Utility-driven Optimization of TTL Cache Hierarchies under Network Delays

Karim S. Elsayed*, Fabien Geyer†, Amr Rizk*
*Faculty of Computer Science, University of Duisburg-Essen, Germany
† Airbus Central R&T, Munich, Germany

***Abstract*—We optimize hierarchies of Time-to-Live (TTL) caches under network delays. A TTL cache assigns individual eviction timers to cached objects that are usually refreshed upon a hit where upon a miss the object requires a random time to be fetched from a parent cache. Due to their object decoupling property, TTL caches are of particular interest since the optimization of a per-object utility enables service differentiation. However, state-of-the-art exact TTL cache utility-based optimization does not extend beyond single TTL caches, especially under network delays.**

In this paper, we leverage the object decoupling effect to formulate the non-linear utility maximization problem for TTL cache hierarchies in terms of the exact object hit probability under random network delays. We iteratively solve the utility maximization problem to find the optimal per-object TTLs. Further, we show that the exact model suffers from tractability issues for large hierarchies and propose a machine learning approach to estimate the optimal TTL values for large systems. Finally, we provide numerical and data center trace-based evaluations for both methods showing the significant offloading improvement due to TTL optimization considering the network delays.

I. Introduction

Caching has an essential role in reducing response times and bandwidth consumption by drawing frequently requested data objects closer to the request origin [1], [2]. As the demand for objects dynamically varies, service differentiation becomes an integral building block of the caching system. This is achieved through a caching utility abstraction [3] that gives rise to concepts such as caching fairness and cache resource allocation. A key approach to fine-grained cache optimization lies in controlling the utility attained *by individual objects*. This calls for systems built around *independently* tuning object-specific cache performance metrics.

Traditional and modern caches that use either fixed or data-driven, learned policies [4]–[6] usually run one or more decision rules that couple the occupancy of all the objects in the cache. For example, Least-recently-used (LRU) is a versatile and popular caching policy that is based on such coupling. In contrast, TTL caching decouples the object occupancy in the cache by assigning *independent* expiry/eviction time tags to the individual objects [7]. In a single TTL cache the aggregate utility of objects is maximized by deriving the object-specific optimal TTL values [3]. While the work in [3] is seminal, it is limited to a single cache within a set of restricting assumptions including, zero network delays and Poisson requests.

ISBN 978-3-903176-63-8 © 2024 IFIP

Caching systems usually consist of multiple caches connected to form a hierarchy [1]. We consider tree-like hierarchies representing systems such as content delivery networks (CDN) and data center caching. We note that optimizing cache hierarchies is hard as it involves *jointly modeling and optimizing* multiple interconnected caches under random network delays. It is also known that optimizing caches in isolation poorly utilizes the storage by allowing redundancy and ignoring the request process correlations [8].

Previous work on modeling and performance evaluation of TTL caching hierarchies offers computable models that are either exact-but-slow [9], [10] or approximate-but-fast [11], [12]. The work in [9] was the first to provide an exact Markov arrival process (MAP) model of a TTL caching tree under zero network latency. As empirically observed in [13], for a wide set of caching systems, the network delay significantly impacts the caching performance. The authors observe that the network delay ranges from being in the order of the inter-request times to being multiple orders of magnitude larger. Recently, the work in [10] extends the MAP model in [9] to model the random network delays within the hierarchy and derives the hierarchy hit probability. Additionally a subsequent work [14] derives the response time for the cache hierarchy MAP in [10].

In this paper, we optimize the aggregate utility of TTL caching hierarchies under non-zero network delays. Our contributions are:

- We formulate a utility maximization problem for optimizing TTL caching hierarchies under network delays to uniquely *leverage* the object decoupling effect of TTL caches. We provide an analytical solution to the non-linear utility maximization (Sect. III-IV), using the analytical performance metrics derived from the exact MAP model in [10].
- We provide a graph neural network (GNN) model to maximize the utility of the caching system for large hierarchies (Sect. V).
- We provide numerical evaluations of the analytical and the GNN utility maximization methods (Sect. VI).

An extended version of this work is available in [15]. Before we delve into the contributions, we first give an overview of the related work on the cache utility maximization in the next section.

II. Background & Related Work

A. Approaches to TTL cache modelling and optimization

Utility functions that are widely used in the context of network resource allocation [16], [17] have been recently adopted to caching models as a means to achieve service differentiation [3]. As the TTL policy decouples objects in the cache it is particularly well suited for utility maximization problems that control individual object performance metrics such as object hit probability and occupancy. Note that capacity-driven caching policies such as LRU or FIFO can be emulated using TTLs by linking the capacity to the expected cache occupancy [5], [9], [18].

For a single cache, the authors of [19] study linear utility maximization for different caching policies, while the authors in [20] consider the TTL utility maximization under heavy-tailed request processes and the works [21], [22] consider utilizing hit rates instead of probabilities for utility maximization. For hierarchies, the authors of [23]–[25] focus on optimizing the aggregate sum of the object utility of each cache in terms of content placement and request routing. Using variants of the fixed strategy Move-Copy-Down (MCD), the authors of [26] consider Poisson requests and use an approximation of the system hit probability, that, however, cannot capture the aggregation of different request streams, to optimize the overall utility of the hierarchy.

In contrast to analytical optimization models, learning-based approaches have been proposed for cache optimizations due to their lower computational complexity and ability to adapt to temporal changes. In addition, the authors of [27] proposed a DRL online caching approach as an alternative to the analytical approach in [3] for a single utility-driven TTL cache to achieve faster adaptation to changes of content popularities. In a TTL caching system that allows elastic adjustment of the cache storage size, a DRL-based algorithm can be used to optimize the system utility and the cost of storage together as outlined in [28].

Our work here differs fundamentally from the utility-driven caching in [3], [19]–[22], [27] as we consider the joint utility maximization for multiple caches in a hierarchy under random network delays. In contrast to these related works, we base our work on an *exact model of TTL cache hierarchies under random network delays*. Further, in comparison, the work in [26] maximizes the system utility of a simplified hierarchy model with decoupled paths and approximate system hit probability. In contrast, our work, first, takes random network delays into the optimization, and second, we build on the exact object hit probability throughout the hierarchy. Hence, instead of a fixed object placement such as MCD in [26] *our optimization yields the optimal object TTLs at every cache in the hierarchy*.

B. Primer on exact TTL cache hierarchy modelling

Next, we provide a quick review of the exact analysis of TTL caching hierarchies given random network link delays. A detailed treatment is found in [10]. We use bold symbols to denote vectors and matrices and non-bold symbols for scalars. Since TTL caches decouple objects, the model expresses the *object-specific hit probability* using a MAP that describes the state of an object of interest in the hierarchy. A MAP [29] is a point process described by two transition matrices; a hidden one $\mathbf{D}_0$, and an active one $\mathbf{D}_1$. Let the states associated with these matrices be denoted by the vector $\boldsymbol{S}$. Transitions that only control the jump process $J(t)$ are in $\mathbf{D}_0$, while $\mathbf{D}_1$ contains transitions which control $J(t)$ and also a counting process $N(t)$. Here, $N(t)$ counts the request misses from the entire cache hierarchy (so called system misses) until time t.

