Optimized Dynamic Cache Instantiation and Accurate LRU Approximations under Time-varying Request Volume

Niklas Carlsson *Linköping University*, Sweden
Derek Eager *University of Saskatchewan*, Canada

Abstract—Content-delivery applications can achieve scalability and reduce wide-area network traffic using geographically distributed caches. However, each deployed cache has an associated cost, and under time-varying request rates (e.g., a daily cycle) there may be long periods when the request rate from the local region is not high enough to justify this cost. Cloud computing offers a solution to problems of this kind, by supporting dynamic allocation and release of resources. In this paper, we analyze the potential benefits from dynamically instantiating caches using resources from cloud service providers. We develop novel analytic caching models that accommodate time-varying request rates, transient behavior as a cache fills following instantiation, and selective cache insertion policies. Within the context of a simple cost model, we then develop bounds and compare policies with optimized parameter selections to obtain insights into key cost/performance tradeoffs. We find that dynamic cache instantiation can provide substantial cost reductions, that potential reductions strongly dependent on the object popularity skew, and that selective cache insertion can be even more beneficial in this context than with conventional edge caches. Finally, our contributions also include accurate and easy-to-compute approximations that are shown applicable to LRU caches under time-varying workloads.

Index Terms—Cloud computing, Edge cloud, Dynamic cache instantiation, Time-varying request volumes, Selective cache insertion, Request count window.

1 Introduction

The performance and scalability of content delivery systems benefit significantly from geographically distributed caches. It is therefore not surprising that caching solutions for these systems have generated much research (Section 7). However, despite the emergence of distributed, regional, and edge cloud computing offering a completely new service paradigm – on-demand caching – surprisingly few works have taken into account on-demand cache provisioning [2]–[6], and, to our knowledge, no prior work has considered, rigorously modelled, and analyzed the problem of when to instantiate and release caches in such environments.

We note that request rates of these systems typically differ between locations and the total aggregate request volumes for a given location typically vary over time according to a relatively predictable daily cycle [7]–[11]. Such cycles have been reported for ISPs [8], CDNs (e.g., Akamai on a per-country basis [7]), university networks (e.g., [9], [10]), enterprise networks (e.g., [10]), and popular services (e.g., YouTube [11]). These are also contexts where regional caches are often used.

Due to such daily patterns, in systems where the service provider pays on an on-demand basis, the cost of a local cache (in some locations) may therefore only be justified during the daily peak in the request rates. Ideally, we would like to incur the cost of a cache only when the request rate is sufficiently high to justify this cost.

In this paper, we take a first look at the potential benefits from dynamically instantiating and releasing caches (e.g., based on daily cycles). In particular, we develop novel analytic models of cache performance that accommodate the important challenges of taking into account (i) arbitrarily time-varying request rates and (ii) periods of transient behavior when a cache fills following instantiation, and apply these models within the context of a simple cost model to study cost/performance tradeoffs. Our models are motivated by scenarios in which a system has many independently operated cache locations and each cache is dynamically allocated within an edge cloud environment. Although there are many geographically distributed caches, for the purpose of our analysis we can consider just one such location. In contrast to prior works we consider a novel cloud context in which the caches are dynamically instantiated so as to minimize the delivery cost under time-varying request volumes.

First, to accommodate time-varying request rates and periods of transient behavior, we develop a modelling approach based on what we term here "request count window" (RCW) caches. Objects are evicted from an RCW cache if not requested over a window consisting of the most recent L requests, where L is a parameter of the system. As we show here empirically, similarly as with "Time-to-Live" (TTL) caches [12]–[16] in scenarios with fixed request rates, the performance of an RCW cache closely approximates the performance of an LRU cache when the size of the window (for an RCW cache, measured in number of requests) is set such that the average occupancy equals the LRU cache size.

Second, we carry out analytic analyses of RCW caches for both indiscriminate *Cache on* 1^{st} *request* and selective *Cache on* k^{th} *request* cache insertion policies. This includes the derivation of explicit, exact expressions for key cache performance metrics under the independent reference

A preliminary version of this paper appears as a 9-page paper at IFIP Networking 2020 [1].

model, including (i) the hit and insertion rates for permanently allocated caches, and (ii) the average rates over the transient period during which a newly instantiated cache is filling. We note that for this context, selective *Cache on k^{th} request* cache insertion policies are of particular interest, since dynamically instantiated caches may be relatively small, and therefore cache pollution may be a particularly important concern. We also derive approximate expressions of $O(1)$ computational cost for the cases of Zipf object popularities with parameter $\alpha = 1$ and $\alpha = 0.5$. These two cases are chosen as representative of high and low popularity skew, respectively. Our RCW analysis makes no assumptions regarding inter-request time distributions or request rate variations, ensuring that our RCW results (in contrast to prior TTL approximations [12]–[16]) can be used to approximate LRU cache performance under highly time-varying request volumes. In general, for time-varying workloads, the concept of RCW caches provides a more natural choice than TTL caches when approximating fixed-capacity LRU caches.

