Timer-based pre-fetching for increasing hazard rates

Andres Ferragut
Universidad ORT Uruguay
Montevideo, Uruguay
ferragut@ort.edu.uy

Matias Carrasco
Universidad ORT Uruguay
Montevideo, Uruguay
carrasco@ort.edu.uy

Fernando Paganini *
Universidad ORT Uruguay
Montevideo, Uruguay
paganini@ort.edu.uy

ABSTRACT

Caching plays a crucial role in today's networks: keeping popular content close to users reduces latency. Timer-based caching policies (TTL) have long been used to deal with *bursts* of requests, and their properties are well understood. However, in some scenarios the traffic is more *regular*. In this work, we define a dual of the TTL policy, dubbed Timer-based Pre-fetching, particularly well suited for these scenarios. We also show how the optimal timers can be computed for a large class of request processes.

1. INTRODUCTION

The notion of local storing (caching) of data has received increased attention in recent years with the emergence of cloud and edge computing architectures. In particular, [5] considered timer-based (TTL) policies fed by general arrival processes, and provided a tractable way to analyze these caching systems. Building upon this work, in [3, 4] the authors derived the optimal TTL caching policy under very general hypotheses for the request processes.

A key result in [3] is that the optimal policy depends on the *hazard rate* function of the inter-request times (see also [6] for a related result for replacement-based policies). Under a decreasing hazard rate (DHR) assumption, a convex optimization problem can be formulated to compute the optimal timers. Furthermore, suitable fluid limits for large scale systems are derived, yielding explicit expressions for the hit probability. However, when hazard rates are *increasing* (IHR), the optimal timer policy is just a static policy that stores the most popular contents at all times [3].

In this work, we propose a new policy better suited to the IHR case: timer-based *pre-fetching*, and show that we can greatly improve the hit probability by *speculatively* retrieving the content, in anticipation of future arrivals.

2. CACHING AND STATIONARY ARRIVALS

Consider a *local memory* or *cache* system, where requests from a *catalog* of N (equally sized) items are received. The cache has limited memory, and thus aims to locally keep available a subset of size $C < N$, which can then be served with lower latency. The natural objective is to maximize the *hit probability* by choosing the appropriate items.

*This work was partially supported by AFOSR US under grant #FA9550-23-1-0350.

MAMA Workshop 2024. June 14, 2024, Venice, Italy

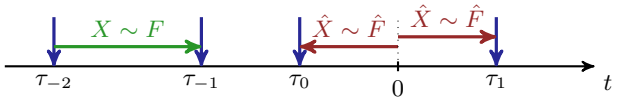

Figure 1: Inter-arrival and age distributions.

We model requests for item i as a stationary point process in $\mathbb{R}$ [1], with a given intensity λ_i (average requests per time unit). Their sum $\Lambda := \sum_{i=1}^{N} \lambda_i$ is the total intensity of requests, and $p_i := \lambda_i/\Lambda$ is the probability of a given request being for item i, i.e. its relative popularity.

For a stationary point process in the real line there are two main distributions: the inter-arrival distribution $F_i(t)$, i.e. if X_i is the time between arrivals, then $F_i(t) = P(X_i \leqslant t)$ and $E[X_i] = 1/\lambda_i$. However, this magnitude is synchronized with the process: when the process is sampled at an arbitrary point in time (e.g. 0 due to stationarity), the random variable $\hat{X}_i$ measuring the time since the last request follows the *age distribution* [1]:

$$\hat{F}_i(t) := P\left(\hat{X}_i \leqslant t\right) = \lambda_i \int_0^t 1 - F_i(s)\, ds. \tag{1}$$

Moreover, the *time to next request* also follows the same distribution, and this is why $\hat{F}_i$ is also known as the *residual lifetime distribution* associated to F_i. An example of this sampling effect is shown in Fig. 1.

Finally, another relevant magnitude in our analysis is the *hazard rate* function (also known as failure rate). If F_i has density f_i, its hazard rate is defined as:

$$\eta_i(t) = \frac{f_i(t)}{1 - F_i(t)}, \tag{2}$$

and serves as a local measure of the likelihood that the current interval is exactly of length t, given that the elapsed time of the interval is at least t.