In general, the state space model is given as a composed MAP that is constructed of the MAPs of the request process, the TTL and delays, i.e., each of these three components can itself be described by a Markov arrival process [10]. The MAP describing the state of object i is, thus, defined over the state vector $\boldsymbol{S}_i$ consisting of $n_{s,i}$ states. To obtain the MAP associated with the object state in a tree-like cache hierarchy, the approach in [10] iteratively groups the individual cache MAPs using Kronecker superposition operations. The operation builds an equivalent MAP for multiple cache MAPs under topological constraints.

The steady state hit probability of an object i given the caching hierarchy MAP is expressed as [30]

$$P_i = 1 - \frac{\lambda_{Mi}}{\sum_{j=1}^{n_l} \lambda_{Iij}} = 1 - \frac{\boldsymbol{\pi}_i \mathbf{D}_{i,1} \mathbf{1}}{\sum_{j=1}^{n_l} \boldsymbol{\pi}_{ij}^{\{I\}} \mathbf{D}_{ij,1}^{\{I\}} \mathbf{1}}\,, \tag{1}$$

where λ_{Mi} and λ_{Iij} are the miss rate from the hierarchy, i.e. at the root, and the input request rate to the j-th leaf, respectively. Here, $\mathbf{D}_{i,1}$ and $\boldsymbol{\pi}_i$ are the active transition matrix and the steady state probability vector of the hierarchy MAP, respectively. Further, $\mathbf{D}_{ij,1}^{\{I\}}$ and $\boldsymbol{\pi}_{ij}^{\{I\}}$ are the active transition matrix and the steady state probability vector of the input request process at the jth leaf cache. The vector $\mathbf{1}$ is an all-one column vector.

Assuming stationary request processes, TTL sequences and delay processes, the steady state probability vector $\boldsymbol{\pi}_i$ is classically found as

$$\boldsymbol{\pi}_i = \boldsymbol{b}\mathbf{A}_i^{-1}\,, \tag{2}$$

where $b = [0, 0, ..., 1]$ and $\mathbf{A}_i = [(\mathbf{D}_{i,0} + \mathbf{D}_{i,1})_{:,n_{s,i}-1}|\mathbf{1}]$. Here, $(.)_{:,n_{s,i}-1}$ represent the first $n_{s,i} - 1$ columns, i.e, all the columns except for the last one. The operation $[\mathbf{X}|\mathbf{1}]$ concatenates the matrix $\mathbf{X}$ and the column vector $\mathbf{1}$.

The expected object occupancy $\mathbb{E}[O_i^k]$ of object i in cache k in the hierarchy is

$$\mathbb{E}[O_i^k] = \sum_{j \in \chi_i^k} \pi_{i,j}\,, \tag{3}$$

where $\pi_{i,j}$ is the jth element of $\boldsymbol{\pi}_i$. Here, χ_i^k is the set of indices of states in $\boldsymbol{S}_i$ where the object is stored in cache k.

Finally, note that it is evident from (2) and as highlighted in [10] that obtaining $\boldsymbol{\pi}_i$ is computationally intensive. With an increasing number of objects N and number of caches n_c the closed-form solution tractability is constrained by the matrix inversion $\mathbf{A}_i^{-1}$. Adopting the results from [10] a direct computation shows that this solution scales as $\mathcal{O}(N \cdot N_s^{\xi n_c})$ with ξ being the typical exponent for matrix inversion (e.g. from [31]) and N_s being the number of states of each individual cache MAP assuming homogeneity.

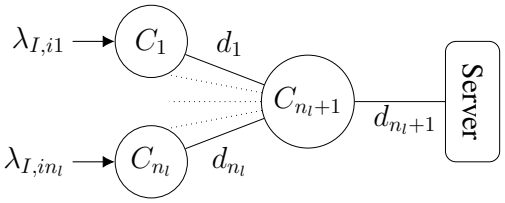

Fig. 1: Two level caching hierarchy tree consisting of n_l leaves. The random variable d_i encodes the delay for cache i to download the content from its parent. The server contains all objects. The request rate of object i at leaf j is denoted $\lambda_{I,ij}$.

III. System model and Problem statement

In this section, we formulate the utility maximization problem for TTL cache hierarchies under random delays and discuss different notions of utility functions. We consider a TTL caching tree with n_c caches of which n_l are leaf caches and the number of equally-sized objects in the system is N as depicted in Fig. 1. Exogenous object requests are only received via leaf nodes. Object retrieval from one cache to a parent cache takes a non-trivial, random download delay. We model each of the request processes, TTLs and the delays using a PH distribution. We define a request to a leaf cache as a *hit* when the object is in any of the caches along the path from that leaf to the root and as a miss otherwise. We use the approach outlined in the previous section to derive the MAP of the hierarchy.

A. Key insight and problem formulation

Next, we introduce our key idea for formulating the object utility given a cache hierarchy. We aim at maximizing the aggregate utility of objects in the cache hierarchy. This is achieved by controlling each object's overall *hierarchy hit probability* through tuning the corresponding object *TTL parameters at every cache* in the hierarchy.

In a cache hierarchy, requests to the same object arrive at heterogeneous leaf caches, where heterogeneity is in terms of leaf-specific object request processes as well as the object fetch delay distributions between child-parent caches. Hence, requests of an object at different leaf caches contribute to the overall object utility differently. The key idea of our formulation is to *decompose the object utility into its per input stream weighted utilities* and express it as

$$U_i = \sum_j \lambda_{Iij} \psi(P_{ij}(\boldsymbol{\mu}_i)) \ , \tag{4}$$

where P_{ij} is the hierarchy hit probability of an object i considering only the hits collected from the request stream to a leaf cache j, and the vector $\boldsymbol{\mu}_i$ denotes the parameters of the given TTL distributions of object i at all caches in the hierarchy. Here, ψ is a real-valued utility function, e.g. expressing α-fairness as considered later on, and λ_{Iij} is the request rate for object i at leaf cache j. Our formulation of the utility function in (4) coincides with the utility definition in [3] in case of a single cache. However, our formulation extends the utility calculation to tree-like cache hierarchies.

We formulate the optimization of the object TTLs within the cache hierarchy as the aggregate of the object utilities given in (4). We formulate the optimization problem as

$$\underset{\boldsymbol{\mu}_i}{\text{maximize}} \ \sum_i^N \sum_j^{n_l} \lambda_{Iij} \psi(P_{ij}(\boldsymbol{\mu}_i)) \ , \tag{5}$$

$$\text{subject to} \sum_i^N \mathbb{E}[O_i^k] = B_k \ \ \forall k \ . \tag{6}$$

Here, $\mathbb{E}[O_i^k]$ is the expected occupancy of object i at cache k from (3) and B_k denotes the size of cache k measured in the number of objects the cache can hold.