Third, in addition to the cache insertion policy, important design issues in a dynamic cache instantiation system include the choice of cache size and the duration of the cache instantiation interval. We develop optimization models for these parameters for both *Cache on 1^{st} request* and *Cache on k^{th} request*. We also develop bounds on the best potentially achievable cost/performance tradeoffs, assessing how much room for improvement there may be through use of more complex caching policies.

Fourth, we apply our analyses to obtain insights into the potential cost reductions possible with dynamic cache instantiation and explore key system tradeoffs. We find that:

> Dynamic cache instantiation has the potential to provide significant cost reductions, sometimes more than halving the costs of (optimized) baselines that uses a permanent cache, with the cache size selected so as to minimize the cost.
>
> The cost reductions are strongly dependent on the object popularity skew. When there is high skew, dynamic instantiation can work particularly well since a newly instantiated cache is quickly populated with frequently requested items that will capture a substantial fraction of the requests.
>
> Selective *Cache on k^{th} request* cache insertion policies can be even more beneficial in this context than with conventional edge caches, and, when there is high popularity skew, there is likely only modest room for improvement in cost/performance through use of more complex cache insertion and replacement policies.

Overall, these results show that dynamic cache instantiation using *Cache on k^{th} request* is a promising approach for content delivery applications.

Finally, it is important to note that there does not exist any analysis (from prior work) that captures the performance of LRU caches under time-varying workloads. Our development of easy-to-compute approximation expressions of the performance of LRU caches under time varying workloads is therefore an important contribution. The reason we use RCW for our analysis (rather than LRU) is in part because it enables both an exact analysis and because it provides a nice approximation for LRU caches, while still capturing the cache performance under time-varying workload volumes. In contrast, exact analysis of large LRU caches is intractable. Of course, in practice, we expect many systems to keep implementing LRU cache replacement policies or some variation thereof.

Roadmap: Section 2 describes our workload and system assumptions, the caching policies considered, and the metrics of interest. Section 3 presents our analysis of RCW caches for the baseline case without use of dynamic instantiation. Section 4 provides an analysis of the period of transient behavior as an RCW cache fills. Optimization models and performance results for dynamic instantiation are presented in Sections 5 and 6, respectively. Throughout the paper we derive and present results for both exact and $O(1)$-approximations. Section 7 describes related work, before Section 8 concludes the paper.

2 System Description and Metrics

Workload Assumptions: We focus on a single region within the service area of a content delivery application, or a cache location to which a subset of geographically distributed clients are directed [17]. For this cache location, we consider a time period of duration T (e.g., one day), over which the total (aggregated over all objects) content request rate $\lambda(t)$ varies. We assume that these variations are predictable (e.g., based on prior days), and so for any desired cache instantiation duration $D < T$, it would be possible to identify in advance the interval of duration D with the highest average request rate over all intervals of duration D within the time period.

Short-term temporal locality, non-stationary object popularities, and high rates of new content creation make dynamic cache instantiation potentially more promising, since they reduce the value of old cache contents. Here, we provide a conservative estimate of the benefits of dynamic cache instantiation, assuming a fixed set of objects with stationary object popularities, and with requests following the independent reference model. We denote the number of objects by N, and index the objects such that $p_i \quad p_{i+1}$ for $1 \quad i < N$, where p_i denotes the probability that a request is for object i.

Cache Policies: We model what we term here "request count window" (RCW) caches. Objects are evicted from an RCW cache if not requested over a window consisting of the most recent L requests, where L is a system parameter. As we empirically demonstrate, the performance of an RCW cache closely approximates the performance of an LRU cache when the value of L is set such that the average occupancy equals the size of the LRU cache.

Both indiscriminate, *Cache on 1^{st} request*, and selective *Cache on k^{th} request* cache insertion policies are considered. For *Cache on k^{th} request* with $k>1$, we assume that the system maintains some state information regarding uncached objects that have been requested at least once over a window consisting of the most recent W requests, where W is a policy parameter. Specifically, for each such "caching candidate", a count of how many requests are made for the object while it is a caching candidate is maintained. When

a request is received for an uncached object that was not already a caching candidate, the object becomes a caching candidate with count initialized to one. Should this count reach k, the object is cached. Should no request be made to the object for W requests, the object is removed as a caching candidate.

For the dynamic instantiation, we assume that the cloud provider returns an empty cache when (re)instantiated. This does not require us to make any assumption of the type of cache (e.g., in memory vs disk-based storage, type of VMs, etc.). However, we note that the cloud provider that is not able to rent out the resources to serve other workloads may decide to only shut down disks/memory to save energy and in some of these cases therefore potentially could return part of the cache in its original state. For such a case, our analysis provides a pessimistic performance bound.

