Performance Analysis of Basic Web Caching Strategies (LFU, LRU, FIFO, etc.) with Time-to-live Data Validation

Gerhard Hasslinger
Deutsche Telekom
Darmstadt, Germany

Konstantinos Ntougias
University of Cyprus
Nicosia, Cyprus

Frank Hasslinger
Darmstadt Univ. of Tech.
Darmstadt, Germany

Oliver Hohlfeld
University of Kassel
Kassel, Germany

***Abstract*—Web caches often use a Time-to-live (TTL) limit to validate data consistency with web servers. We study the impact of TTL constraints on the hit ratio of basic strategies in caches of fixed size. We derive analytical results and confirm their accuracy in comparison to simulations. We propose a score-based caching method with awareness of the current TTL per data for improving the hit ratio close to the upper bound.**

***Index Terms*—LRU, LFU, FIFO caching strategies, hit ratio analysis and simulation, TTL validation of data consistency**

1. Caches with TTL Data Validation

Time-to-live caching policies assign a time limit to each data object for control of valid cache content. A main purpose of TTL constraints is to validate data consistency between web servers and caches, where invalid data may still be retrieved from a cache, but only for a limited time [1][2]. The German Telemedien law (TMG) § 9 [3] allows Internet service providers to store data for a limited time in order to improve efficiency, if they keep them up to date according to approved industry norms [4]. Data invalidation concepts with strict synchronization between web servers and caches have been analysed [5]. According to Zheng et al. [5], TTL validation is more usual: *"Most web applications apply validation rather than invalidation to maintain cache consistency due to the extra overhead on the network caused by the latter"*.

Then web caching strategies have to combine the selection of most relevant cache content for the caching goal with TTL validation, where shorter TTL values reduce the sojourn time and the entire amount of valid data in a TTL cache, as well as the hit ratio [6]. We still agree to a statement by Shim et al. [7]: *"However, the cache consistency algorithms are not typically well integrated into cache replacement algorithms. The published work on the topic usually considers the two algorithms as separate mechanisms and studies one of the two in isolation."*

The work by [7][8][9] is addressing cache performance together with consistency control. Berger et al. [9] consider two TTL timers per object, one for the control of the amount of data in the cache and a second for data consistency. Such approaches are generalized towards overall TTL-based utility concepts [10]. However, far most of the TTL caching analysis work stream as well as the analysis of fixed size cache performance does not include data consistency [11].

In fixed size caches, TTL restrictions are added on top of a usual cache management policy, such as Least Recently/Frequently Used (LRU/LFU), First-In-First-Out (FIFO), score-based or machine learning (ML) methods. As to the authors' knowledge, the impact of TTLs on the hit ratio has been studied for pure TTL caches [10][11], but results for strategies in fixed size caches are missing.

In Section 2, we introduce three main options of TTL reset policies. Their impact on the hit ratio is analyzed in Section 3 for independent requests to caches of fixed size with LFU or static strategies. In Section 4, we extend the LRU hit ratio approximations by Fagin and Che with regard to TTLs. Section 5 compares the impact of TTLs on LRU and LFU. The evaluations are extended to FIFO, Random and an optimized score-gated strategy, which comes close to the upper hit ratio bound. Section 6 addresses further extensions and Section 7 concludes the study.

2. Control Options for TTL Web Caches

Three cases can be distinguished for TTLs under cache or server control, as illustrated in Fig. 1 [10][11][12]:

- TTL Reset per Cache Miss (RpM)
- TTL Reset per Request (RpR)
- Periodic TTL Resets (PR)