Standard analytical TTL models such as [3], [5], [9], [18] assume an infinite storage capacity and use the constraint (6) on the expected occupancy to introduce a separate cache capacity constraint. In this basic formulation of the optimization problem, we take the same approach. We note that this constraint is not sufficient to prevent a storage capacity overflow or in other words a mismatch between the closed-form solution of (5), (6) and an algorithmic realization using any finite cache size. However, as discussed in [3], this mismatch diminishes for large number of objects in the systems.

As discussed in Sect. I, TTL caching has the advantage of tuning the hit probability of each object independently. Thus, we aim to find the object TTLs at each cache that maximize the overall utility of the cache hierarchy.

B. Utility and fairness in cache hierarchies

We adopt the α-fair utility function proposed in [32], which groups different notions of fairness for resource allocation. We use the α-fair utility function

$$\psi(P) = \begin{cases} \frac{P^{1-\alpha}}{1-\alpha} \ \alpha \geq 0, \ \alpha \neq 1 \\ \log P, \ \alpha = 1 \end{cases} \tag{7}$$

where P is the hit probability. The α-fair utility function is used in the context of resource allocation to unify different notions of fairness [17]. By letting $\alpha \in \{0, 1\}$ or $\alpha \to \infty$, the α-fair utility can represent the notion of fairness ranging from throughout (also called offloading) maximization (no fairness, $\alpha = 0$) and proportional fairness ($\alpha = 1$) to max-min fairness as $\alpha \to \infty$. By solving the optimization function (5), (6) we obtain the TTL values that maximize the desired utility.

IV. Interior point utility maximization for TTL cache hierarchies under random Network Delays

In this section, we derive the solution to the utility maximization problem defined in Sect. III. In the appendix VII-A we derive a closed form solution for the optimal TTL values for a *single* cache and illustrate the hardeness of analytically solving the optimization problem for cache hierarchies in the appendix VII-B.

A. Non-linear inequality constrained optimization

The interior-point optimization method is used to solve non-linear optimization problems of the form of

$$\underset{\boldsymbol{x}}{\text{minimize}} \ f(\boldsymbol{x}) \ , \ \text{subject to} \ \ c(\boldsymbol{x}) = 0 \ , \ \boldsymbol{x} > 0 \ ,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, and the vector-valued function $c : \mathbb{R}^n \to \mathbb{R}^m$ contains m equality constraints. The interior point approach uses barrier functions to embed the inequality constraints into the objective function [33] such that the problem can be rewritten as

$$\underset{\boldsymbol{x}}{\text{minimize}}\ f(\boldsymbol{x}) - \eta \sum_i \ln(x_i)\ , \text{subject to}\ \ c(\boldsymbol{x}) = 0\ ,$$

where x_i is the ith element of the vector $\boldsymbol{x}$. The key idea behind the barrier function is to penalize the objective function when approaching the inequality constraint. Here, we show the logarithmic barrier function where the penalty of x_i approaching the constraint at 0 is $-\infty$. In addition, the parameter η controls the scale of the penalty, where the solution to the problem for a decreasing value of η approaches the solution to the original problem. This requires solving the problem multiple times after reducing the value of η gradually. At each optimization step (superscript (i)), the value of η is calculated for the next iteration ensuring a super-linear convergence by [34]

$$\eta^{(i+1)} = \max\left\{\frac{\epsilon_{\text{tol}}}{10}, \min\left\{\kappa\eta^{(i)}, {\eta^{(i)}}^{\theta}\right\}\right\}\ ,$$

where ϵ_{tol} is the user defined error tolerance, $\kappa \in [0, 1]$ and $\theta \in [1, 2]$. As a result, the value of η decreases exponentially as the optimization gets closer to convergence.

For a fixed η, the optimization problem is solved by applying Newton's method to the Lagrangian

$$\mathcal{L} = f(\boldsymbol{x}) + c(\boldsymbol{x})^T\boldsymbol{\nu} - \boldsymbol{x}^T\boldsymbol{z}\ .$$

Given the Lagrangian multipliers for the equality and inequality constraints, i.e., $\boldsymbol{\nu}$ and $\boldsymbol{z}$, respectively, the method solves the problem iteratively for the steps to the minimum ${\boldsymbol{\delta_x}}^{(m)}$, ${\boldsymbol{\delta_\nu}}^{(m)}$ and ${\boldsymbol{\delta_z}}^{(m)}$ at iteration m to update the value of the decision variables and the Lagrangian multipliers [34] according to

$$\begin{aligned}\boldsymbol{x}^{(m+1)} &= \boldsymbol{x}^{(m)} + \alpha^{(m)}{\boldsymbol{\delta_x}}^{(m)}\\ \boldsymbol{\nu}^{(m+1)} &= \boldsymbol{\nu}^{(m)} + \alpha^{(m)}{\boldsymbol{\delta_\nu}}^{(m)}\\ \boldsymbol{z}^{(m+1)} &= \boldsymbol{z}^{(m)} + \alpha_z^{(m)}{\boldsymbol{\delta_z}}^{(m)}\ .\end{aligned}$$

where, the step sizes α and $\alpha_z \in [0, 1]$. Note that for a more flexible and less restrictive update of the variables, different step sizes are chosen for $\boldsymbol{z}$ than $\boldsymbol{x}$. In addition, in order to ensure that the values of $\boldsymbol{x}$ and $\boldsymbol{z}$ are positive at each iteration [34], $\alpha^{(m)}$ and $\alpha_z^{(m)}$ are calculated with respect to a fraction to the boundary value $\varepsilon^{(i)} := \max\{\varepsilon_{\min}, 1 - \eta^{(i)}\}$ as

$$\begin{aligned}\alpha^{(m)} &\le \max\{\alpha \in [0, 1] : \boldsymbol{x}^{(m)} + \alpha{\boldsymbol{\delta_x}}^{(m)} \ge (1 - \varepsilon)\boldsymbol{x}^{(m)}\}\\ \alpha_z^{(m)} &= \max\{\alpha \in [0, 1] : \boldsymbol{z}^{(m)} + \alpha{\boldsymbol{\delta_z}}^{(m)} \ge (1 - \varepsilon)\boldsymbol{z}^{(m)}\}\ ,\end{aligned} \tag{8}$$

where $\varepsilon_{\min} \in [0, 1]$. Furthermore, choosing a specific value of $\alpha^{(m)} \in [0, \alpha_{\max}^{(m)}]$ from (8) by applying a backtracking line search method ensures sufficient progress towards the optimal solution. Using a variant of Fletcher and Leyffer's filter method [35], the authors of [36] proved under mild conditions that the interior point achieves global convergence.

