Optimum Caching versus LRU and LFU: Comparison and Combined Limited Look-Ahead Strategies

Gerhard Hasslinger
Deutsche Telekom
Darmstadt, Germany
gerhard.hasslinger@telekom.de

Juho Heikkinen
F-Secure
Helsinki, Finland
juho.heikkinen@f-secure.com

Konstantinos Ntougias
Athens Information Technology
Athens, Greece
kontou@ait.gr

Frank Hasslinger
Darmstadt Univ. of Tech.
Darmstadt, Germany
frank.hasslinger@stud.tu-darmstadt.de

Oliver Hohlfeld
RWTH Aachen
Aachen, Germany
oliver@comsys.rwth-aachen.de

Abstract — **We compare web caching strategies based on the least recently used (LRU) and the least frequently used (LFU) replacement principles with optimum caching according to Belady's algorithm. The achievable hit rates of the strategies are shown to improve with the exploited knowledge about the request pattern while the computation effort is also increasing. The results give an overview of performance tradeoffs in the whole relevant range for web caching with Zipf request pattern.**

In a second part, we study a combined approach of the optimum strategy for a limited look-ahead with LRU, LFU or other non-predictive methods. We evaluate the hit rate gain depending on the extent of the look-ahead for request traces and for the independent reference model (IRM) via simulation and derive an analytic confirmation of the observed behaviour. It is shown that caching for video streaming can benefit from the proposed look-ahead technique, when replacement decisions can be partly revised due to new requests being encountered during long lasting content updates.

Keywords — **Web cache strategies, optimum caching, Belady's algorithm, hit rate, simulation, Zipf distributed requests, least recently used (LRU), least frequently used (LFU)**

I. Introduction

Caching strategies have an important role for optimizing the transport and quality of experience of popular web services. Caching architectures on global scale are distributing the major traffic volume on the Internet, while servers and caches close to the users become more important for a broadening set of 5G applications with stringent delay bounds.

We study the hit rate and update effort of usual caching strategies versus optimum caching as an upper performance bound. The three basic methods for performance estimation of caching strategies are simulation via traces, simulation of randomly generated request pattern following a synthetic model, and analysis. In a first part, we involve all three methods to combine their strengths to evaluate cache performance. We evaluate caching methods based on web request traces from F-Secure's caching platform [5], which exhibit Zipf distributed requests with low time dynamics in the set of popular objects.

Experience from the traces as well as from other measurement-based studies of web requests [13][23] confirm the relevance of the independent reference model (IRM) as a simple and realistic benchmark for web cache performance. A number of studies suggest non-predictive caching variants improving the LRU hit rate [12][13][17][18][20][21][23]. Only few of them include a comparison with the optimum caching bound. Early work on optimum caching does not refer to web caches [2][16] and recent work [21] shows largely different gaps of usual caching methods towards Belady's clairvoyant strategy depending on the different scenarios.

Therefore we evaluate the cache hit rate for independent Zipf distributed requests, which are confirmed to represent a realistic model of web request pattern. We obtain comprehensive results in a series of simulations covering the whole relevant parameter range. The shape parameter β of the Zipf distribution and the ratio M/N of the cache size to the size of the object catalogue (in number of objects) turn out to have most significant impact on the hit rates and the differences in the performance of the caching strategies.

Cache performance results are partly confirmed by analytical methods. The Che approximation [4] is known to provide a good estimate of the LRU hit rate for IRM requests [11][12], although without clear precision bound. The LFU hit rate is approaching the sum of the top-M IRM request probabilities. Belady's algorithm corresponds to a simulation with precomputation as pointed out in Section II. A recent study [9] derives new analysis results on IRM hit rates for optimum caching but exact analysis is experienced to become intractable for moderate or large cache size. However, an equivalent Markov approach is established in this study as an alternative to Belady's algorithm for evaluating simulations and traces.

In a second part, we study scenarios to exploit the optimum caching gain by limited look-ahead. We focus on scenarios, with non-negligible delay from the decision to evict data from the cache until the data is overwritten by uploads of new data. Uploads of video streams usually can last for several minutes. During this time, a decision to evict a video can be withdrawn when new requests are encountered where the portion of data is kept in the cache that has not yet been evicted. In this way, limited look-ahead scenarios can improve the cache performance towards the optimum caching bound. Other approaches have been proposed to improve cache efficiency using awareness of optimum caching and Belady's algorithm [1][6][8].

The derived performance results for a single caching system represent the essential component in extended studies on distributed and/or hierarchical caching systems in content delivery via clouds, CDN and ICN architectures [4][10][19][24].