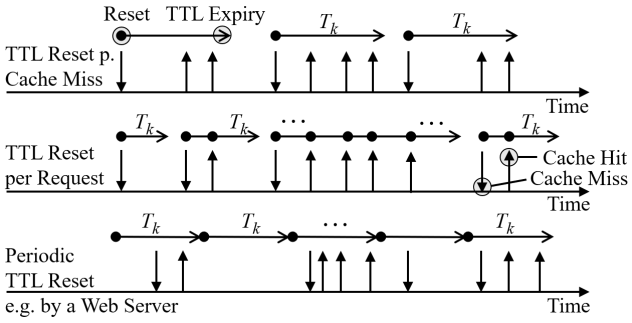

Figure 1: TTL reset options for caches and web servers

RpM and RpR are options under cache control, where resets per request lead to more frequent renewal of the TTL T_k of an object O_k than reset per miss. We assume a unique TTL T_k for all TTL resets for an object, even if the performance analysis is often extensible to varying TTL reset values until a next expiry. RpM is a useful cache validation option without further instructions from a server, whereas RpR may lead to many random resets until a next TTL expiry. Web server control often demands for periodical resets, independent of request and hit/miss events in caches. Then

the caches can be informed via HTTP [4] about the currently valid TTL policy of a server. The server-driven PR variant is most relevant for synchronized cache data validation.

3. Impact of TTL on Static Fixed Size Caches

We assume an independent reference model (IRM), which is characterized by request probabilities p_k to each object O_k in a fixed object catalogue ($k = 1, \cdots, N$). p_k is valid per request and independent of past requests.

We consider a TTL variant based on the request count, i.e., we assume that an object O_k is invalidated after R_k requests to all data objects, such that the limit R_k refers to requests in the entire data set and can be assumed to be large. Then $R_k = \lceil \lambda T_k \rceil$ (λ: entire request arrival rate) has similar effect as a TTL limit T_k on each object O_k and can be implemented and modelled without reference to the time. For dynamic caching strategies this allows to directly follow the cache content updates per request via Markov models, see Section 4. Section 6 extends the results for a mapping of T into a varying number of requests during time T. For a fixed reset counter R_k, we obtain the static IRM hit ratios:

$$h_{Static,IRM}^{TTL,RpM} = \sum_{k:\, O_k \in C} p_k^2 R_k/(p_k R_k + 1); \quad (1)$$

$$h_{Static,IRM}^{TTL,RpR} = \sum_{k:\, O_k \in C} p_k(1-(1-p_k)^{R_k}); \quad (2)$$

$$h_{Static,IRM}^{TTL,PR} = \sum_{k:\, O_k \in C} p_k - \frac{1-(1-p_k)^{R_k+1}}{R_k+1}. \quad (3)$$

Objects O_k in the cache contribute by p_k to the hit ratio of the next request under IRM. For RpM, there are a mean number of $p_k R_k$ cache hits in a reset interval R_k followed by a cache miss, which starts the next interval. Thus, the hit ratio on O_k is reduced by the factor $p_k R_k/(p_k R_k + 1)$. For RpR, each R_k interval without request to O_k causes a cache miss, which reduces the hit ratio by the factor $1-(1-p_k)^{R_k}$.

For periodic reset, there is one cache miss on O_k per reset interval with at least one request to O_k. Let R^* denote the number of all requests, R_k^* the number of requests to O_k and $p_k^* = R_k^*/R^*$. We subdivide the request sequence into n_k TTL intervals $R^* = n_k(R_k+1)$, where invalidation after R_k requests means that an object is valid for $R_k + 1$ requests, starting from the request at the reset instant. Then we obtain Eq. (4) for arbitrary request sequences and Eq. (3) for IRM, where the fraction of intervals without a request to O_k is denoted as p_k^0:

$$h_{Static}^{TTL,PR} = \sum_{k:\, O_k \in C} \frac{R_k^* - n_k(1-p_k^0)}{R^*} = \sum_{k:\, O_k \in C} p_k^* - \frac{1-p_k^0}{R_k+1}. \quad (4)$$

The LFU strategy converges to static caching of the most popular objects, yielding a maximum hit ratio under IRM requests and unit size data. Score-gated [13], GreedyDual and corresponding machine learning methods also converge to static caching and are covered by the results of Eq. (1-4), if the rank order of objects due to scores becomes stable over time for IRM. When objects of different size are preferred via scores for the highest value density, then those strategies maximize the caching value according to static knapsack solutions [11] extending LFU as unit data size optimum.