The optimal steps ${\boldsymbol{\delta_x}}^{(m)}$, ${\boldsymbol{\delta_\nu}}^{(m)}$ and ${\boldsymbol{\delta_z}}^{(m)}$ at iteration m are calculated by solving

$$\begin{bmatrix}\mathbf{H}^{(m)} & \mathbf{J}^{(m)} & -\mathbf{I}\\ {\mathbf{J}^{(m)}}^T & \mathbf{0} & \mathbf{0}\\ \mathbf{Z}^{(m)} & \mathbf{0} & \mathbf{X}^{(m)}\end{bmatrix}\begin{bmatrix}{\boldsymbol{\delta_x}}^{(m)}\\ {\boldsymbol{\delta_\nu}}^{(m)}\\ {\boldsymbol{\delta_z}}^{(m)}\end{bmatrix} = -\begin{bmatrix}\boldsymbol{\Gamma}^{(m)}\\ c(\boldsymbol{x}^{(m)})\\ \mathbf{X}^{(m)}\mathbf{Z}^{(m)}\mathbf{1} - \eta\mathbf{1}\end{bmatrix}\ , \tag{9}$$

where $\mathbf{H}$ is the Hessian of the Lagrangian $\mathcal{L}$ defined as

$$\mathbf{H} := \nabla^2_{xx}\mathcal{L} = \nabla_{xx}f(\boldsymbol{x}) + \sum_k \nu_k\nabla_{xx}c_k(\boldsymbol{x})$$

and using the shorthand matrix $\boldsymbol{\Gamma}^{(m)} := \nabla f(\boldsymbol{x}^{(m)}) + \mathbf{J}^{(m)}\boldsymbol{\nu}^{(m)} - \boldsymbol{z}^{(m)}$, where $\mathbf{J} := \nabla c(\boldsymbol{x})$ is the Jacobian matrix of the equality constraints. Here, $\mathbf{X}^{(m)}$ and $\mathbf{Z}^{(m)}$ are diagonal matrices of $\boldsymbol{x}^{(m)}$ and $\boldsymbol{z}^{(m)}$, respectively.

B. Utility maximization for TTL cache hierarchies

Next, we first apply the interior point optimization to solve the TTL cache utility maximization (5),(6). We express the objective function $f(\boldsymbol{x})$ and the constraints $c_k(\boldsymbol{x})$ as

$$\begin{aligned}f(\boldsymbol{x}) &= -\sum_i\sum_j \lambda_{Iij}\psi(P_{ij}(\boldsymbol{\mu}_i))\\ c_k(\boldsymbol{x}) &= \sum_i \mathbb{E}[O_i^k] - B_k,\ \forall k \in \{0, .., n_c\}\end{aligned} \tag{10}$$

The decision column vector $\boldsymbol{x}$ contains *the TTL parameters of each object at each cache* and controls the aggregate utility of the hierarchy. We express $\boldsymbol{x}$ in terms of the TTL parameters of each object as

$$\boldsymbol{x} = [\boldsymbol{\mu}_1^T,\ \boldsymbol{\mu}_2^T, \ldots, \boldsymbol{\mu}_N^T]^T$$

We calculate P_{ij} analytically from the MAP of object i as

$$P_{ij} = 1 - \frac{\lambda_{Mij}}{\lambda_{Iij}} = 1 - \frac{\boldsymbol{\pi}_i\mathbf{D}_{ij,1}\mathbf{1}}{\boldsymbol{\pi}_{ij}^{\{I\}}\mathbf{D}_{ij,1}^{\{I\}}\mathbf{1}}\ , \tag{11}$$

where only the miss rate λ_{Mij} of the requests to leaf cache j is considered[1]. This is calculated using the active transition matrix $\mathbf{D}_{ij,1}$ containing only the transitions in $\mathbf{D}_{i,1}$ that generate misses due to a request to leaf cache j.

Next, we derive the gradients $\nabla_{xx}f(\boldsymbol{x})$, $\nabla f(\boldsymbol{x})$, $\nabla_{xx}c(\boldsymbol{x})$ and $\nabla c(\boldsymbol{x})$ for the cache utility maximization problem to calculate $\mathbf{H}$, $\mathbf{J}$ and $\boldsymbol{\Gamma}$ from (9), to obtain the optimal search directions. We formulate the gradient and Hessian of $f(\boldsymbol{x})$ with respect to the TTL parameters of each object $\boldsymbol{\mu}_i$. The key to using interior point optimization is that TTL caching decouples the objects in the cache, i.e, the hit probability of an object i only depends on its TTL parameters $\boldsymbol{\mu}_i$, and *thus we can express* $\nabla f(\boldsymbol{x})$ *and* $\nabla_{xx}f(\boldsymbol{x})$ *as*

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix}\nabla^T_{\boldsymbol{\mu}_1}f(\boldsymbol{x}) & \nabla^T_{\boldsymbol{\mu}_2}f(\boldsymbol{x}) & \ldots & \nabla^T_{\boldsymbol{\mu}_N}f(\boldsymbol{x})\end{bmatrix}^T \tag{12}$$

$$\nabla_{xx}f(\boldsymbol{x}) = \begin{bmatrix}\nabla_{\boldsymbol{\mu}_1\boldsymbol{\mu}_1}f(\boldsymbol{x}) & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \nabla_{\boldsymbol{\mu}_N\boldsymbol{\mu}_N}f(\boldsymbol{x})\end{bmatrix} \tag{13}$$

Similarly, $\nabla c(\boldsymbol{x})$ and $\nabla_{xx}c(\boldsymbol{x})$ are represented as in (12) and (13), which we do not show here for space reasons.

Recall that the object hit probability (11) and the expected occupancy (3) are calculated using the steady state probabilities $\boldsymbol{\pi}_i$, which in turn depend on the transition matrices of the

[1]Note from the formulation that the hierarchy hit probabilities per-leaf request stream are *not mutually independent* as $\boldsymbol{\pi}_i$ is a function of all the parameters of the MAP

hierarchy MAP. Therefore, we next derive $\nabla f(\boldsymbol{x})$, $\nabla_{xx} f(\boldsymbol{x})$, $\nabla c(\boldsymbol{x})$ and $\nabla_{xx} c(\boldsymbol{x})$ in terms of the derivatives of the steady state vector of each object i with respect to its TTL parameters.

Given the TTL parameters a, b each representing any of the TTL parameters in $\boldsymbol{\mu}_i$ of an object i. From (2) the derivatives $\boldsymbol{\pi}_i^{a'} := \frac{\partial \boldsymbol{\pi}_i}{\partial a}$ and $\boldsymbol{\pi}_i^{a'b'} := \frac{\partial^2 \boldsymbol{\pi}_i}{\partial a \partial b}$ are given as

$$\begin{aligned} \boldsymbol{\pi}_i^{a'} &= -\boldsymbol{b}\mathbf{A}_i^{-1}\mathbf{A}_i^{a'}\mathbf{A}_i^{-1} \\ \boldsymbol{\pi}_i^{a'b'} &= [\mathbf{A}_i^{a'}\mathbf{A}_i^{-1}\mathbf{A}_i^{b'} + \mathbf{A}_i^{b'}\mathbf{A}_i^{-1}\mathbf{A}_i^{a'}]\mathbf{A}_i^{-1} \end{aligned} \tag{14}$$

Theorem 1. *The derivatives $f^{a'} := \frac{\partial f(\boldsymbol{x})}{\partial a}$ and $f^{a'b'} := \frac{\partial^2 f(\boldsymbol{x})}{\partial a \partial b}$ associated with the TTL cache objective function* (5) *are given in terms of the derivatives of the steady state vector $\boldsymbol{\pi}_i^{a'}$ and $\boldsymbol{\pi}_i^{a'b'}$ from* (14) *as*

$$f^{a'} = \sum_j \psi' \boldsymbol{\pi}_i^{a'} \mathbf{D}_{ij,1}\mathbf{1} \,, \tag{15}$$

$$f^{a'b'} = \sum_j \left[\lambda_{Iij}\psi' \boldsymbol{\pi}_i^{a'b'} - \psi'' \boldsymbol{\pi}_i^{b'} \mathbf{D}_{ij,1}\mathbf{1}\boldsymbol{\pi}_i^{a'}\right] \frac{\mathbf{D}_{ij,1}\mathbf{1}}{\lambda_{Iij}} \tag{16}$$

where ψ' is the derivative of ψ with respect to P_{ij}.