We briefly introduce usual caching schemes and Belady's algorithm in Section II. Zipf request pattern are defined in Section III. Evaluations of request traces are presented in section IV and evaluations of independent Zipf distributed requests in Section V. Section VI is devoted to the efficiency of limited look-ahead scenarios followed by the conclusions.

Performance of Caching Methods	Information Base	Effort per request	IRM hit rate analysis	IRM hit rate performance
LRU	Sequence of most recent requests of items in the cache	Constant, minimum effort	Accurate Che approximation	~10% below LFU in most relevant cases
(Sl. Window) LFU	(Limited) Count of past requests per item	Constant	Exact formula	Sum of top-M request probabilities
Score Gated LRU	Any statistics of past requests per item as score	Constant, depends on score updates	Simulation, $O(1)$ p. request	Depends on scores includes LRU, LFU
Optimum Caching	Future request sequence	Belady's algor. $O(\ln(M))$	Simulation, $O(\ln(M))$ p. req.	Optimum; often ~10% above LFU

Table 1: Main performance properties of basic caching methods

II. Performance Comparison of Web Caching Strategies

We start comparing the performance of LRU, LFU and optimum caching as basic strategies for web caching and we consider useful combinations of those methods in further evaluations. An overview of the expected tradeoff in the hit rate performance is given especially for independent Zipf distributed requests, which are regarded as a realistic benchmark for accessing web platforms. Moreover, we include trace based evaluations from F-Secure's caching platform to verify the accuracy of simulations under the independent reference model (IRM) [5]. We also address the update effort of the caching methods, whose main properties are summarized in Table 1.

The least recently used (LRU) principle, which keeps the M most recently requested items in a cache of size M, is widely used because of its simple implementation and updates [4][20]. The currently requested object is always put on top of a stack being organized as a double chained list, whereas the bottom object of the cache is evicted in case of a cache miss [12]. This leads to low constant effort for updating the stack per request by manipulating a few pointers belonging to the requested and the bottom object. On the other hand, the achievable LRU hit rate is low [12][17][18][21] because relevant information for predicting the next requests is not fully exploited.

The least frequently used (LFU) principle puts the objects with highest request count into the cache and therefore maintains statistics of past requests per object. If we assume an independent reference model (IRM) with request probabilities $p_1 \geq p_2 \geq \ldots \geq p_N$ for a fixed catalogue of N objects, then LFU will collect the most popular objects in the cache in a long term steady state behaviour. For cache size M, LFU approaches the hit rate $h_{\text{LFU}} = p_1 + p_2 + \ldots + p_M$, which is the maximum IRM hit rate for any caching strategy based on past requests.

LFU can be implemented as a sorted list of N items according to the request counts. Updates of an LFU cache can still be done at constant $O(1)$ effort per request, where only the count of the requested object is incremented [22]. Variants with limited request count are preferable to pure LFU in order to adapt to changing popularity of web content. Sliding window LFU is often used as a variant, which restricts the count statistics to the W most recent requests. Sliding window LFU can still be implemented at constant effort per request. Score gated LRU strategies [12] provide a flexible alternative with constant update effort, which include window LFU and other scheme [17].

A completely different approach considers the cache hit rate with full information about future requests. Then Belady's algorithm is known to maximize the hit rate [2], which decides about the cache content based on the time until the next request to each object. In case of a cache miss, the requested object is put into the cache, if its next request comes earlier than that of a cached object, replacing the object with longest time until its next request. It is a greedy algorithm, always enabling the next possible hit in the future request sequence r_T.

Therefore the implementation maintains a sorted cache list according to the time index of the next requests. We assume the future requests r_t being available for $1 \leq t \leq T_{\max}$, provided by a trace that has been monitored on a web platform or by a random generator for simulating requests, e.g., for independent IRM requests.

Together with object $o_k(t)$ being addressed at request r_t we store the index $To_k^+(t)$ of the next request to the same object o_k in a second array also for $1 \leq t \leq T_{\max}$, where $To_k^+(t) = T_{\max} + 1$ if o_k isn't requested anymore. Figure 1 illustrates the data sets being required for efficient execution of Belady's algorithm.

$To_k^+(t)$ is determined by a single complete scan through the future request sequence. During the scan, the time $To_k^-(t)$ of the currently most recent request to each object o_k is stored ($To_k^- = 0$ if there was no previous request to o_k). Then $To_k^+(t)$ is fixed for a new request to o_k in the scan backwards at previous time index To_k^- and To_k^- is then updated to t. In Figure 1, r_{T+3} is a request to o_2 and $To_2^-(T+3) = T \Leftrightarrow To_2^+(T) := T+3$; next, at r_{T+4}: $To_2^+(T+2) := T+4$; for r_{T+6}: $To_1^+(T+5) := T+6$; etc.