4. Hit Ratio Analysis of LRU with TTL

4.1. LRU hit ratio with TTL and reset per miss

The exact solution of the IRM hit ratio for LRU as derived by King [14] is tractable only for small caches. Instead, approximations by Fagin [15] and Che et al. [16] are applied for usual cache sizes with approved precision [17]. Both approximations assume a common characteristic time T_C until a data object is handed over from the top LRU position to the bottom and evicted, if there are no new requests to O_k. T_C equals the time to fill an empty cache of size M, until M out of N objects have been referenced and represents the LRU convergence time [11]. Then two causes have to be considered for a cache miss in the next request:

1. T_C is exceeded and the object is evicted, or
2. the TTL T_k of O_k has expired before the next request.

TTLs are again mapped to request count limits $R_k = \lceil \lambda T_k \rceil$ and $R_C = \lceil \lambda T_C \rceil$. If $R_k \geq R_C$, we analyse the requests from a cache miss until the next one as a Markov process with $R_k + 1$ states, as illustrated in Fig. 2. State 0 marks a cache miss on O_k. The states $1, \ldots, R_k$ mark the next requests to the entire data set. States beyond the expiry limit R_k lead to caches misses and thus back to state 0. O_k is evicted after a sequence of R_C requests without a reference to O_k as the other cache miss case with transit back to 0.

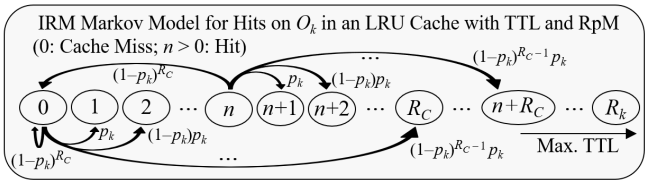

Figure 2: Markov model for LRU with TTL and RpM

Then the steady state probabilities $q_n (n = 1, \ldots, R_k)$ that the object O_k is referenced at state n before the next miss, i.e., for a cache hit on O_k at state n can be derived. Based on independent requests, a hit in state n is followed by a next hit in state $n+m$ with probability $(1-p_k)^{m-1}p_k$. We directly obtain $q_1, \cdots, q_{R_k}$ as multiples $q_n = f_n q_0$ of q_0, because all downward transitions lead to state 0 ($q_0 = 1/\sum_n f_n$):

$$\begin{aligned} q_1 &= p_k q_0; \quad q_2 = (1-p_k)p_k q_0 + p_k q_1 = p_k q_0; \ \cdots \\ q_n &= (1-p_k)^{n-1} p_k q_0 + \cdots + (1-p_k)p_k q_{n-2} + p_k q_{n-1} \\ &= p_k q_0 \qquad \text{for } n \leq R_C; \\ q_n &= (1-p_k)^{R_C-1} p_k q_{n-R_C} + \cdots \\ &\quad + (1-p_k)p_k q_{n-2} + p_k q_{n-1} \qquad \text{for } n \geq R_C. \end{aligned} \quad (5)$$

If $R_k < R_C$, the object O_k always expires before eviction and we obtain the hit ratio $h_k = p_k R_k/(p_k R_k + 1)$ for O_k from Eq. (5) similar to Eq. (1). In general, the IRM hit ratio of LRU with TTLs R_k and RpM resets is given by:

$$h_k = q_1 + \cdots + q_{R_k}; \ h_{LRU,IRM}^{TTL,RpM} = \sum_{k=1}^{N} p_k\, h_k. \quad (6)$$