The proof of Thm. 1 is in the appendix VII-C

Now, the equality constraint is expressed in terms of the steady state probability vector according to (3). Therefore, we calculate its first and double differentiation as

$$\begin{aligned} c_k^{a'}(\boldsymbol{x}) &= \frac{\partial \mathbb{E}[O_i^k]}{\partial a} = \sum_{l \in \chi_i^k} \pi_{i,l}^{a'} \,, \\ c_k^{a'b'}(\boldsymbol{x}) &= \frac{\partial^2 \mathbb{E}[O_i^k]}{\partial a \partial b} = \sum_{l \in \chi_i^k} \pi_{i,l}^{a'b'} \,, \end{aligned} \tag{17}$$

where $\pi_{i,l}^{a'}$ and $\pi_{i,l}^{a'b'}$ are the l-th elements of $\boldsymbol{\pi}_i^{a'}$ and $\boldsymbol{\pi}_i^{a'b'}$, respectively. We can now use (14) and Thm. 1 to calculate $\nabla_x f(\boldsymbol{x})$ and $\nabla_{xx} f(\boldsymbol{x})$. Similarly, we use (14), (17) to calculate $\nabla_{xx} c(\boldsymbol{x})$ and $\nabla_x c(\boldsymbol{x})$. As a result, we obtain $\mathbf{H}$, $\mathbf{J}$ and $\boldsymbol{\Gamma}$.

V. Learning on Graph transformations of TTL Cache hierarchies

In contrast to the closed-form solution presented in Sect. IV, we propose next a learning-based approach to predict the optimal TTLs. Our rationale for using GNNs is that these can process graphs of large sizes, circumventing the scaling limitation of the closed-form optimization. We evaluate the execution time of both methods in Sect. VI.

Next, we present our neural network architecture and data transformation used for optimizing the TTL configuration of the caching hierarchy. The goal is to enable an agent to process the caching hierarchy and its properties (e.g. structure, cache capacities, request arrival rates) and predict the optimal TTL to use for each object at each cache. Our approach is based on a transformation of the caching hierarchy into a graph data structure, which is then processed using a GNN.

A. A GNN approach

We use the framework of GNNs introduced in [37], [38]. These are a special class of neural networks (NNs) for processing graphs and predicting values for nodes or edges depending on the connections between nodes and their properties. Fundamentally, GNNs utilize *message passing* where messages are updated and passed between neighboring nodes. Essentially, these messages are vectors $\mathbf{h}_v \in \mathbb{R}^k$ (here $k = 2^7$) that are propagated throughout the graph over multiple iterations. This principle is depicted in Fig. 2 for an examplary graph. We refer to [39] for a formalization of GNN concepts.

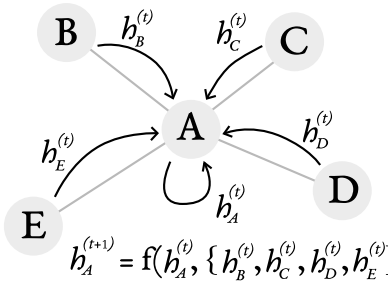

Fig. 2: Illustration of the message passing principle of GNNs. The message of node A is updated with f, a trainable function processing the current message and the set of messages from the neighbors. The same process is applied for each node in the graph.

We select gated graph neural network (GGNN) [40] as GNN model (i.e. function f in Fig. 2), with the addition of edge attention. For each node v in the graph, its message $\mathbf{h}_v^{(t)}$ is updated at iteration t as

$$\mathbf{h}_v^{(t=0)} = FFNN_{init}(\iota_v) \tag{18}$$

$$\mathbf{h}_v^{(t)} = GRU\left(\mathbf{h}_v^{(t-1)}, \sum_{u \in q(v)} \lambda_{(u,v)}^{(t-1)} \mathbf{h}_u^{(t-1)}\right) \tag{19}$$

$$\lambda_{(u,v)}^{(t)} = \sigma\left(FFNN_{edge}\left(\left\{\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}\right\}\right)\right) \tag{20}$$

$$\mathrm{o} = FFNN_{out}(\mathbf{h}_v^{(d)}) \tag{21}$$

with ι_v and o being the input features vector at node v and the output vector (i.e. predictions of the GNN), respectively. Also $\sigma(x) = 1/(1+e^{-x})$ denotes the sigmoid function, $q(v)$ the set of neighbors of node v, $\{a, b\}$ the concatenation operator of vectors a and b, GRU a gated recurrent unit (GRU) cell, $FFNN_{(\cdot)}$ are feed-forward neural networks (FFNNs), and $\lambda_{(u,v)} \in (0, 1)$ being the (trainable) weight for the edge (u, v).

The messages are initialized by a FFNN in (18) according to the input features vector of the nodes. The messages updates (illustrated in Fig. 2) are done with (19) for d iterations, with d corresponding to the diameter of the analyzed graph, where a weight is assigned to each edge according to the messages of its two nodes. The attention mechanism is described in (20). Finally, we obtain the prediction for each node by a FFNN with (21).

B. Graph transformation

Next, we use the graph induced through the hierarchy, where nodes are caches and edges are links, as an undirected graph data-structure that is processed using the GNN. Each node has an input vector of fixed size describing its features.

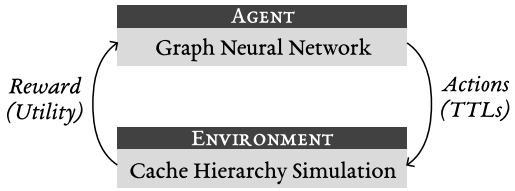

Fig. 3: Illustration of the reinforcement learning (RL) training approach

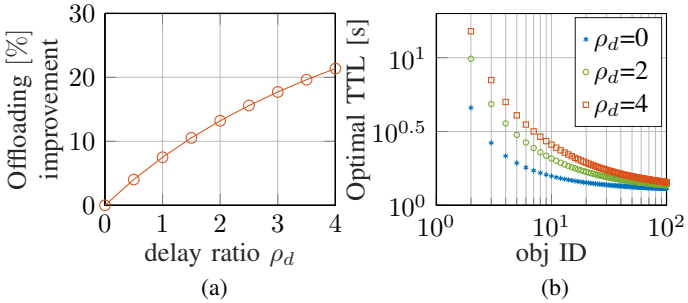

Fig. 4: **Single cache results:** TTL optimization under random network delay vs. the idealized zero-delay assumption where ρ_d denotes the ratio of the expected network delay to the expected inter-request time of the hottest object. (a) Significant Improvement of origin offloading, i.e. traffic served by the cache, due to optimizing under delays ($\text{OPT}|_{\text{delay}}$) vs. idealization ($\text{OPT}|_{\text{ideal}}$) increases with the delay. (b) The optimal TTL values are larger for $\text{OPT}|_{\text{delay}}$ especially for hottest objects. Object IDs sorted in descending arrival rate.