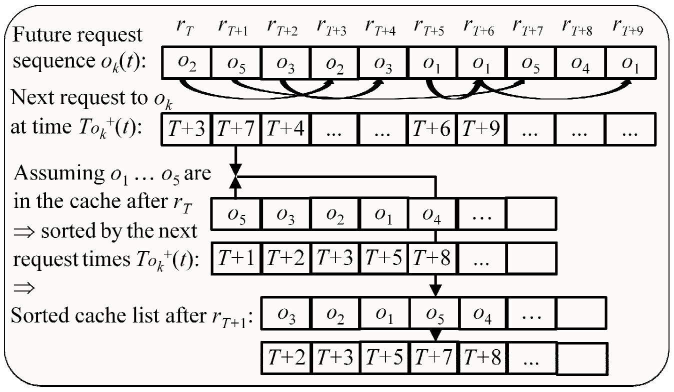
Future request sequence $o_k(t)$:
r_T r_{T+1} r_{T+2} r_{T+3} r_{T+4} r_{T+5} r_{T+6} r_{T+7} r_{T+8} r_{T+9}
o_2 o_5 o_3 o_2 o_3 o_1 o_1 o_5 o_4 o_1
Next request to o_k at time $To_k^+(t)$:
$T+3$ $T+7$ $T+4$ ... ... $T+6$ $T+9$ ... ...
Assuming $o_1 \ldots o_5$ are in the cache after r_T ⇒ sorted by the next request times $To_k^+(t)$:
o_5 o_3 o_2 o_1 o_4 ...
$T+1$ $T+2$ $T+3$ $T+5$ $T+8$ ...
⇒ Sorted cache list after r_{T+1}:
o_3 o_2 o_1 o_5 o_4 ...
$T+2$ $T+3$ $T+5$ $T+7$ $T+8$ ...

Figure 1: Data set for efficient support of Belady's algorithm

While the precomputation of $To_k^+(t)$ is executed at constant effort per request, Belady's algorithm finally requires $O(\log(M))$ complexity per request for inserting objects into the sorted list of cached objects, e.g. via heapsort [8][17].

III. Hit Rate Evaluation for Zipf Request Pattern

In the following comparison of hit rates for LRU, LFU and optimum caching strategies, we assume Zipf distributed requests as confirmed in many studies about access pattern to web platforms [3][12][13][14]. Zipf's law assigns decreasing request probabilities $z(r)$ corresponding to the objects' popularity ranks $r \in \{1, 2, \ldots, N\}$:

$$z(r) = \alpha\, r^{\beta} \text{ for } \alpha > 0; \beta \le 0; \quad \alpha = z(1) = 1/\sum_{r=1}^{N} r^{\beta} \qquad (1)$$

with shape parameter β and a normalization constant α. The measurement studies have experienced Zipf distributed requests on a number of different web platforms with estimations of the shape parameter β in the range $-0.4 \ge \beta \ge -1$.

IV. Evaluation for Traces of Web Requests

We consider trace-based evaluation of requests from F-Secure's platform [5] as well as analytic and simulation results of IRM requests in the next section. Trace-based results are shown for an example of about 26 million requests to 2.09 million keys over a one week period from Oct. 17-23, 2016. Again, a Zipf-like distribution with shape parameter $\beta \approx -0.75$ is observed for the top-N most popular objects in the trace, where about 50% of the requests are addressing the top-10000 keys.

The traces were collected from a caching proxy, through which F-Secure's applications queried a backend database for information on application files or URLs they encountered. After stripping any personally identifiable information from the data, a hash was calculated from the object data, representing a query key. No information that could identify from which individual client a particular query came from was kept. The traces contain only a list of hash strings with timestamps. The pool of clients connecting to the caching proxy consists almost entirely of Android mobile devices, querying hashes of new and updated Android application files.

We compare cache hit rates for optimum caching with LRU and LFU strategies. Each evaluation includes the complete trace starting from an empty cache. Pure LFU includes the count statistics over the whole one week trace, which is subject to an inflexibility regarding dynamics on shorter time scales. Therefore LFU variants over a limited time frame (sliding window LFU) are more efficient. The time scale ranging from an hour up to a day is experienced to be most relevant for the dynamics in web request pattern [13][23]. Therefore we add an evaluation for LFU with daily count statistics, i.e. with counts being reset at the start of each day.