4.2. LRU hit ratio with TTL and reset per request

Based on the LRU approximations [15][16], an object O_k is again evicted after $R_C = \lfloor \lambda T_C \rfloor$ requests to other objects, or TTL expiry after $R_k = \lfloor \lambda T_k \rfloor$ is the second cache miss case. Then the probability of a hit on O_k beforehand is $h_k = 1 - (1 - p_k)^{\min(R_C, R_k)}$. We conclude:

$$h_{LRU,IRM}^{TTL,RpR} = \sum_{k=1}^{N} p_k (1 - (1 - p_k)^{\min(R_C, R_k)}). \quad (7)$$

Eq. (7) is equal to Fagin's approach [15], if $\forall k$: $R_k \geq R_C$.

4.3. LRU hit ratio with TTL and periodic reset

For periodic TTL resets we again have to regard both cases of TTL expiry after R_k and eviction after R_C requests as cache misses for O_k. Let $h_k(n)$ denote the probability that the next request to O_k is a hit, when $n \leq R_k$ is the remaining request count before expiry of O_k. We obtain:

$$h_k(n) = 1 - (1 - p_k)^n \qquad \text{if} \quad n \leq R_C;$$
$$h_k(n) = 1 - (1 - p_k)^{R_C} \qquad \text{if} \quad n \geq R_C.$$

In the case of periodic TTL resets, the process of TTL expiry is decoupled and independent of the request instants. Then in steady state, the time from a request to O_k until expiry is equally distributed between 0 and R_k. We conclude:

$$h_k = \sum_{n=0}^{R_k} \frac{h_k(n)}{R_k+1} = 1 - \frac{1 - (1 - p_k)^{R_k+1}}{p_k(R_k+1)} \quad \text{if } R_k \leq R_C;$$

$$h_k = 1 - \frac{1 + [(R_k+1-R_C)p_k - 1](1-p_k)^{R_C}}{p_k(R_k+1)} \quad \text{if } R_k \geq R_C.$$

Note that the formula for $R_k \leq R_C$ is equivalent to Eq. (3).

Finally, we obtain: $h_{LRU,IRM}^{TTL,PR} = \sum_{k=1}^{N} p_k h_k.$ (8)

5. Performance Evaluation

5.1. Effect of unique TTL on LFU & LRU hit ratios

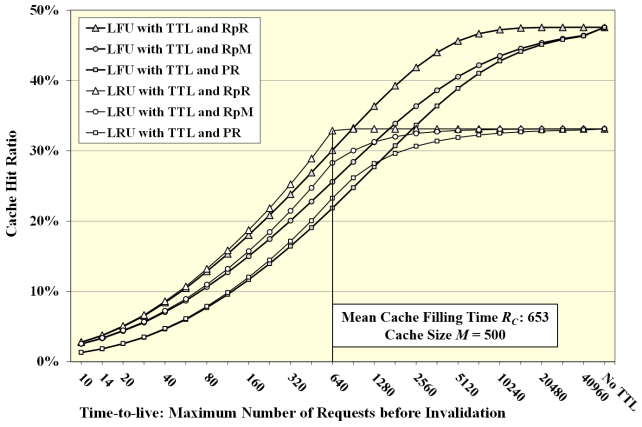

Figure 3: Impact of TTL on LFU and LRU caches

Figure 3 shows hit ratio results for an example of LRU and LFU caches with IRM Zipf distributed requests $p_k = \alpha\, k^{-\beta}$ with $\beta = 0.8$ among $N = 10\,000$ objects for a cache size $M = 500$. Zipf distributed requests with similar shape parameter β have been confirmed manifold for access to web data, usually with moderate correlation among the requests, where IRM represents a simplifying assumption not far from realistic pattern. We assume a unique TTL $R = R_k$ for resets of all objects. Reset per request leads to more resets, higher remaining TTL and thus higher hit ratio than reset per miss, while periodic reset has lowest hit ratios. LFU yields the maximum IRM hit ratio without TTL, but goes down below the LRU level for $R < R_C$.