Specifically, we use the (i) cache type, i.e. leaf or non-leaf node using binary encoding, (ii) average network delay to the parent cache, and (iii) cache capacity. Additionally, the following features are used for the leaves: (iv) sum of expected inter-arrival time for all the objects, (v) expected inter-arrival time of requests for the top-M objects, and (vi) expected inter-arrival time for the remaining $M+i$ objects.

For each node, an output vector of fixed size is predicted, with the following features: (a) M TTL values to use for the top-M objects, (b) A default TTL value for the remaining $M+i$ objects, (c) the index of the node where each object should be cached in the hierarchy. The last restriction stems from the efficiency goal to cache an object only at a single cache along each leaf-root path.

C. Training approach

In order to train the GNN, we use RL using the `REINFORCE` policy gradient algorithm [41]. We use a basic training loop illustrated in Fig. 3, where the TTL configuration and the object locations predicted by the GNN are used as input configuration for a cache simulator. At the end of a simulation run, the empirical object hit probabilities are computed and fed back to the training loop as reward.

Instead of simply returning a single reward value (of the objective function), we return a reward vector corresponding to the utility for each object in the cache. We observe that this detailed reward function results in a performance enhancement of the trained GNN.

Another detail to take into account is the $\log$ function used in the proportional fairness utility function (7). Since the GNN might predict a TTL configuration leading to having a zero cache hit for given objects, it would lead to a reward value of $-\infty$ for the RL training process. This occurs frequently during the first epochs of the training, as the GNN does not predict yet good TTLs. Additionally, only negative values are produced since the hit rate is in the $[0,1]$ interval. To overcome these issues, we replaced the $\log$ function in (7) with its Taylor approximation and adding its value at zero:

$$\psi(P) \coloneqq \sum_{k=1}^{K} \left((-1)^{k+1} \frac{(P-1)^k}{k} - \frac{1}{k} \right) \tag{22}$$

This approximation enables us to have positive rewards. For the numerical evaluations presented later, we selected $K = 30$.

Finally, we sample from a Student's t-distribution during the `REINFORCE` algorithm for exploring the action space of the TTL values as the heavier tail achieves a better exploration of the possible TTLs configuration. This came from the realization that a fat-tailed distribution was required in order to achieve a better exploration of the possible TTLs configuration.

VI. Evaluations and Lessons Learned

Next, we show numerical performance evaluations of the proposed cache utility maximization approaches. If not stated otherwise we consider the two-level caching tree in Fig. 1 and use the proportional fairness parametrization ($\alpha = 1$) of the utility function (7). The request rates at the leaf caches follow a Zipf distribution, i.e., the rate of index j is $\lambda_j = \frac{1}{j^s}$ and $s = 0.8$ as observed in [42]. Note that for a given object the request rates are inhomogeneous across the leaves, i.e., an object i is assigned an index j at each leaf uniformly at random. Empirical values stem from long enough emulations of at least 5×10^5 requests to generate enough events for cold objects. For illustration, we let the inter-request times, network delays and TTLs to be exponentially distributed and illustrate the impact of the network delays on the optimal utility. As noted in Sect. II-B it is simple to include these three model components as MAPs into Thm. 1. We use Pytorch [43] to train the GNN and the evaluations run on a 24-core machine with 64 GB RAM.

A. Single cache optimization under random network delays

First, we consider a single cache under random network delays to the origin server. We use $N = 100$ objects under Zipf-popularity and capacity $B = 10$. Fig. 4a shows a strong improvement in a classical cache metric, i.e., the origin offloading, when optimizing the object TTLs considering the random network delay compared to ignoring it. The origin offloading is the fraction of request traffic served by the cache and it is obtained from the utility (7) by $\alpha = 0$. Here, we vary the ratio ρ_d of the expected network delay to the expected inter-request time of the hottest object based

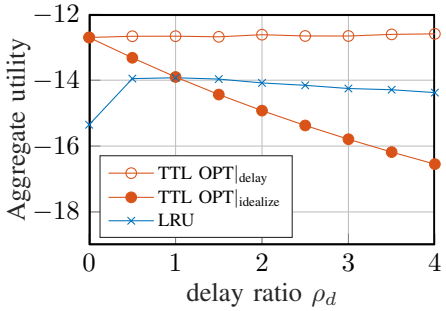

Fig. 5: **Two-level binary tree:** Aggregate utility of the analytical optimization under delay ($\text{OPT}|_{\text{delay}}$) vs. the idealized zero delay optimization ($\text{OPT}|_{\text{ideal}}$) vs. LRU.

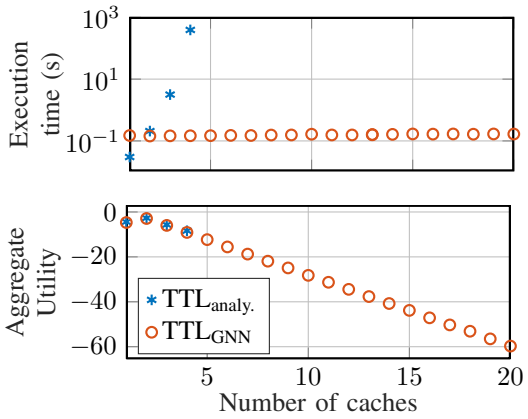

Fig. 6: **Scaling the number of caches:** Optimized aggregate utility and execution times of the analytical and the GNN-based TTL optimizations. The analytical optimization becomes intractable while the GNN provides good results (albeit less than the analytical optimization) at a fraction of the execution time.

on the arguments in [10], [13]. One manifestation of the observed improvement are the *per-object optimal TTLs* that are higher when accounting for the random delays as shown in Fig. 4b. Observe that the possible loss in hit probability, consequently utility, due to misses during the object fetch delay is compensated by the longer TTLs.

B. Hierarchy optimization under random network delays

Next, we show the influence of optimizing the hierarchy caches together compared to using traditional algorithms such as LRU throughout the hierarchy. We parameterized the hierarchy with $N = 100$ objects and a storage of $B = 5$ per cache. Fig. 5 shows that in contrast to the detrimental impact of the increasing delay on the idealized optimal utility assuming zero delay (denoted $\text{OPT}|_{\text{ideal}}$), we can maintain the optimal utility for varying the delay ratio ρ_d using the optimization from Sec. IV-B (denoted $\text{OPT}|_{\text{delay}}$). Observe that the aggregate utility remains higher than that of LRU which is utility agnostic. which lacks cache cooperation and, hence, stores objects redundantly in the hierarchy.

