Y. Kawahara, and K. Aihara. Size-constrained submodular minimization through minimum norm base. In *ICML*, pages 977–984, 2011.