Metrics and Cost Assumptions: The metrics of primary interest are the expected *fraction of requests served locally* from cache (over the entire time period), and the *cache cost*. With dynamic cache instantiation, the first of these two metrics is given by $\frac{H_{t_a:t_d}\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t}{\int_0^T\lambda(t)\mathrm{d}t}$, where t_a denotes the time at which the cache is allocated, t_d the time at which it is deallocated, and $\bar{H}_{t_a:t_d}$ the average hit rate over this interval. Note that the hit rate (probability) will vary over the interval, with the hit rate immediately after instantiation being zero (empty cache).

Implementations of dynamic cache instantiation could use a variety of technologies. One option would be to use dynamic allocation of a virtual machine, with main memory used for the cache. We assume here a simple cost model where the cost per unit time of a cache of capacity C objects is proportional to $C + b$, where the constant b captures the portion of the cost that is independent of cache size. The total cost over the period T is then proportional to $(t_d - t_a)(C + b)$. More complex cost models could be easily accommodated; the only issue being the computational cost of evaluating the cost function when solving our optimization models.

In addition to the above metrics, when analyzing RCW caches we evaluate the hit rate H, the cache insertion rate I (fraction of requests that result in object insertions into the cache) as well as the insertion fraction $I/(I + H)$, and the average number A of objects in the cache. The insertion fraction is an important measure of overhead; cache insertions consume node resources, but do not yield any benefit unless there are subsequent resulting cache hits. The average number of objects in the cache is used to match the cache capacity C of an LRU cache with similar performance.

3 RCW Cache Analysis

In this section, we present analysis and performance results for a permanently allocated RCW cache. Section 3.1 derives exact expression for such an "always on" RCW cache with arbitrary file object popularity. Section 3.2 then derives approximations of $O(1)$ computational cost for the case of Zipf object popularities with $\alpha = 1$ and $\alpha = 0.5$. Next, Section 3.3 leverages these results to derive $O(1)$ approximation expressions for the RCW performance. For readers wanting to skip the derivation details, we refer to equations (11)-(13) or equations (14) for exact expressions, Table 2 for the key $O(1)$ approximations, and to Section 3.4 for validation results and policy comparisons. Table 1 summarizes our notation.

TABLE 1
Summary of notation

Notation	Definition
T	Total duration of time period
t_a	Time at which cache is allocated
t_d	Time at which cache is deallocated
N	Number of objects
α	Parameter of Zipf popularity distribution
p_i	Probability that a request is to object i
L	Cache lifetime parameter (# requests)
W	*Cache on k^{th} request* window (# requests)
C	Cache capacity (# objects)
b	Cost per unit time independent of cache size
H	Cache hit rate
I	Cache insertion rate
A	Average number of objects in cache
Θ_i	Object i duration in cache (# requests)
Δ_i	Object i duration out of cache (# requests)
γ	Euler-Mascheroni constant (≈ 0.577)
Ω	Zipf normalization constant

3.1 Exact "Always on" RCW Analysis

Cache on 1^{st} Request: The probability that a request for object i finds it in the cache is given by $1 - (1 - p_i)^L$, since object i will be in the cache if and only if at least one of the most recent L requests was to object i. The average number A of objects in the cache, as seen by a random request, the insertion rate I, and the hit rate H, are therefore given by

$$A = N \sum_{i=1} (1 - p_i)^L;\ I = \sum_{i=1} p_i(1 - p_i)^L;\ H = 1 \sum_{i=1} p_i(1 - p_i)^L. \quad (1)$$

Cache on 2^{nd} Request: The expected value $E[\Theta_i]$ of the object i duration in the cache, measured in number of requests, is given by the average number of requests until there is a sequence of L requests in a row that do not include a request for object i. Since requests follow the independent reference model and the probability of a request for object i is p_i, this is the same as the average number of flips of a biased coin that are required to get L heads in a row, with the probability of a head equal to $1 - p_i$:

$$E[\ _i] = \sum_{r=1} \frac{1}{(1 - p_i)^r} = \frac{1 - (1 - p_i)^L}{p_i(1 - p_i)^L}. \quad (2)$$

With *Cache on 2^{nd} request*, the expected value $E[\Delta_i]$ of the object i duration out of the cache, measured in number of requests, satisfies the following equation:

$$E[\ _i] = 1=p_i + (1 - p_i)^W (W + E[\ _i]) + (1 - (1 - p_i)^W)\ 1=p_i \quad \frac{(1 - p_i)^W W}{1 - (1 - p_i)^W} \quad (3)$$

Here, the first term $(1/p_i)$ gives the expected number of requests until the first request for object i following its removal from the cache. The second term gives the expected number of additional requests until object i is added to the cache, conditional on the first request not being followed by another request within the window W, multiplied by the probability of this condition. The third term gives the expected number of additional requests until object i is added to the cache, conditional on the first request being followed

by another request within the window W, multiplied by the probability of this condition. Solving for $E[\Delta_i]$ yields:

$$E[\quad_i] = \frac{2 \quad (1 \quad p_i)^W}{p_i(1 \quad (1 \quad p_i)^W)}: \tag{4}$$