C. Scaling with the hierarchy size: A GNN to the rescue

Fig. 6 compares the performance of cache hierarchy emulations using the GNN approach and using the analytical TTL optimization for $N = 30$ number of objects, a capacity of $B = 4$ for each cache and delay ratio $\rho_d = 4$. We increase the total number of caches n_c in the two level caching tree (cf. Fig. 1) with $n_c = 1$ representing a single cache and $n_c = 2$ denoting one parent and one child cache. We observe in Fig. 6 that as we increase the number of caches, the execution time of the optimal TTL optimization increases exponentially (cf. the discussion in Sect. II-B). We, hence, show that utilizing the GNN-based approach from Sect. V provides very good results (albeit not analytically proven optimal) while maintaining a constantly small execution time. We note that the analytically computed optimal utility is slightly higher than the GNN-based utility, which is not clearly observed in Fig. 6 due to the scale of the utility axis. Note that this comparison is consistent as the GNN is trained on the same cache hierarchy.

The optimal utility decreases as the number of leaf caches increases as observed in Fig. 6. By increasing the number of the leaf caches, the total request rate to the hierarchy increases while the number of shared caches among the paths, i.e., the parent cache, remains fixed.

D. Trace-based evaluation

Next, we describe a trace-based evaluation of the presented TTL optimization procedure. We use a data center trace [44] having 10^6 requests to more than 8×10^4 objects as input to a single cache. We optimize the TTL cache proportional fairness utility ($\alpha = 1$). As the trace contains cold transient objects, we only consider for the TTL optimization the most frequent objects receiving each at least 15 requests (overall 580 objects) using the analytical method in Sect. IV and do not cache the colder objects (set their TTLs to zero). We estimate and verify the mean request rates of each object assuming exponentially distributed inter-request times. As the objects may, however, receive requests only for a certain duration within the trace, we use the average number of requests per object ω_i to gauge (24) as $\sum_i^N \sum_j^{n_l} \omega_i \psi(P_{ij}(\boldsymbol{\mu}_i))$. We run simulations using the optimal TTLs vs. vanilla LRU, FIFO and Random caching for a fixed cache size of 50 objects and vary the expected network delay relative to the mean inter-request time. Fig. 7 shows the strong improvement in the amount of traffic served by the cache (origin offloading) due to optimizing the object TTLs under random network delays compared to vanilla LRU, FIFO and Random caching.

VII. Conclusion

We study the optimization of TTL cache hierarchies under random network delays. By leveraging object decoupling in the exact TTL cache model we analytically solve the non-linear utility maximization problem for cache hierarchies and find the optimal per-object TTL values. As the optimal solution inherits tractability issues from the exact model, we propose a GNN-based approach that scales with the hierarchy size. Numerical and trace-based evaluations of both methods show

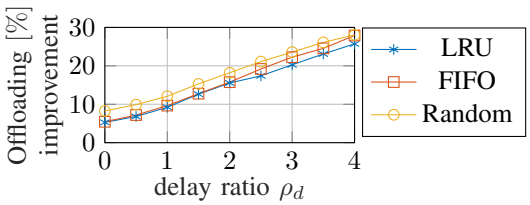

Fig. 7: **Trace-based evaluation**: Origin offloading of the optimal TTL relative to LRU, FIFO and Random caching for increasing delay.

strong caching performance improvements when incorporating the network delay into the cache hierarchy optimization.

APPENDIX

A. Straw man: Maximizing the utility of a single cache under exponential distributions

We derive a closed form solution to the utility maximization of a simple single cache where the mutually independent inter-request times, the TTL and the fetching (network) delays are each independently and identically distributed (iid), specifically, exponentially distributed, with parameters $1/\lambda_i$, $1/\mu_i$ and $1/\mu_F$, respectively. The object hit probability and occupancy are obtained from (1)-(3) as

$$P_i = \mathbb{E}[O_i] = \frac{\lambda_i \mu_F}{\mu_i(\mu_F + \lambda_i) + \lambda_i \mu_F} \tag{23}$$

Note that $P_i = \mathbb{E}[O_i]$ is only valid for memoryless inter-request time, TTL and delay distributions. Now, achieving maximum utility is equivalent to optimizing P_i, hence, calculating the corresponding optimal TTL parameter μ_i in (23). The optimization function (5)-(6) becomes

$$\underset{P_i}{\text{maximize}} \sum_i^N \lambda_{Ii}\psi(P_i) \ , \ \text{subject to} \sum_i^N P_i = B \ , \tag{24}$$

with P_i as a decision variable for this cache and $\lambda_{Ii} = \lambda_i$ for the exponentially distributed inter-request times. The optimization function as expressed in (24) coincides with the one proposed in [3] for a single cache with delays. Note, however, that a difference exists in the relation between the hit probability and TTL that depends in our single cache model on the delay as expressed within (24). The following result can also be obtained from [3].

The optimal hit probability for the example above is $P_i = \psi'^{-1}(\beta/\lambda_i)$ with $\sum_i \psi'^{-1}(\beta/\lambda_i) = B$ where $\psi'^{-1}(\beta/\lambda_i)$ is the inverse function of the derivative of the utility with respect to P_i. Here, P_i is expressed in terms of β which is implicitly given and numerically obtained when a given utility function is inserted above. For example, in the case of proportional fairness utility ($\alpha = 1$) the optimal hit probability is $P_i = \frac{\lambda_i}{\sum_i \lambda_i} B$. Consequently, the optimal TTL for this simple single cache with delay is calculated from (23) as

$$\hat{\mu_i} = \frac{1}{\mu_F + \lambda_i}\left(\frac{\mu_F \sum_i \lambda_i}{B} - \lambda_i \mu_F\right) \ .$$

B. Why maximizing the utility of a cache hierarchy is hard?

The approach illustrated above of finding the optimal object hit probabilities P_i that maximize the utility and, hence, calculating the corresponding optimal TTL parameters μ_i is feasible under two conditions: (a) the object hit probability is a bijective function of the TTL parameter (b) the object occupancy function is a composite function $g(P_i(\boldsymbol{\mu}_i))$.

The first condition is only valid for a single cache while the second is only guaranteed in case of a single cache with exponentially distributed iid inter-request times, TTL and delays. Extending this simple single cache to PH distributed inter-request times, TTL and delays, condition (b) is not guaranteed to be fulfilled. Moreover, for a caching tree model, the hit probability P_{ij} is no longer a bijective function of the TTL parameters as it is controlled by the TTL parameter vector representing multiple caches along a path from a leaf j to the root of the hierarchy. Trivially, for a leaf cache connected to a parent, storing an object permanently in either cache achieves an object hit probability of one.