The LRU stack is sorted by recent request times, i.e., due to the remaining TTL for RpR, when all objects reset to the same TTL R. Then objects are evicted before TTL expiry, if the maximum TTL exceeds R_C, and LRU is not affected by TTLs $R \geq R_C$ with RpR. Otherwise, if $R < R_C$, there are not enough valid objects to fill the LRU cache, i.e., only a smaller top part of the LRU cache is utilized corresponding to a reduced sojourn and convergence time R.

For a second evaluation case, we include FIFO, Random, an optimized score-based method and a hit ratio bound.

5.2. FIFO TTL analysis transfer from LRU results

We can only briefly summarize the FIFO hit ratio analysis approach with TTL constraints. The results are analogous to the LRU results, when we refer to the FIFO hit ratio approximation by Dan and Towsley [18] with characteristic time T_C^{FIFO}, which corresponds to Fagin's or Che's LRU approximation with characteristic and convergence time T_C. A new object is put on a FIFO stack for each cache miss, such that the sojourn time of an object until eviction in a FIFO cache is longer $T_C^{FIFO} \geq T_C$ without new requests, see the computation scheme in Eq. (5) of [17]. Since the FIFO cache is modified only per miss, FIFO sorts the objects due to TTL with RpM, whereas LRU provides sorting due to TTL with RpR. Based on $R_C = \lfloor \lambda T_C^{FIFO} \rfloor$, we conclude that the FIFO hit ratio result with TTL and RpM is provided by Eq. (7), whereas the FIFO hit ratio with TTL and RpR is derived according to the Markov chain result of Eq. (5-6), and Section 4.3 is also valid for FIFO with TTL and PR.

5.3. An optimized score-gated caching strategy

Finally, we adapt a score-gated caching (SGC) policy [13] to optimize the performance beyond the basic strategies. The SGC method prefers the top-M most popular objects as cache content like LFU. Moreover, a clock pointer is used to check if a top-M object is expired, which steps forward to a next object in the cache per miss. If the clock points to an (almost) expired object, it is replaced by the currently requested object, to provide a chance for new hits. Upon a request to an evicted top-M object, it is restored to the cache position given by its popularity rank. In this way, SGC combines and partly improves the best of the LFU and LRU performance over the entire TTL range, see Figure 4.

5.4. Upper bound of the IRM hit ratio with TTL

In order to check whether the SGC results are optimal, we introduce an upper bound on the hit ratio performance for

a cache of size M with TTL limits R_k per object and RpM (RpR, PR) reset policy. Under IRM request pattern, the optimum strategy puts the most popular objects into the cache, which are currently not expired. This principle can be realized only by partly reloading objects to the cache without a request, but we can analyse the optimum hit ratio.

Let p_k^{Valid} denote the probability that O_k is not expired. For TTLs with RpM we obtain $p_k^{Valid} = p_k R_k/(p_k R_k + 1)$, see Eq. (1). The phases when an object is valid or expired are independent for each object. Let $p_k^{\#Valid}(l)$ denote the probability that currently l out of the top-k most popular objects are valid. We assume the objects to be ordered due to popularity such that $p_1 \geq p_2 \geq \cdots \geq p_N$. All valid top-$M$ objects are put into the cache. For $k > M$, O_k is put into the cache, if O_k is valid and less than M of the top-(k-1) objects are valid, i.e. with probability $p_k^{Valid} \cdot \sum_{l=1}^{M-1} p_{k-1}^{\#Valid}(l)$. Finally, we compute $p_k^{\#Valid}(l)$ via a simple iterative scheme:

$$p_k^{\#Valid}(l) = p_{k-1}^{\#Valid}(l)(1-p_k^{Valid}) + p_{k-1}^{\#Valid}(l-1)p_k^{Valid}.$$

5.5. Hit ratio bound and performance of strategies

Figure 4 compares the results for LFU, LRU, FIFO, Random, SGC and the bound with a unique initial TTL R after resets and RpM for data consistency. We again consider an IRM Zipf distributed request pattern with $\beta = 0.6$ for 10 000 objects and a cache of size $M = 1000$.