Noting that $A = \sum_i \frac{E[\quad_i]}{E[\quad_i]+E[\quad_i]}$, $H = \sum_i p_i \frac{E[\quad_i]}{E[\quad_i]+E[\quad_i]}$, and $I = \sum_i \frac{1}{E[\quad_i]+E[\quad_i]}$, we have:

$$A = \sum_{i=1}^{N} \frac{1 \quad (1 \quad p_i)^L \quad (1 \quad p_i)^W + (1 \quad p_i)^{L+W}}{1 + (1 \quad p_i)^L \quad (1 \quad p_i)^W}; \tag{5}$$

$$H = \sum_{i=1}^{N} p_i \frac{1 \quad (1 \quad p_i)^L \quad (1 \quad p_i)^W + (1 \quad p_i)^{L+W}}{1 + (1 \quad p_i)^L \quad (1 \quad p_i)^W}; \tag{6}$$

$$I = \sum_{i=1}^{N} p_i \frac{(1 \quad p_i)^L \quad (1 \quad p_i)^{L+W}}{1 + (1 \quad p_i)^L \quad (1 \quad p_i)^W}: \tag{7}$$

Cache on k^{th} Request: The analysis for general $k \quad 2$ differs from that for *Cache on 2^{nd} request* with respect to $E[\Delta_i]$, the expected value of the object i duration out of the cache, measured in number of requests. Denoting $E[\Delta_i]$ for *Cache on k^{th} request* by $E^k[\Delta_i]$, $E^k[\Delta_i]$ ($k \quad 2$) can be expressed as a function of $E^{k \quad 1}[\Delta_i]$ as follows:

$$E^k[\quad_i] = \frac{E^{k \quad 1}[\quad_i] + (1 \quad p_i)^W W + (1 \quad (1 \quad p_i)^W) \quad \frac{1}{p_i} \quad \frac{(1 \quad p_i)^W W}{1 \quad (1 \quad p_i)^W}}{1 \quad (1 \quad p_i)^W}; \tag{8}$$

with $E^1[\Delta_i]$ defined as $1/p_i$. The numerator of the right-hand side of this equation gives the expected number of requests from when an object is removed from cache or removed as a caching candidate, until it next exits from the state in which it is a caching candidate with a count of $k \quad 1$ (either owing to being cached because of a request occurring within the window W, or removed as a caching candidate if no such request occurs). The denominator is the probability of being cached when exiting from the state in which it is a caching candidate with a count of $k \quad 1$, and therefore the inverse of the denominator gives the expected number of times the object will enter this state until it is finally cached. Simplifying yields

$$E^k[\quad_i] = \frac{E^{k \quad 1}[\quad_i]}{1 \quad (1 \quad p_i)^W} + \frac{1}{p_i}; \tag{9}$$

implying

$$E^k[\quad_i] = \frac{1}{p_i} \quad \frac{1 \quad (1 \quad (1 \quad p_i)^W)^k}{(1 \quad p_i)^W (1 \quad (1 \quad p_i)^W)^{k \quad 1}} \quad : \tag{10}$$

Expressing A, H, and I in terms of $E^k[\Delta_i]$ and $E[\Theta_i]$, and then substituting in the above expression for $E^k[\Delta_i]$ and the expression for $E[\Theta_i]$ from (2), yields:

$$A = \sum_{i=1}^{N} \frac{1 \quad (1 \quad p_i)^L}{1 \quad (1 \quad p_i)^L + \frac{(1 \quad p_i)^L (1 \quad (1 \quad (1 \quad p_i)^W)^k)}{(1 \quad p_i)^W (1 \quad (1 \quad p_i)^W)^{k \quad 1}}}; \tag{11}$$

$$H = \sum_{i=1}^{N} p_i \frac{1 \quad (1 \quad p_i)^L}{1 \quad (1 \quad p_i)^L + \frac{(1 \quad p_i)^L (1 \quad (1 \quad (1 \quad p_i)^W)^k)}{(1 \quad p_i)^W (1 \quad (1 \quad p_i)^W)^{k \quad 1}}}; \tag{12}$$

$$I = \sum_{i=1}^{N} p_i \frac{(1 \quad p_i)^L}{1 \quad (1 \quad p_i)^L + \frac{(1 \quad p_i)^L (1 \quad (1 \quad (1 \quad p_i)^W)^k)}{(1 \quad p_i)^W (1 \quad (1 \quad p_i)^W)^{k \quad 1}}}: \tag{13}$$