Figure 2 shows how the hit rate is developing with the cache size for the alternative strategies. On the whole, caching is efficient, such that a cache with size $M/N = 1‰$ of the catalogue size N achieves more than 20% hit rate in all cases. However, there is a large gap of up to 15% hit rate visible between LRU and LFU and a gap of about half the size between LFU and optimum caching. LFU with daily count statistics improves the LFU hit rate with count over the whole week by only 1-2%. We also studied sliding window LFU [12][17] with count statistics over the K most recent requests. Then a maximum hit rate was obtained for $K \approx 50\,000$ roughly corresponding to 20 minute of the trace, but the improvement over LFU with daily request count turned out to be negligible.

The results confirm that LFU based on daily statistics is close to optimum for non-predictive caching strategies, but still leaves a considerable gap open towards clairvoyant optimum caching.

V. Evaluation for IRM Zipf Request Pattern

In the sequel, we evaluate the cache hit rate for independent Zipf distributed requests via simulation. Assuming IRM Zipf request pattern, a caching system is specified by 3 parameters: the catalogue size N, the cache size M and the shape parameter β of the Zipf distribution. The results give an overview of the hit rate performance in the relevant range of those parameters. Each simulation is running over at least 10^8 requests, where a cache filling phase at the start is ignored. The precision is validated via 2nd order statistics [12], confirming standard deviations for the simulated hit rates in the order 10^{-4}. The Markov model of optimum caching is used for an independent check of Belady's algorithm [9].

Figure 3 - Figure 6 show results for $\beta = 0, -0.5, -0.75, -1$. The case $\beta = 0$ refers to a uniform request distribution, whereas the three other cases are distributed over the range, which is experienced as most relevant in measurements of web request pattern ($-0.4 \ge \beta \ge -1$). The performance of optimum caching represents the main new insight of this study, which has not been considered for web caching examples or Zipf request pattern in basic work in literature [2][8][9][15][16].

The LRU and LFU hit rates for uniform request shown in Figure 3 are equal the ratio M/N of the cache size to the catalogue and mark a common worst case for both strategies. However, predictive optimum caching achieves much higher hit rates h_{Opt}, which mainly depend on the fraction M/N, with almost identical curves for $N = 10^3, \ldots, 10^6$ within the bounds $1.37\sqrt{M/N} > h_{\text{Opt}} > 1.15\sqrt{M/N}$.