2.1 Timer based caching policies

Timer based (TTL) caching policies work as follows: upon arrival of a request for item i, the item is stored in the cache (if not already present) and a timer of length T_i is started (or reset). When the timer expires, the content is removed. See Figure 2 for an example. This policy subsumes static policies (just choose $T_i = 0$ or ∞). It also provides a good approximation for the classical least-recently-used (LRU) policy, due to the "Che approximation" [2], which corresponds to choosing a homogeneous timer for all items.

In [3, 4], the authors formulate an optimization problem

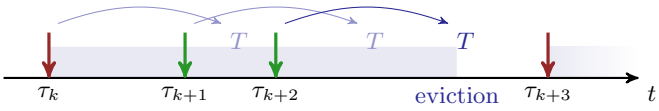

Figure 2: TTL caching policy for a single item.

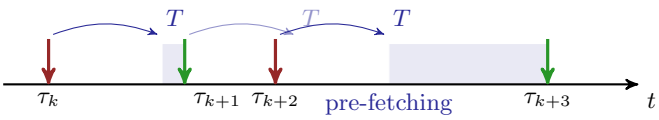

Figure 3: Timer pre-fetching policy for a single item.

to characterize the optimal timers as a function of the inter-arrival distributions F_i, and conclude that:

- For *decreasing hazard rates*, the problem can be shown to have a unique non-trivial solution, which amounts to choose timers that *equalize the hazard rates* of the underlying request processes.
- For *constant hazard rates* (i.e. Poisson arrivals) or *increasing hazard rates*, the best caching policy is the *static* policy of keeping the C most popular objects at the cache at all times.

The above results are expected: decreasing hazard rates correspond to *bursty* traffic, where requests are clustered in time, and thus caching can provide benefits. However, if traffic is purely random (Poisson) or more *regular*, then timer based caching cannot improve over the static policy.

This leads to the following question: can we improve performance in the case of increasing hazard rates? A positive answer is given hereafter.

3. TIMER BASED PRE-FETCHING

We now introduce our policy: the key insight is that, under the increasing hazard rate assumption, the likelihood of a subsequent request for item i *decreases* immediately upon the item being requested. Therefore, removing this item from memory and only retrieving it at a later time may improve performance. We now make this precise.

Consider the following policy: after a request for item i, we *remove* it from memory if it was already present, and start a timer T_i. At timer expiration, we *fetch the item again* and store it on memory. If a new request arrives before this, we will have a *miss* and the timer is reset. Otherwise, the item will have been *pre-fetched* for the next arrival, and we will have a *hit*. We call our policy *timer based pre-fetching* and its behavior is depicted in Fig. 3. An important observation is that the analysis of the hit probability *decouples* among the processes, as in the case of TTL caching.

The steady state hit-probability of item i for the aforementioned policy can be readily computed by observing that:

$$P(\text{item } i \text{ hit}) = P\left(X_i > T_i\right) = 1 - F_i(T_i).$$

Also, the steady state occupation probability can be computed by observing that item i is stored at time $t = 0$ iff its last request was more than T_i units of time before, i.e.:

$$P(\text{item } i \text{ in memory}) = P\left(\hat{X}_i > T_i\right) = 1 - \hat{F}_i(T_i),$$

where $\hat{F}_i$ is defined in (1). Note that the average memory occupation is therefore $E\left[\sum_i \mathbf{1}_{\{\hat{X}_i > T_i\}}\right] = \sum_i 1 - \hat{F}_i(T_i)$.

We can now formulate the optimal timer problem:

$$\max_{T_i \geqslant 0} \sum_{i=1}^N \lambda_i (1 - F_i(T_i)), \text{ s.t.: } \sum_{i=1}^N (1 - \hat{F}_i(T_i)) \leqslant C,$$

or equivalently, by getting rid of the constant terms:

$$\min_{T_i \geqslant 0} \sum_{i=1}^N \lambda_i F_i(T_i), \text{ s.t.: } \sum_{i=1}^N \hat{F}_i(T_i) \geqslant N - C.$$

The above Problem is closely related to Problem 1 in [3]. We are now minimizing the *miss* rate, subject to the number of *non-stored* items being larger than $N - C$ on average.