Note that for W=L, equations (11), (12), and (13) reduce to:

$$A = \sum_{i=1}^{N} (1 \quad (1 \quad p_i)^L)^k; \quad H = \sum_{i=1}^{N} p_i (1 \quad (1 \quad p_i)^L)^k;$$

$$I = \sum_{i=1}^{N} p_i (1 \quad p_i)^L (1 \quad (1 \quad p_i)^L)^{k \quad 1}: \tag{14}$$

3.2 Summation approximations

We have derived approximate expressions of $O(1)$ computational cost for the cases of Zipf object popularities with α = 1 and α = 0.5. Table 2 summarizes the key approximations that we obtain. As an important step in deriving these novel approximations, this subsection derives foundational summation approximations, which we then apply in Section 3.3 to derive the RCW approximations shown in Table 2. Readers not interested in the derivations of these approximations can skip to the evaluation comparisons of RCW vs LRU and exact vs approximate RCW analysis provided in Section 3.4.

3.2.1 *Zipf with* $\alpha = 1$

Consider the case of a Zipf object popularity distribution with $\alpha = 1$, and denote the normalization constant $\sum_{i=1}^{N} 1/i$ by Ω. Note that for large N, $\Omega \quad \ln N + \gamma$ where γ denotes the Euler-Mascheroni constant (0.577).

For large N, L, and $g \quad ! \quad 1$,

$$\sum_{i=1}^{N} (1 \quad p_i)^L \quad \sum_{i=1}^{N} e^{\quad L=(i \quad)} \quad \int_{L=(g(\ln N + \quad))}^{N} e^{\quad L=(x(\ln N + \quad))} dx: \tag{15}$$

Using a Taylor series expansion for e^y gives:

$$\begin{aligned}
&\int_{L=(g(\ln N + \quad))}^{N} e^{\quad L=(x(\ln N + \quad))} dx \\
&= \int_{L=(g(\ln N + \quad))}^{N} \sum_{j=0}^{\infty} \frac{(\quad L=(x(\ln N + \quad)))^j}{j!} dx \\
&= N \quad \frac{L}{\ln N + } \left@ \ln N + \frac{L}{2N(\ln N + \quad)} \sum_{j=2}^{\infty} \frac{(\quad L=(N(\ln N + \quad)))^j}{(j+1)!j} \right^1_0 \\
&\quad \frac{L}{\ln N + } \left@_{1=g} \quad \ln(L=(\ln N + \quad)) + \ln(g) + \sum_{j=1}^{\infty} \frac{(\quad g)^j}{(j+1)!j} \right^1 :
\end{aligned} \tag{16}$$

Note that

$$\ln(g) + \sum_{j=1}^{\infty} \frac{(\quad g)^j}{j!j} = \mathrm{Ei}(\quad g) \quad ; \tag{17}$$

where Ei is the exponential integral function, and that

$$\sum_{j=1}^{\infty} \frac{(\quad g)^j}{(j+1)!j} \quad \sum_{j=1}^{\infty} \frac{(\quad g)^j}{j!j} = \frac{1}{g} \sum_{j=1}^{\infty} \frac{(\quad g)^{j+1}}{(j+1)!} = \frac{1}{g} \quad e^{\quad g} + g \quad 1 \; ; \tag{18}$$

which tends to 1 as $g \quad ! \quad 1$. Also, for $g \quad ! \quad 1$, Ei($\quad g$) $\quad ! \quad 0$. Therefore, for $g \quad ! \quad 1$,

$$1=g + \ln(g) + \sum_{j=1}^{\infty} \frac{(\quad g)^j}{(j+1)!j} \quad ! \quad 1 \quad : \tag{19}$$

Substituting this result into (16), and neglecting the terms in the summation on the third line of (16) under the assumption that L is substantially smaller than $N(\ln N + \gamma)$, yields

$$\sum_{i=1}^{N} (1 \quad p_i)^L \quad N \quad \frac{L}{\ln N + } \quad \ln N \quad \ln \quad \frac{L}{\ln N + } \quad +1 \quad + \frac{L}{2N(\ln N + \quad)} \quad : \tag{20}$$

Again assuming large N, L, and $g \quad ! \quad 1$,

$$\sum_{i=1}^{N} p_i (1 \quad p_i)^L \quad \sum_{i=1}^{N} \frac{e^{\quad L=(i \quad)}}{i} \quad \int_{L=(g(\ln N + \quad))}^{N} \frac{e^{\quad L=(x(\ln N + \quad))}}{x(\ln N + \quad)} dx: \tag{21}$$

TABLE 2
$O(1)$ approximations.

