AoI based TTL caching for Dynamic content

***Abstract*—**

I. Introduction

II. Model and Problem statement

A. Exact Model

We consider a single cache that receives objects' requests defined by the rates λ_i for an object i. N is the total number of objects available and B is the cache capacity. The cache assigns a TTL T_i for each object i upon admittance and refreshes the TTL upon a cache hit. The objects age in the cache increases linearly with time after admittance. The TTL caching policy has the advantage of freely tuning the allocation of the objects in the cache independently and differently from each other. Our goal is to jointly optimize caching and age decisions to minimize the costs incurred by the system. We propose two different approaches depending on the abundance of the bandwidth between the cache and the server. In case of limited bandwidth, we optimize the TTL choice of every object to optimally balance the trade-off between the caching goal and the growing age depending on the cost function. In case of excess bandwidth, in addition to tuning the TTL we introduce an updating process for each cached object between the cache and the server. The updating process make use of the available bandwidth to reduce the age of the cached object.

B. Optimization problem

First, we formulate the general cost function as a linear combination of the cost of a cache miss and the cost incurred by the age. We write the objective function as

$$C = \sum_i \lambda_i \left[C_m(h_i) + C_\delta(\bar{\delta}_i) \right], \tag{1}$$

$C_m(.)$ and $C_\delta(.)$ are the cost functions of the caching misses and the age of retrieved objects, respectively. The caching cost is only dependent on the hit probability controlled by the TTL while the age cost depends on the expected age controlled by the TTL and the updating process if exists.

$$\min_{(h_i, r_i)\forall i} \sum_i \lambda_i [C_m(h_i) + C_\delta(\bar{\delta}_i(h_i, r_i))] \tag{2}$$

$$\text{subject to} \sum_i \mathrm{E}[O_i] \le B, \tag{3}$$

$$\sum_i \left(\lambda_i (1 - h_i) + \frac{\mathrm{E}[O_i]}{r_i} \right) \le R, \tag{4}$$

1) Case I: Limited Bandwidth: We find the optimal TTLs that optimize the trade-off between the caching hit probability and the freshness according to the objective function. Since the inter-request times and the inter-miss times are renewal, the retrieved object age between two misses is renewal. As a result, it is sufficient to derive the expected age within one cycle i.e., time between two misses. In one cycle, the age grows linearly during the object occupancy in the cache, where the age of the retrieved object upon a hit is equivalent to the request arrival time t_a assuming the cycle starts at time $t = 0$ and the age is 0 upon a miss. This establishes the trade-off between the increasing age for the cached objects and the zero age for objects fetched from the server. We calculate the expected age of the retrieved object upon a request as

$$\begin{aligned} \bar{\delta}_i := \mathrm{E}[\delta_i] &= h_i \mathrm{E}[t_a | \mathbb{h}] \\ &= \frac{1}{2} h_i \mathrm{E}[O_i] = \frac{1}{2} h_i^2 \end{aligned} \tag{5}$$

It is known that the expected value of Poisson arrival times within a time interval τ is $\tau/2$ []. Therefore, the expected age during the object occupancy in the cache is equivalent to $\mathrm{E}[O_i]/2$ and the expected occupancy for Poisson request arrivals is $\mathrm{E}[O_i] = h_i$ [].

The expected age is explicitly expressed in terms of the hit probability and since the hit probability is a bijective function in terms of the TTL, finding the optimal TTLs is equivalent to finding the optimal hit probabilities that dictate a unique TTL. The constant object TTL is related to its hit probability for Poisson request arrivals as

$$T_i = \begin{cases} \frac{-1}{\lambda_i} \ln(1 - h_i), & \text{for} \quad h_i < 1 \\ \infty, \quad h_i = 1. \end{cases} \tag{6}$$

We formulate the optimization problem using the objective function in (**??**) and the caching capacity constraint as

$$\min_{h_i \forall i} \sum_i \lambda_i [(1 - h_i) c_f + \frac{1}{2} h_i^2 c_\delta] \tag{7}$$

$$\text{subject to} \sum_i h_i \le B, \tag{8}$$

III. Approach

We derive the solution to the optimization problems in Sect. **??** by solving the Karush–Kuhn–Tucker (KKT) conditions.