Using the fact that $\hat{F}_i$ is increasing, consider the (monotonically increasing) change of variables $u_i = \hat{F}_i(T_i)$; here $u_i \in [0, 1]$ is the probability of *not* being stored. The above problem can be rewritten as:

PROBLEM 1 (OPTIMAL TIMER BASED PRE-FETCHING).

$$\min_{u_i \in [0,1]} \sum_{i=1}^N \lambda_i F_i \left(\hat{F}_i^{-1}(u_i)\right), \text{ s.t.: } \sum_{i=1}^N u_i \geqslant N - C. \quad (3)$$

3.1 Increasing hazard rates

Let us compute the gradient of the objective function using eq. (1) and the inverse function theorem:

$$\frac{\partial}{\partial u_i} \lambda_i F_i \circ \hat{F}_i^{-1}(u_i) = \frac{\lambda_i f_i(\hat{F}_i^{-1}(u_i))}{\lambda_i \left(1 - F_i(\hat{F}_i^{-1}(u_i))\right)} = \eta_i(\hat{F}_i^{-1}(u_i)) \quad (4)$$

with η_i as in (2).

From (4), it follows that if the F_i have increasing hazard rates, the objective function in Problem 1 is a convex optimization program. In this case, we can introduce a multiplier θ for the constraint and write its Lagrangian as:

$$\begin{aligned}\mathcal{L}(u, \theta) &= \sum_{i=1}^N \lambda_i F_i \left(\hat{F}_i^{-1}(u_i)\right) + \theta \left(N - C - \sum_{i=1}^N u_i\right) \\ &= \sum_{i=1}^N \left[\lambda_i F_i \left(\hat{F}_i^{-1}(u_i)\right) - \theta u_i\right] + \theta(N - C). \quad (5)\end{aligned}$$

We are now ready to prove:

THEOREM 1. *Provided that the distributions F_i satisfy the IHR property, there exists a unique threshold $\theta^* > 0$ and timers T_i^* such that the optimal timer based pre-fetching policy defined by Problem 1 is given by:*

$$\eta_i(T_i^*) \geqslant \theta^*,$$

whenever $T_i^* < \infty$ (pre-fetching). The inequality is strict if and only if $T_i^* = 0$, i.e. the content is always stored.

PROOF. Starting from the Lagrangian (5), let us impose the KKT conditions for optimality. We first minimize over $u_i \in [0, 1]$, where the problem decouples over i:

$$\min_{u_i \in [0,1]} \lambda_i F_i(\hat{F}_i^{-1}(u_i)) - \theta^* u_i$$

By using (4), the gradient of the objective is $\eta_i(\hat{F}_i^{-1}(u_i)) - \theta^*$ and its increasing by hypothesis. Therefore, if $\eta_i(\hat{F}_i^{-1}(0)) = \eta_i(0) \geqslant \theta^*$, the above optimum is attained for $u_i^* = T_i^* = 0$ and the content is always stored. If instead, $\eta_i(\hat{F}_i^{-1}(1)) = \lim_{t \to \infty} \eta_i(t) < \theta^*$, the optimum is attained for $u_i^* = 1$ ($T_i^* =$

∞) and the item is never stored. In the remaining cases, the optimum is interior and satisfies:

$$\eta_i(\hat{F}_i^{-1}(u_i^*)) = \eta_i(T_i^*) = \theta^*.$$

Finally, the optimal threshold θ^* must satisfy the *complementary slackness* condition:

$$\theta^* \left(N - C - \sum_{i=1}^N u_i^* \right) = 0. \tag{6}$$

Note that in the optimum of Problem 1, we cannot have strict inequality in the constraint, because in that case the objective can be improved by lowering some u_i^*. Therefore, the second term in (6) must be 0 and θ^* must satisfy:

$$\sum_{i=1}^N u_i^* = \sum_{i=1}^N \hat{F}_i(T_i^*(\theta^*)) = N - C, \tag{7}$$

with $T_i^*(\theta^*)$ constructed as before. □

Theorem 1 shows that, under the IHR property, the optimal policy is a *threshold* policy: there exists a threshold θ^* such that an item is stored in the local memory if and only if its *current hazard rate* is greater than the threshold. The items with $\eta_i(0) \geqslant \theta^*$ are always stored, the items with $\eta_i(\infty) < \theta^*$ are never stored, and the remaining items are pre-fetched after a time T_i^* since the last request, where their hazard rate reaches the threshold. The underlying idea being that the hazard rate is the *marginal utility* of storing some object in the local memory with a fixed budget C.