	Policy (or sum)	Zipf, $\alpha = 1$	Zipf, $\alpha = 0.5$
Sums	$\sum_{i=1}^{N}(1-p_i)^L$	$N - \frac{L}{\ln N+\gamma} - \ln N - \ln\frac{L}{\ln N+\gamma} + 1 + \frac{L}{2N(\ln N+\gamma)}$	$N - L + \frac{L^2}{4N}\ln((2N)/L) + \frac{L}{6N} + \frac{3}{2} - \gamma$
	$\sum_{i=1}^{N} p_i(1-p_i)^L$	$1 - \frac{\ln(L/(\ln N+\gamma))+2}{\ln N+\gamma} - \frac{L/(N(\ln N+\gamma))}{\ln N+\gamma}$	$1 - \frac{L}{2N}\ln((2N)/L) + \frac{L}{4N} + 1 - \gamma$
Always on (steady state)	Cache on 1^{st}	$A - \frac{L}{\ln N+\gamma} - \ln N - \ln\frac{L}{\ln N+\gamma} + 1 + \frac{L}{2N(\ln N+\gamma)}$ $I - 1 - \frac{\ln(L/(\ln N+\gamma))+2}{\ln N+\gamma} - \frac{L/(N(\ln N+\gamma))}{\ln N+\gamma}$ $H - \frac{\ln(L/(\ln N+\gamma))+2}{\ln N+\gamma} - \frac{L/(N(\ln N+\gamma))}{\ln N+\gamma}$	$A - L - 1 - \frac{L}{4N}\ln((2N)/L) + \frac{L}{6N} + \frac{3}{2} - \gamma$ $I - 1 - \frac{L}{2N}\ln((2N)/L) + \frac{L}{4N} + 1 - \gamma$ $H - \frac{L}{2N}\ln((2N)/L) + \frac{L}{4N} + 1 - \gamma$
	Cache on 2^{nd}, $W = L$	$A - \frac{(2\ln 2 - L/(N(\ln N+\gamma)))L}{\ln N+\gamma}$ $H - \frac{\ln(L/(\ln N+\gamma))+2}{\ln N+\gamma} - \frac{\ln 2}{\ln N+\gamma}$ $I - \frac{\ln 2 - L/(N(\ln N+\gamma))}{\ln N+\gamma}$	$A - \frac{L^2}{2N}\ln(N/(2L)) + \frac{L}{2N} + \frac{3}{2} - \gamma$ $H - \frac{L}{N} - \ln 2 - \frac{L}{4N}$ $I - \frac{L}{2N}\ln(N/(2L)) + \frac{3L}{4N} + 1 - \gamma$
	Cache on k^{th}, $k \geq 3$, $W = L$	$A - \frac{\sum_{j=2}^{k}(-1)^j\binom{k}{j}j\ln(j) - L}{\ln N+\gamma}$ $H - \frac{\ln(L/(\ln N+\gamma))+2 - \sum_{j=2}^{k}(-1)^j\binom{k}{j}\ln(j)}{\ln N+\gamma}$ $I - \frac{\sum_{j=2}^{k}(-1)^j\binom{k}{j}\ln(j) - 1}{\ln N+\gamma}$	$A - \begin{cases}(9\ln 3 - 12\ln 2 - L/N)L^2/(4N) & k=3;\\ \sum_{j=2}^{k}(-1)^{j+1}\binom{k}{j}j^2\ln(j) - L^2/(4N) & k\geq 4;\end{cases}$ $H - \frac{L}{2N} - \sum_{j=2}^{k}(-1)^j\binom{k}{j}j\ln(j)$ $I - \begin{cases}(L/(2N)) - 3\ln 3 + 4\ln 2 - \frac{L}{2N} & k=3;\\ (L/(2N)) - \sum_{j=1}^{k-1}(-1)^j\binom{k}{j}(j+1)\ln(j+1) & k\geq 4;\end{cases}$
Transient	Cache on 1^{st}	$H_{\text{transient}} - \frac{\ln(L/(\ln N+\gamma))+2 - 1 - L/(2N(\ln N+\gamma))}{\ln N+\gamma}$	$H_{\text{transient}} - \frac{L}{4N}\ln((2N)/L) + \frac{L}{6N} + \frac{3}{2} - \gamma$
	Cache on 2^{nd}, $W = L$	$H_{\text{transient}} - \frac{\ln(L/(\ln N+\gamma))+2 - 2\ln 2}{\ln N+\gamma}$	$H_{\text{transient}} - \frac{L}{N} - \ln 2 - \frac{L}{8N} - \frac{1}{4}$
	Cache on k^{th}, $k \geq 3$, $W = L$	Insert above into $H_{\text{transient}} = H + I - \frac{A}{L}$	Insert above into $H_{\text{transient}} = H + I - \frac{A}{L}$

Using a Taylor series expansion for e^y gives:

$$\int_{L/(g(\ln N+\gamma))}^{N} \frac{e^{-L/(x(\ln N+\gamma))}}{x(\ln N+\gamma)}dx = \int_{L/(g(\ln N+\gamma))}^{N}\sum_{j=0}^{\infty}\frac{(-L)^j(x(\ln N+\gamma))^{-(j+1)}}{j!}dx$$
$$= \frac{1}{\ln N+\gamma}\left(\ln(N) + \frac{L}{N(\ln N+\gamma)}\sum_{j=2}^{\infty}\frac{(-L/(N(\ln N+\gamma)))^j}{j!j}\right) - \frac{1}{\ln N+\gamma}\left(\ln(L/(\ln N+\gamma)) - \ln(g) - \sum_{j=1}^{\infty}\frac{(-g)^j}{j!j}\right). \quad (22)$$