C. Proof of Theorem 1

First, we prove (15). Let a represent any of the TTL parameters in $\boldsymbol{\mu}_k$ of object k. The partial derivative of f with respect to a is represented as

$$f^{a'} = -\frac{\partial}{\partial a}\left(\sum_i^N \sum_j^{n_l} \lambda_{Iij}\psi(P_{ij})\right) \ . \tag{25}$$

All the terms of the first summation do not depend on μ_k except for the term at $i = k$. Therefore,

$$f^{a'} = -\sum_j^{n_l} \lambda_{Ikj} \frac{\partial \psi(P_{kj})}{\partial a} = -\sum_j^{n_l} \lambda_{Ikj} \frac{\partial \psi}{\partial P_{kj}} \frac{\partial P_{kj}}{\partial a} \ . \tag{26}$$

$\psi' := \frac{\partial \psi}{\partial P_{kj}}$ is calculated depending on the chosen utility notion, e.g., for proportional fairness utility, $\psi' = [\ln(10)\log(P_{kj})]^{-1}$. Using (11), we obtain $\frac{\partial P_{kj}}{\partial a}$ in terms of $\boldsymbol{\pi}_k^{a'} = \frac{\partial \boldsymbol{\pi}_k}{\partial a}$ as $\frac{\partial P_{kj}}{\partial a} = -\frac{1}{\lambda_{Ikj}} \boldsymbol{\pi}_k^{a'} \mathbf{D}_{kj,1}\mathbf{1}$. Note that $\mathbf{D}_{kj,1}$ only depends on the request process. Using $\frac{\partial P_{kj}}{\partial a}$ in (26), we obtain (15). Next, we prove (16). We derive

$$\begin{aligned} f^{a'b'} &= \frac{\partial f^{a'}}{\partial b} = \sum_j \frac{\partial(\psi' \boldsymbol{\pi}_k^{a'})}{\partial b} \mathbf{D}_{kj,1}\mathbf{1} \\ &= \sum_j \left[\psi' \boldsymbol{\pi}_k^{a'b'} + \psi'' \frac{\partial P_{kj}}{\partial b} \boldsymbol{\pi}_k^{a'}\right] \mathbf{D}_{kj,1}\mathbf{1} \ . \end{aligned}$$

Inserting the expression of $\frac{\partial P_{kj}}{\partial b}$ from above we obtain

$$f^{a'b'} = \sum_j \left[\lambda_{Ikj}\psi' \boldsymbol{\pi}_k^{a'b'} - \psi'' \boldsymbol{\pi}_k^{b'} \mathbf{D}_{kj,1}\mathbf{1} \boldsymbol{\pi}_k^{a'}\right] \frac{\mathbf{D}_{kj,1}\mathbf{1}}{\lambda_{Ikj}} \ .$$

ACKNOWLEDGEMENT

This work has been funded by the German Research Foundation (DFG) as part of project B4 within the Collaborative Research Center (CRC) 1053 - MAKI.

REFERENCES

[1] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, p. 52–66, 2015.

[2] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. of IEEE INFOCOM*, 2010.

[3] M. Dehghan, L. Massoulié, D. Towsley, D. S. Menasché, and Y. C. Tay, "A utility optimization approach to network cache design," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1013–1027, 2019.

[4] C. Aggarwal, J. Wolf, and P. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 94–107, 1999.

[5] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, 2002.

[6] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Deepcache: A deep learning based framework for content caching," in *ACM Workshop on Network Meets AI & ML - NetAI*, 2018, p. 48–53.

[7] J. Jung, A. Berger, and H. Balakrishnan, "Modeling TTL-based internet caches," in *Proc. of IEEE INFOCOM*, 2003, pp. 417–426.

[8] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 758–770, 2013.

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

[10] K. Elsayed and A. Rizk, "Time-to-live caching with network delays: Exact analysis and computable approximations," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2022.

[11] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Performance evaluation of hierarchical TTL-based cache networks," *Computer Networks*, vol. 65, pp. 212–231, 2014.

[12] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley, "On the analysis of caches with pending interest tables," in *Proc. of ACM Conference on Information-Centric Networking*, 2015, p. 69–78.

[13] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," in *Proc. of ACM SIGCOMM*, 2020, pp. 495–513.

[14] K. Elsayed and A. Rizk, "Response times in time-to-live caching hierarchies under random network delays," *Würzburg Workshop on Next-Generation Communication Networks (WueWoWas'22)*, 2022.

[15] K. S. Elsayed, F. Geyer, and A. Rizk, "Utility-driven optimization of TTL cache hierarchies under network delays," *arXiv preprint arXiv:2405.04402 [cs.NI]*, 2024.

[16] F. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.

[17] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control and Stochastic Networks Perspective*. Cambridge University Press, 2014.

[18] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Performance evaluation of hierarchical TTL-based cache networks," *Computer Networks*, vol. 65, pp. 212 – 231, 2014.

[19] G. Neglia, D. Carra, and P. Michiardi, "Cache policies for linear utility maximization," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 302–313, 2018.

[20] A. Ferragut, I. Rodriguez, and F. Paganini, "Optimizing TTL caches under heavy-tailed demands," *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, p. 101–112, 2016.

[21] N. K. Panigrahy, J. Li, and D. Towsley, "Hit rate vs. hit probability based cache utility maximization," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, p. 21–23, 2017.

[22] N. K. Panigrahy, J. Li, D. Towsley, and C. Hollot, "Network cache design under stationary requests: Exact analysis and poisson approximation," *Computer Networks*, vol. 180, p. 107379, 2020.

[23] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *Proc. of IEEE INFOCOM 2001*, vol. 3, 2001, pp. 1416–1424.

[24] T. X. Tran and D. Pompili, "Octopus: A cooperative hierarchical caching strategy for cloud radio access networks," in *Proc. of IEEE MASS*, 2016, pp. 154–162.

[25] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of internet content delivery systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, p. 315–327, 2003.

[26] N. K. Panigrahy, J. Li, F. Zafari, D. Towsley, and P. Yu, "A TTL-based approach for content placement in edge networks," in *EAI International Conference on Performance Evaluation Methodologies and Tools*. Springer, 2021, pp. 1–21.

[27] C. Cho, S. Shin, H. Jeon, and S. Yoon, "TTL-Based Cache Utility Maximization Using Deep Reinforcement Learning," in *Proc. of IEEE GLOBECOM*, 2021, pp. 1–6.

[28] ——, "Elastic network cache control using deep reinforcement learning," in *Proc. of International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 1006–1008.

[29] S. Asmussen, *Applied Probability and Queues*, ser. Stochastic Modelling and Applied Probability. Springer, 2008.

[30] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks: The case of caching policies driven by stopping times," in *Proc. of ACM SIGMETRICS*, 2014, pp. 595–596.

[31] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990.

[32] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, 2000.

[33] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical programming*, vol. 89, pp. 149–185, 2000.

[34] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[35] R. Fletcher, S. Leyffer, and P. L. Toint, "On the Global Convergence of a Filter-SQP Algorithm," *SIAM Journal on Optimization*, vol. 13, no. 1, pp. 44–59, 2002.

[36] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, p. 1–31, 2005.

[37] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. of IEEE IJCNN*, 2005.

[38] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.

[39] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021, arxiv:2104.13478.

[40] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.

[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[42] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *Proc. of ITC*, 2012, pp. 1–8.

[43] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[44] O. Eytan, D. Harnik, E. Ofer, R. Friedman, and R. Kat, "IBM object store traces (SNIA IOTTA trace set 36305)," in *SNIA IOTTA Trace Repository*. Storage Networking Industry Association, 2019.