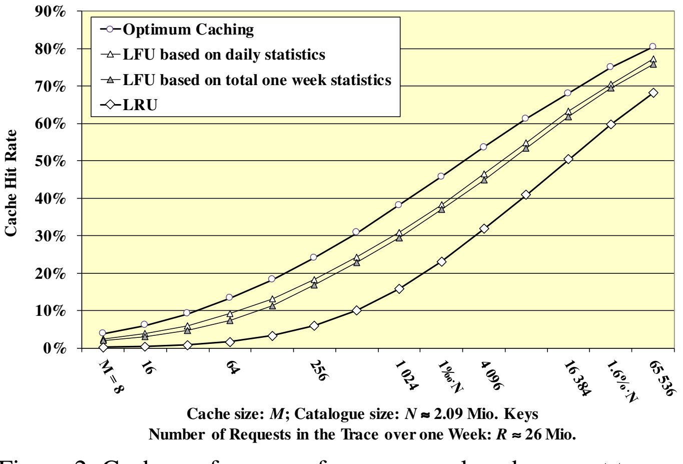

Figure 2: Cache performance for a one week web request trace

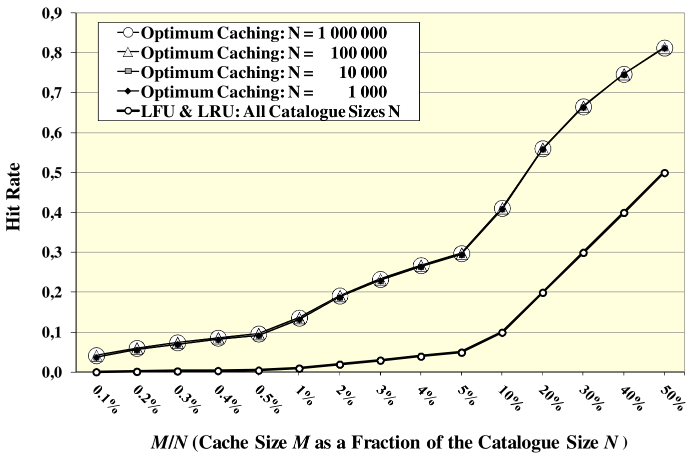

Figure 3: Hit rate of optimum caching for uniform requests

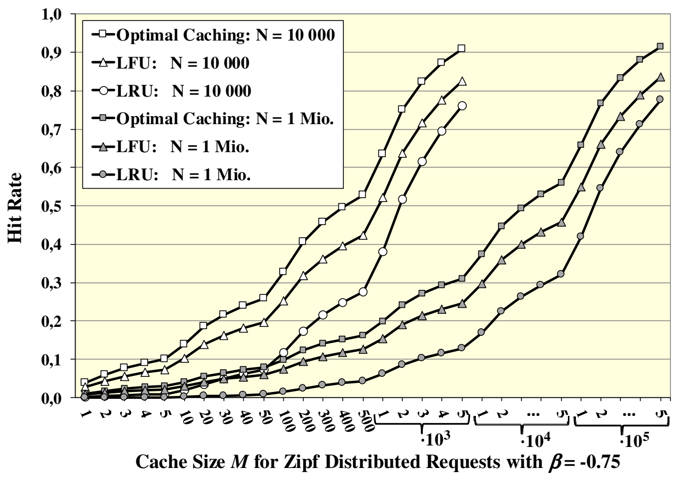

Figure 5: Cache hit rates for Zipf law requests with $\beta = -0.75$

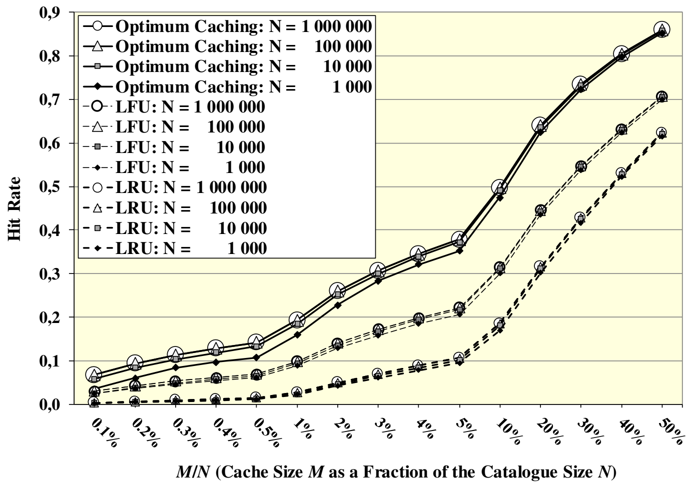

Figure 4: Cache hit rates for Zipf law requests with $\beta = -0.5$

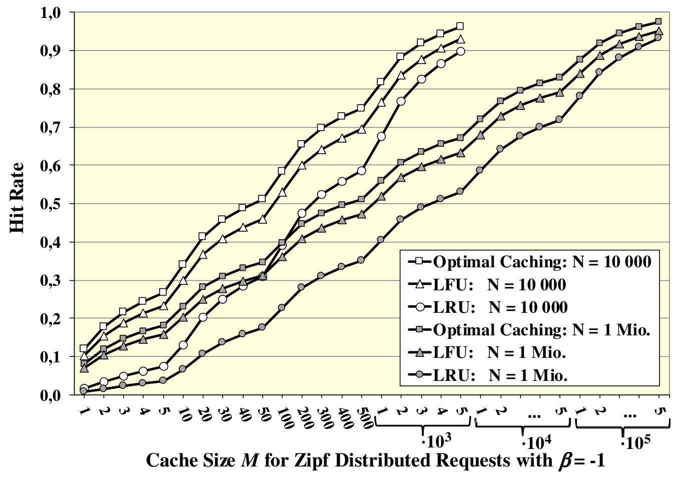

Figure 6: Cache hit rates for Zipf law requests with $\beta = -1$

Although uniform requests are a worst case for optimum caching as well, the gain over LRU and LFU is extreme, up to 37%.

Figure 4 shows results for $\beta = -0.5$ which corresponds to a moderate request focus on popular objects within the relevant range of web request pattern. The hit rates still mainly depend on the fraction M/N of objects in the cache, leading to bundles of four closely adjacent curves for $N = 10^3, \ldots, 10^6$ with largest deviation for optimum caching in case $N = 1000$. Compared to uniform requests, the hit rate gap of LRU and LFU towards optimum caching becomes smaller but still opens up to 30% for LRU and 20% for LFU.

For Zipf distributions with more skewness towards popular objects for $\beta \rightarrow -1$, caching efficiency depends on both parameters M and N, rather than mainly on M/N as for $0 > \beta > -0.5$. Thus Figure 5 ($\beta = -0.75$) and Figure 6 ($\beta = -1$) show two hit rate curves for $N = 10^4$ and $N = 10^6$ for each of the three considered strategies. In general, cache hit rates essentially increase for all caching strategies with higher request focus on popular objects. However, LRU hit rates remain far below optimum caching in all cases of small and moderate cache sizes.