Similar to the results in Figure 3, LFU outperforms LRU for large or no TTL ($R > R_C$), but LRU and FIFO yield higher hit ratio when the TTL is smaller ($R < R_C$). Random replacements are outperformed by LRU and FIFO over the entire range of R. FIFO keeps the hit ratio constant on the level without TTL for all TTLs $R > R_C$.

FIFO puts a new requested object on top of the cache for each miss. Consequently, a FIFO cache is sorted due to the remaining TTL for RpM and excludes only expired objects, if the TTL is longer than the characteristic time $R > R_C^{FIFO}$ [17]. The effect is similar to the LRU with TTL and RpR in Fig. 3, where LRU caches are sorted due to remaining TTL for RpR. Therefore the FIFO hit ratio partly can cope with LRU and partly outperforms LFU. However, for $R < R_C^{FIFO}$, expired objects in the cache are not put to the top upon a request, such that the sorting due to remaining RpM reset times is violated and FIFO becomes suboptimal.

SGC almost achieves the optimum in the TTL range below R_C, whereas a small gap of up to 2% remains for larger TTLs towards the hit ratio bound.

The results in the Figs. 3-4 were obtained by analysis as derived or summarized in the previous Sections, except for SGC and Random. Moreover, we checked them via simulation of each involved caching strategy, where only hardly noticable deviations below 1% are encountered for sufficiently long IRM request sequences. The analysis is exact for LFU and static caching as well as for the bound. For LRU and FIFO, the analysis is subject to small approximation errors of the approaches [15][16][18], which are shown to be limited to <1.3% for LRU and <3% for FIFO for $M \geq 10$ [17] with proven asymptotical exactness for $M \to \infty$ [19].

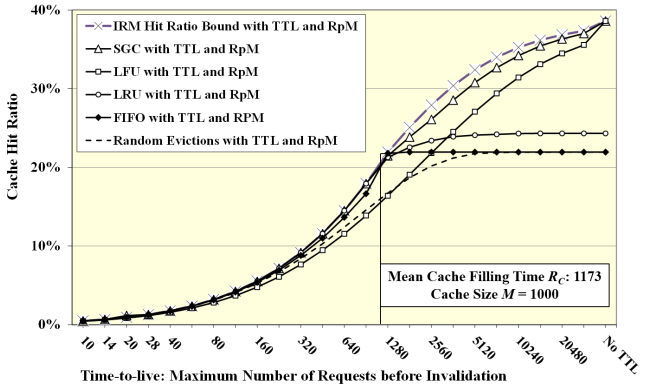

Figure 4: Impact of TTLs with RpM on different strategies

6. Extensions

The results are extensible in several ways, which are mostly left for future study. However, we address a first simple extension from constant reset interval length of R_k requests per object O_k to variable reset intervals. Then let $r_k^*(n)$ denote the fraction of reset intervals with n requests. Let $h^*(R_k)$ denote one of the previous analysis results of Eq. (1-8) which are derived per reset interval. The corresponding result h^* for variable R_k is then obtained as a weighted sum including all reset interval lengths $R_k = n$, regarding that the mean number of requests to O_k is proportional to n for IRM pattern:

$$h^* = \sum\nolimits_n n\, r_k^*(n)\, h^*(n) / \sum\nolimits_n n\, r_k^*(n). \qquad (9)$$

In Section 3 we mapped a TTL timer T into a request counter with fixed value $R_k = \lceil \lambda T_k \rceil$. Based on Eq. (9) we can extend the mapping for a varying number of requests, which may reflect e.g. a Poisson distribution $r_k^*(n)$ for the number of requests in the interval T with mean λT.