Applying (17) and considering $g \to 1$, and neglecting the terms in the summation on the second line of (22) yields

$$\sum_{i=1}^{N} p_i(1-p_i)^L \sim 1 - \frac{\ln(L/(\ln N+\gamma)) + 2 - \gamma - L/(N(\ln N+\gamma))}{\ln N+\gamma}. \quad (23)$$

3.2.2 Zipf with $\alpha = 0.5$

Consider the case of a Zipf object popularity distribution with $\alpha = 0.5$, and denote the normalization constant $\sum_{i=1}^{N} 1/\sqrt{i}$ by Ω. Note that for large N, $\Omega \sim 2\sqrt{N}$.

For large N, L, and $g \to 1$,

$$\sum_{i=1}^{N}(1-p_i)^L \sim \sum_{i=1}^{N} e^{-L/((\sqrt{i})\Omega)} \sim \int_{(L/(2g\sqrt{N}))^2}^{N} e^{-L/(2\sqrt{x}\sqrt{N})}dx. \quad (24)$$

Using a Taylor series expansion for e^y gives:

$$\int_{(L/(2g\sqrt{N}))^2}^{N} e^{-L/(2\sqrt{x}\sqrt{N})}dx = \int_{(L/(2g\sqrt{N}))^2}^{N}\sum_{j=0}^{\infty}\frac{(-L/(2\sqrt{x}\sqrt{N}))^j}{j!}dx$$
$$= N - L + (L^2/(8N))\ln N + \frac{L^3}{24N^2} - \frac{L^2}{4N}\sum_{j=4}^{\infty}\frac{(-L/(2N))^{j-2}}{j!(j-2-1)}$$
$$\frac{L^2}{4N}\left(1 - g^2 - 2g + \ln(L/(2\sqrt{N})) - \ln(g) - \frac{2}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{j!(j-2)}\right). \quad (25)$$

Applying (17) as well as the Taylor series expansion for e^y, note that

$$\frac{2}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{j!(j-2)}$$
$$= \frac{1}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{(j-1)!(j-2)} + \frac{2}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{(j-1)!(j-2)}\left(\frac{1}{j} - \frac{1}{2}\right)$$
$$= \frac{1}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{(j-1)!(j-2)} - \frac{1}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{j!}$$
$$= \frac{1}{g}\left(e^{-g} + g - 1 + \mathrm{Ei}(-g) - \ln g - \gamma\right) - \frac{1}{g^2}\left(e^{-g} - \frac{g^2}{2} + g - 1\right). \quad (26)$$

Therefore, for $g \to 1$,

$$1 - 1/g^2 - 2/g - \ln(g) - \frac{2}{g^2}\sum_{j=3}^{\infty}\frac{(-g)^j}{j!(j-2)} \sim -\frac{3}{2} + \gamma. \quad (27)$$

Substituting this result into (25), and neglecting the terms in the summation on the second line of (25) under the assumption that L is substantially smaller than $2N$, yields

$$\sum_{i=1}^{N}(1-p_i)^L \sim N - L + \frac{L^2}{4N}\left(\ln((2N)/L) + \frac{L}{6N} + \frac{3}{2} - \gamma\right). \quad (28)$$

Again assuming large N, L, and $g \to 1$,

$$\sum_{i=1}^{N} p_i(1-p_i)^L \sim \sum_{i=1}^{N} e^{-L/((\sqrt{i})\Omega)} \sim \int_{(L/(2g\sqrt{N}))^2}^{N} e^{-L/(2\sqrt{x}\sqrt{N})}dx. \quad (29)$$

Using a Taylor series expansion for e^y gives:

$$\int_{(L/(2g\sqrt{N}))^2}^{N} e^{-L/(2\sqrt{x}\sqrt{N})}dx = \int_{(L/(2g\sqrt{N}))^2}^{N}\sum_{j=0}^{\infty}\frac{(-L)^j}{j!(2\sqrt{x}\sqrt{N})^{j+1}}dx$$
$$= 1 - (L/(4N))\ln N - L^2/(8N^2) - \sum_{j=3}^{\infty}\frac{(-L/(2N))^j}{j!(j-1)}$$
$$\frac{L}{2N}\left(1 - g - \ln(L/(2\sqrt{N})) + \ln(g) + \sum_{j=2}^{\infty}\frac{(-g)^{j-1}}{j!(j-1)}\right). \quad (30)$$