On the other hand, LFU can lower the hit rate gap towards optimum caching for $\beta \rightarrow -1$, since count statistics on past requests are more useful for higher concentration on few popular items. The results suggest $h_{\text{Opt}} - h_{\text{LFU}} \approx 0.25 + 0.2\beta$ as a rough estimate of the gap for $-0.5 \geq \beta \geq -1$. Extensions for $\beta < -1$ are possible, but this range is not relevant for web caching.

VI. Using Limited Knowledge of Future Requests

A. *Look-ahead options for caching due to delayed uploads*

The previous results show a significant advantage of optimum caching over other methods owing to knowledge of future requests. Often some partial knowledge about upcoming requests is available and used for prediction and prefetching [1][6]. Within this scope, we evaluate how far a limited look-ahead can realize part of the optimum caching gain when a fixed number L of the next upcoming requests are known, which corresponds to a more or less fixed time window.

Such a limited look-ahead can be realized, when there is a delay between a request and the corresponding upload of content to the cache. During the delay time some "future" requests become visible which can still be regarded in a replacement decision until the update is actually processed. Data uploads are initiated only in case of cache misses, when external objects is requested that have to be retrieved from a server or higher layer cache. Therefore we can always assume a short transfer delay for web cache updates between data centers, roughly in a range of 0.1s - 1s for small data units.

A considerable number of requests is served by web caches already in short time. Wikipedia as a popular web site reported peaks of over 50 000 requests per second in 2014 being handled by a few caching servers [24]. Akamai's CDN had peak loads beyond 30 million requests per second in 2013 [7], which are distributed over a large web caching hierarchy. Thus caches for popular web content usually have information about several thousand upcoming requests available until data is retrieved for an update, even in short transfer delay scenarios.

Moreover, we consider video streaming as a scenario with much longer upload times, when uploads are synchronized with the video stream to the user. Then data is continuously transferred in small chunks during the video viewing time, which can last in a range over seconds and minutes up to an hour [14]. While data chunks of a video in the cache are being overwritten, new requests to the video can be regarded in a look-ahead scenario to stop further replacement. When the replacement of a one minute video is stopped, e.g. after 15 s, then 75% of the data can still be kept to serve a new request.

B. Combined caching scheme to exploit limited look-ahead

In order to utilize a limited look-ahead L for improved hit rates, we suggest combining Belady´s algorithm for those objects with known next request time ($< L$) with a secondary usual caching scheme (LFU, LRU etc.) for all other objects.

At a request r_T, all objects in the cache with a next request before r_{T+L+1}, i.e. within the look-ahead window are sorted according to their next request time as shown for Belady's algorithm in Figure 1. The next request time of other cached objects is unknown and invalidated, e.g. as 0. After request r_T, the sorted list is updated by reinserting the requested object, if its next request comes before r_{T+L+1}. The object requested at r_{T+L+1} is added at the end of the list, if the latter is in the cache with a previously invalid next request time.

If the sorted cache list of object with valid next request times exceeds the cache size M, then the object with the farthest next request time is evicted. Otherwise, if the list is shorter than the cache size, then all objects in the list stay in the cache, whereas the secondary caching strategy (LRU, LFU or another) is applied to select and update the content in the remaining part of the cache.

C. Hit rate evaluation for the limited look-ahead scheme

We simulate the gain obtainable by a limited look-ahead for the next L requests using a combined strategy of optimum caching with LRU. We show two evaluations extending the previous results for the one week trace of Figure 2 and for the Zipf distributed requests with $\beta = -0.5$ of Figure 4.

The performance of limited look-ahead is presented for both cases in Figure 7 and Figure 8, respectively. They include the curves for the optimum caching hit rate and for LRU known from Figure 2 and Figure 4 as the maximum and the minimum. Moreover, four curves are added in between, corresponding to evaluations of four look-ahead variants with different limit L. In each case we observe a common effect that caching performance of limited look-ahead

- starts along the optimum caching curve up to cache size M^*,
- then only slightly improves with increasing cache size beyond M^*, while sliding from optimum caching performance towards the LRU curve
- and finally approaches the LRU curve for large cache size.