Further extensions can be considered

- for the analysis of the random eviction principle,
- for more evaluations with different TTLs per object,
- for caching of data of different size and value,
- for correlated request pattern beyond IRM and,
- for optimized score-based methods for all reset variants.

7. Conclusions

We have analysed the caching performance of basic strategies with TTL for data consistency and RpR, RpM and PR reset policies under IRM request pattern. Exact results are obtained for LFU and static caching. For LRU and FIFO, asymptotically exact hit ratio approximations retain their precision with regard to TTLs. LFU outperforms LRU and FIFO when the TTL exceeds the mean cache filling time, whereas LRU and partly also FIFO almost achieve the optimum hit ratio for smaller TTLs. A score-gated method for preferring cache content with high popularity and high remaining TTL count performs close to optimum for small TTLs and outperforms all basic strategies for large TTLs.

References

[1] P. Cao and C. Liu, "Maintaining strong cache consistency in the world wide web," *IEEE Trans. Computers*, vol. 47(4), pp. 445–457, 1998.

[2] E. Cohen, E. Halperin, and H. Kaplan, "Performance aspects of distributed caches using TTL-based consistency," *Proc. 28th ICALP, Crete, Greece*, pp. 744–756, 2001.

[3] Bundesministerium der Justiz, "Telemediengesetz (TMG)," *https://www.gesetze-im-internet.de/tmg/__9.html*, 2007.

[4] R. Fielding, M. Nottingham, and J. Reschke, "HTTP caching," *IETF Internet standards track, RFC 9111*, 2022.

[5] Q. Zheng et al., "On the analysis of cache invalidation with LRU replacement," *IEEE Trans. TPDS*, vol. 33/3, pp. 654–666, 2022.

[6] J. Jung, A. Berger, and H. Balakrishnan, "Modelling TTL-based Internet caches," *Proc. IEEE Infocom*, pp. 417–426, 2003.

[7] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy cache algorithms: Design, implementation, and performance," *IEEE Trans. Knowledge Data Engineering*, vol. 11(4), pp. 549–562, 1999.

[8] J. Yang, Y. Yue, and K. Rashmi, "A large-scale analysis of key-value cache clusters at Twitter," *ACM Trans. Storage*, vol. 17(3)17, 2021.

[9] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," *Perform. Eval.*, vol. 79, pp. 2–23, 2014.

[10] M. Dehghan et al., "A utility optimization approach to network cache design," *IEEE/ACM Trans. Networking*, pp. 1013–1027, 2019.

[11] G. Hasslinger et al., "An overview of analysis methods and evaluation results for caching strategies," *Computer Networks*, vol. 228, 2023.

[12] M. Garetto, E. Leonardi, and V. Martina, "A unified approach to the performance analysis of caching systems," *TOMPECS*, vol. 1(3)12, pp. 1–28, 2016.

[13] 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*, vol. 125, pp. 172–186, 2017.

[14] W. King III, "Analysis of paging algorithms," *Proc. IFIP Congress, Ljublanjana, Yugoslavia*, pp. 485–490, 1971.

[15] R. Fagin, "Asymptotic miss ratios over independent references," *J. Comput. Syst. Sci.*, vol. 14, no. 2, pp. 222–250, 1977.

[16] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: modeling, design and experimental results," *IEEE JSAC*, vol. 20(7), pp. 1305–1314, 2002.

[17] G. Hasslinger, K. Ntougias, F. Hasslinger, and O. Hohlfeld, "Scope and accuracy of analytic and approximate results for FIFO, clock-based and LRU caching performance," *Future Internet*, vol. 15/3, pp. 1–17, 2023.

[18] A. Dan and D. F. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," *Proc. ACM SIGMETRICS, Boulder, Colorado, USA*, pp. 143–152, 1990.

[19] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," *24th International Teletraffic Congress, ITC, Kraków, Poland*, pp. 1–8, 2012.