3.2 Constant and decreasing hazard rates

For constant hazard rates, the arrivals become Poisson and Problem 1 becomes linear. It is easy to see that in this case the policy degenerates to the *static* policy where $T_i^* = 0$ for the C most-popular objects, which remain in memory forever. If instead the hazard rates are *decreasing*, we have the following result, which can be proved using similar arguments to [4, Theorem 1]:

THEOREM 2. *Provided that the distributions F_i satisfy the DHR property, the optimal timer based pre-fetching policy is to statically store the C most popular contents.*

The result of Theorem 2 is expected in light of the discussion of Section 2: when arrivals have the DHR property, traffic is bursty and pre-fetching does not help, i.e. *caching* makes more sense for DHR and *pre-fetching* for IHR traffic.

4. A PARAMETRIC EXAMPLE

To illustrate the gains obtained by the pre-fetching policy, consider the following example: the inter-arrival times X_i follow the Erlang distribution with $k-$stages, i.e.

$$f_i(t) = \frac{1}{(k-1)!}(k\lambda_i)^k t^{k-1} e^{-\lambda_i k t}, \quad t \geqslant 0,$$

where we have chosen the parametrization so $E[X_i] = 1/\lambda_i$. $k = 1$ corresponds to a Poisson process; as k increases, the inter-arrival times become more regular. We have:

LEMMA 1. *If X follows the aforementioned Erlang distribution, the hazard rate function satisfies:*

$$\eta_i(t) = k\lambda_i B(k\lambda_i t, k-1),$$

where $B(A, C)$ is the classical Erlang blocking probability from the $M/M/C/C$ queue. Note that in particular $\eta_i(t)$ is strictly increasing for $k \geqslant 2$.

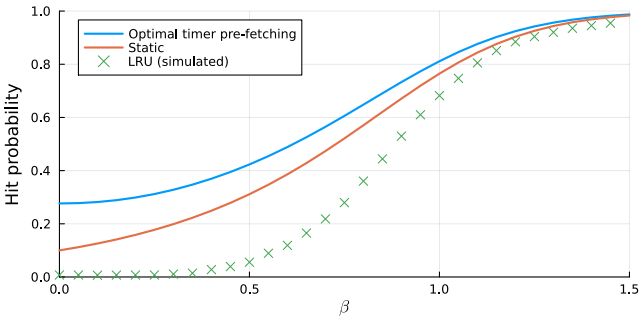

Figure 4: Hit rate comparison for timer-based pre-fetching, static storage (optimal caching) and LRU caching.

In this particular case, we can numerically solve Problem 1 via convex optimization, with the help of Lemma 1 to compute the gradient. As a particular example, consider a system catalog of size $N = 10000$ and capacity $C = 1000$. The total arrival rate is normalized to be $\Lambda = 1$ and relative popularities follow a Zipf(β), distribution, i.e. $\lambda_i \propto i^{-\beta}$, a common model for object popularities.

In Fig. 4, we plot the optimal hit probability as a function of the parameter β, for an Erlang distribution with $k = 5$ stages. As a comparison, we plot the hit rate for the static most popular policy, and the simulated hit rate of the classical LRU caching policy.

5. FINAL REMARKS

As we can see from the above example, for *regular* processes, classical caching ideas can deliver poor performance. Our main contribution is to highlight the role of the hazard rate function and provide a new timer-based pre-fetching policy to handle these scenarios.

Several lines of future work remain open: in particular how to estimate the timers based on previous data, and obtaining analogues of the classical caching policies that can be applied for pre-fetching.

6. REFERENCES

[1] P. Brémaud. *Point process calculus in time and space.* Springer, NY, 2020.

[2] H. Che, Y. Tung, and Z. Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, 2002.

[3] A. Ferragut, I. Rodriguez, and F. Paganini. Optimizing TTL caches under heavy tailed demands. In *Proc. of ACM/SIGMETRICS 2016*, pages 101–112, June 2016.

[4] A. Ferragut, I. Rodríguez, and F. Paganini. Optimal timer-based caching policies for general arrival processes. *Queueing Systems*, 88(3–4):207–241, 2018.

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

[6] N. K. Panigrahy, P. Nain, G. Neglia, and D. Towsley. A new upper bound on cache hit probability for non-anticipative caching policies. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 7(2–4), November 2022.