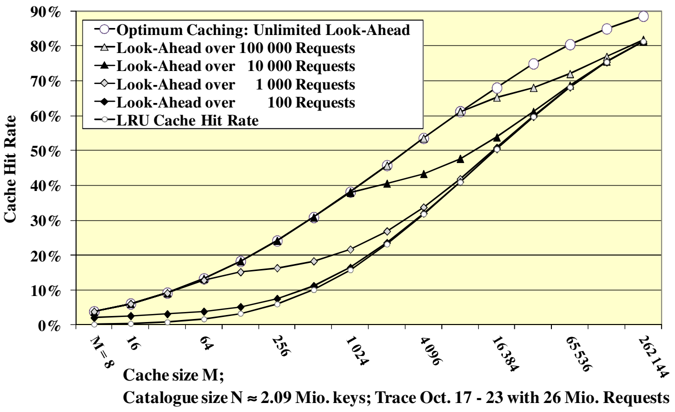

Figure 7: Effect of limited look-ahead for trace data

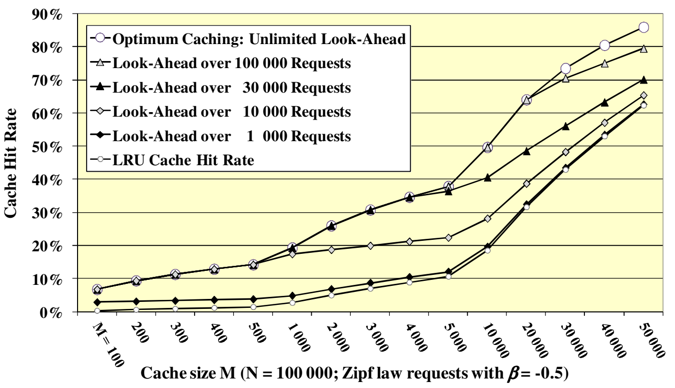

Figure 8: Effect of limited look-ahead for Zipf law requests

When optimum caching performance is achieved, the cache is almost filled with objects whose next request comes in the limited look-ahead region before r_{T+L+1}. Otherwise, many objects are encountered, whose next request comes beyond the limit. We obtain similar results for combinations with LFU or another strategy instead of LRU, where the LRU hit rate is replaced by their curve as the lower bound.

D. Analytical result on optimum limited look-ahead caching

We can derive an analytic result for the cache size M^* up to which optimum caching is fully exploited with limited look-ahead for IRM requests. M^* marks the points in Figure 8, at which the limited look-ahead curves start to deviate from optimum caching. Under IRM, we have geometrically distributed intervals I_j between requests to the same object o_j depending on the request probability p_j. We can compute the probability $\Pr\{I_j \le L\}$ that the next request comes within the limit L, the mean number $E(I_j \mid I_j \le L)$ of requests that such an object o_j has to stay in the cache until a next hit on o_j. Both values finally determine the mean number $E(n_{\le L})$ of objects with next request before r_{T+L+1}:

$$\Pr\{I_j \le L\} = 1-(1-p_j)^L; \; E(I_j \mid I_j \le L) = \frac{1}{p_j} - \frac{L(1-p_j)^L}{1-(1-p_j)^L};$$

$$E(n_{\le L}) = \sum_{j=1}^{N} p_j \Pr\{I_j \le L\} E(I_j \mid I_j \le L) = \sum_{j=1}^{N} 1-(1+p_j L)(1-p_j)^L$$

If $M < E(n_{\le L})$ then the cache is expected to fill up with objects having a valid next request time within the limit L, such that optimum caching is prevalently applied for updates. We calculate $E(n_{\le L})$ for the four curves with different limits L in Figure 8 based on the underlying Zipf request probabilities p_j: $E(n_{\le 1000}) \approx 12.7$; $E(n_{\le 10\,000}) \approx 740.3$; $E(n_{\le 30\,000}) \approx 4328$; $E(n_{\le 100\,000}) \approx 23\,079$. Those numbers precisely mark the cache sizes at which each limited look-ahead curve starts deviating from optimum caching.

The results in Figure 7 and Figure 8 indicate that the proposed combined caching strategy with limited look-ahead has significant effect for $L \ge 10\,000$, and in the trace evaluation already for $L \ge 1\,000$. The trace is taken from a cache serving about 100 requests per second. Thus a delay of 10 s - 100 s is sufficient to make an exploitation of the look-ahead beneficial. Concluding, the look-ahead can improve cache efficiency for video streaming, with delays of > 10 s until most of the data is uploaded, whereas for short transfer delays < 1 s the look-ahead will be reasonable only for caches serving huge request rates.

Conclusions and Outlook

The comparison of usual non-predictive caching schemes with clairvoyant optimum caching due to Belady's algorithm shows large gaps in the hit rate not only in extreme cases for uniform requests, but also over the complete range of Zipf request pattern that is confirmed to be relevant for web caching in literature. The results in Figure 3 - Figure 6 give an overview of the expected hit rate performance and differences in comparison of optimum caching, LFU and LRU for the independent reference model. Our trace-based study confirms that such Zipf distributed IRM results meet request pattern on web platforms.