A. Case I: Limited Bandwidth

We express the KKT conditions for the optimization problem (??) as

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial h_i} &= 0 \ \text{(Stationarity)} \\ \sum_i h_i &\leq B \ \text{(Primal feasibility)} \\ \eta &\geq 0 \ \text{(Dual feasibility)} \\ \eta \left[\sum_i h_i - B \right] &= 0 \ \text{(Complementary slackness)}\ , \end{aligned} \tag{9}$$

where $\mathcal{L}$ is the Lagrangian and η is the Lagrangian multiplier of the inequality constraint. We obtain the optimal hit probability in terms of the optimal Lagrangian multiplier using the stationary condition as

$$h_i^* = \frac{1}{c_\delta} \left(c_f - \frac{\eta^*}{\lambda_i} \right) \tag{10}$$

Using the optimal hit probability expression into the primal feasibility condition, we derive the range of the optimal Lagrangian multiplier as

$$\eta^* \geq \frac{N c_f - B c_\delta}{\sum_i (1/\lambda_i)} \tag{11}$$

The complementary slackness condition implies that

$$\eta^* = \begin{cases} \frac{N c_f - B c_\delta}{\sum_i (1/\lambda_i)} & \text{for } \ \sum_i h_i - B = 0, \\ 0 \ \text{ for } \ \sum_i h_i - B < 0 \end{cases} \tag{12}$$

Since η^* must not be negative, $\sum_i h_i - B$ can not be zero if $\frac{N c_f - B c_\delta}{\sum_i (1/\lambda_i)} < 0$, hence

$$\eta^* = \begin{cases} 0 \ \text{ if } \ B c_\delta > N c_f \\ \frac{N c_f - B c_\delta}{\sum_i (1/\lambda_i)} & \text{otherwise} \end{cases} \tag{13}$$

We deduce from the above condition that utilizing the whole cache is not optimal when the age cost of the full cache utilization exceeds the fetching cost when not caching at all. the balance between the fetching cost in case of caching no objects and the age cost when the whole cache is utilized. The optimal hit probability in such case is the same for all objects and is given from (??) as $\frac{c_f}{c_\delta}$, hence the expected utilized cache capacity is $N \frac{c_f}{c_\delta}$. Note that the optimal TTLs resulting in the same hit probabilities for all users calculated from (??) are not the same and are decreasing linearly in terms of λ_i.

Recall that we do not include the hit probabilities constraint $0 \leq h_i \leq 1$ in our derivation, thus the optimal derived hit probability in (??) might be infeasible. We simply account for that by iteratively calculating the optimal hit probabilities after projecting the most infeasible hit probability in each iteration to the nearest barrier and excluding it from the optimization in the next iteration. We summarize the iterative approach in Algorithm **??**

Algorithm 1: Joint Caching and Freshness Optimization

```
Input: $N$, $B$, and related parameters
Output: Optimized caching and freshness variables
1 Initialization: $m \leftarrow N$, $\mathcal{S} \leftarrow \{1, 2, \ldots, N\}, b = B$ ;
2 repeat
3     Find the optimal Lagrangian multiplier:
          $\eta^* \leftarrow \max\left(0, \frac{N c_f - B c_\delta}{\sum_i (1/\lambda_i)}\right)$
4     Calculate $h_i$ from (??) $\forall i \in \mathcal{S}$ ;
5     Find the most infeasible $h_i$:
          $k \leftarrow \arg\max(h_i - 1, h_i)$
6     Project $h_k$ to the feasible range:
          $h_k \leftarrow \max(\min(h_k, 1), 0)$
7     Exclude $k$ from the decision variables:
          $\mathcal{S} \leftarrow \mathcal{S} \setminus \{k\}$
8     Update the optimization function:
          $m \leftarrow N - 1, \quad b \leftarrow b - h_k$
9 until $0 \leq h_i \leq 1$ are feasible;
```

IV. Numerical Evaluations

Different experiments:

- Eq 5 verify simulation - for ourselves
- Numerical vs theoretical comparison, for different values of λ
- f_c and f_δ on each axis and plot hit probs or TTL
- TTL compared to LRU and static TTL
- ..
- Different μ and λ
- Comparison to others such as: LRU, random eviction, TTL without refresh rate $\mu = 0$, static TTL if the constraints are fulfilled.
- comparison to the TTL and u coming from an optimizer tool (in python)

References