The evaluation of a combined caching method making use of limited look-ahead scenarios enabled by delays in cache uploads show that optimum caching is useful not only to provide an upper bound on cache hit rates, but can partly be exploited for caching of video streams and for caches serving huge request workloads. A more detailed analysis of the impact of caching and request specific parameters on the applicability of the limited look-ahead scheme is for future study.

Acknowledgements

This work has received funding from the European Union´s Horizon 2020 research and innovation programme in the EU SSICLOPS project <www.ssiclops.eu> under grant agreement No. 644866. This work reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] N. Beckmann and D. Sanchez, Maximizing cache performance under uncertainty, Proc. 23rd HPCA Conf. (2017)

[2] L.A. Belady, A study of replacement algorithms for a virtual-storage computer, IBM Systems Journal 2 (1966) 78-101

[3] L. Breslau et al., Web caching and Zipf-like distributions: Evidence and implications, Proc. IEEE Infocom (1999)

[4] H. Che, Y. Tung, and Z. Wang, Hierarchic web caching systems: modeling, design and experimental results, IEEE JSAC 20(7) (2002) 1305-14

[5] F-Secure Inc. <www.f-secure.com>

[6] J. Famaey, F. Iterbeke, T. Wauters and F. De Turck, Towards a predictive cache replacement strategy for multimedia content, Journal of Network and Computer Applications 36/1 (2013) 219-227

[7] P.W. Gilmore, Akamai & ISPs, Presentation UKNOF25 (2013) <slideplayer.com/slide/10588098>

[8] J. Guo et al., The power of Belady's algorithm, Proc. 16th Workshop on Languages and Compilers for Parallel Computing, College Station, TX, USA, Springer Lecture Notes in Computer Science, LNCS 2958 (2003) 374-390

[9] G. Hasslinger, Markov analysis of optimum caching as an equivalent alternative to Belady's algorithm without look-ahead, Proc. MMB 2018, Springer LNCS 10740, Erlangen, Germany (2018) 35-52

[10] G. Hasslinger and F. Hartleb, Content delivery and caching from a network provider's perspective, Special Issue on Internet based Content Delivery, Computer Networks 55 (Dec. 2011) 3991-4006

[11] G. Hasslinger, K. Ntougias and F. Hasslinger, Performance and Precision of Web Caching Simulations, Proc. 18th MMB Conf., Münster, Germany, Springer LNCS 9629 (2016) 60-76

[12] G. Hasslinger, K. Ntougias, F. Hasslinger and O. Hohlfeld, Performance evaluation for new web caching strategies combining LRU with score based object selection, Computer Networks 17 (2017) 172-186

[13] G. Hasslinger et al., Web Caching Evaluation from Wikipedia Request Statistics, Proc. 15th Symposium on Wireless Optimization (WiOpt'17), CCDWN workshop, Paris (2017)

[14] M. Hefeeda and O. Saleh, Traffic modeling and proportional partial caching for peer-to-peer systems, IEEE/ACM Trans. on Networking 16/6 (2008) 1447-1460

[15] A. Jain and C. Lin, Back to the future: Leveraging Belady's algorithm for improved cache replacement, in Proc. ISCA-4, (2016)

[16] D.E. Knuth, An analysis of optimum caching, Journal of algorithms 6 (1985) 181-199

[17] D. Lee et al., LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Transactions on Computers 50/12 (2001) 1352-1361

[18] N. Megiddo and S. Modha, Outperforming LRU with an adaptive replacement cache algorithm, IEEE Computer (Apr. 2004) 4-11

[19] M. Pathan, R.K. Sitaraman and D. Robinson, Advanced content delivery, streaming and cloud services, Wiley (2014)

[20] S. Podlipnik and L. Böszörmenyi, A survey of web cache replacement strategies, ACM Computer Surveys (2003) 374-398

[21] A.A. Rocha et al., DSCA: Data stream caching algorithm, Proc. CCDWN workshop, Heidelberg, Germany (2015)

[22] K. Shah, A. Mitra and D. Matani, An O(1) algorithm for implementing the LFU cache eviction scheme (2010) Technical report, available via <dhruvbird.com/lfu.pdf>

[23] S. Traverso et al., Unravelling the impact of temporal and geographical locality in content caching systems, IEEE Trans. on Multimedia 17 (2015) 1839-1854

[24] Wikipedia statistics and information available on https://meta.wikimedia.org/wiki/Wikimedia_servers https://wikitech.wikimedia.org/wiki/Global_traffic_routing