Finally, making the assumption that L is substantially smaller than $2N$, we neglect the terms in the summation on the second line of (30). Applying (19) then yields

$$\sum_{i=1}^{N} p_i(1-p_i)^L \sim 1 - \frac{L}{2N}\left(\ln((2N)/L) + \frac{L}{4N} + 1 - \gamma\right). \quad (31)$$

3.3 RCW Cache Approximations

3.3.1 Cache on 1st Request, Zipf with $\alpha = 1$

Consider now the case of a Zipf popularity distribution with $\alpha = 1$. Applying (20) to the equation for A in (1), and (23) to the equations for I and H, yields:

$$A \quad \frac{L}{\ln N + } \quad \ln N \quad \ln \quad \frac{L}{\ln N + } \quad + 1 \quad + \frac{L}{2N(\ln N +)} \quad ; \tag{32}$$

$$I \quad 1 \quad \frac{\ln(L=(\ln N +)) + 2 \quad L=(N(\ln N +))}{\ln N + }; \tag{33}$$

$$H \quad \frac{\ln(L=(\ln N +)) + 2 \quad L=(N(\ln N +))}{\ln N + }: \tag{34}$$

As we show empirically, the performance of an RCW cache closely approximates the performance of an LRU cache when L is set such that the average occupancy equals the size of the LRU cache. Suppose that the LRU cache capacity $C = N^{\beta}$ for $0 < \beta < 1$. Equating the LRU cache capacity to the approximation for A given in (32) yields

$$N \quad = \frac{L}{\ln N + } \quad \ln N \quad \ln \quad \frac{L}{\ln N + } \quad + 1 \quad + \frac{L}{2N(\ln N +)} \quad : \tag{35}$$

An accurate approximation for the value of L satisfying this equation in the region of interest can be obtained by substituting for L in this equation with $N^{\beta}/((1 \quad (\ln N/(\ln N + \gamma))\beta)(1 + a))$, using the approximation $\ln(1 + a) \quad a$ when $|a| < 1$, neglecting the last term on the right-hand side, and then solving for a to obtain:

$$a = \frac{\ln((1 \quad)\ln N +) + 1 \quad 2}{(1 \quad)\ln N \quad (1 \quad)}: \tag{36}$$

This yields:

$$L \quad \frac{N \quad (\ln N +)((1 \quad)\ln N + \quad 1)}{((1 \quad)\ln N +)((1 \quad)\ln N \quad + \ln((1 \quad)\ln N +))}: \tag{37}$$

Note that for large N, L is substantially smaller than $N(\ln N + \gamma)$, as was assumed for the approximations (20) and (23). Substituting into expressions (34) and (33) yield cache hit rate and corresponding insertion rate approximations. For the hit rate, the resulting approximation is β minus a term that (slowly) goes to zero as $N \to 1$:

$$H \quad \frac{\ln \frac{((1 \quad)\ln N +)((1 \quad)\ln N \quad + \ln((1 \quad)\ln N +))}{(1 \quad)\ln N + \quad 1}}{\ln N + } + \frac{\frac{(1 \quad)\ln N + \quad 1}{N^{1} \quad ((1 \quad)\ln N +)((1 \quad)\ln N \quad + \ln((1 \quad)\ln N +))} \quad (2 \quad)}{\ln N + }: \tag{38}$$

We observe that further approximations can yield a simpler approximation for H, accurate over a broad range of cache sizes, of $\beta \quad c(1 \quad \beta)/(2 \quad \beta)$ where c is a small constant dependent on N (e.g. $c = 1/3$ gives good results for N in the 10,000 to 100,000 range). In contrast, note that the hit rate when the cache is kept filled with the $\lfloor C \rfloor$ most popular objects (the optimal policy under the IRM assumption, without knowledge of future requests) is given in this case by $\sum_{i=1}^{\lfloor C \rfloor} p_i \quad (\ln C + \gamma)/(\ln N + \gamma)$. When $C = N^{\beta}$, this equals $\beta + \gamma(1 \quad \beta)/(\ln N + \gamma)$.

3.3.2 Cache on 1st Request, Zipf with $\alpha = 0.5$

Consider now the case of a Zipf popularity distribution with $\alpha = 0.5$. Applying (28) to the equation for A in (1), and (31) to the equations for I and H, yields:

$$A \quad L \quad 1 \quad \frac{L}{4N} \quad \ln((2N)=L) + \frac{L}{6N} + \frac{3}{2} \quad ; \tag{39}$$

$$I \quad 1 \quad \frac{L}{2N} \quad \ln((2N)=L) + \frac{L}{4N} + 1 \quad ; \tag{40}$$

$$H \quad \frac{L}{2N} \quad \ln((2N)=L) + \frac{L}{4N} + 1 \quad : \tag{41}$$

Suppose now that the corresponding LRU cache capacity $C = fN$ for some $f > 0$. Equating the cache capacity C to the approximation for the average number of objects in the cache as given by (39) gives

$$fN = L \quad 1 \quad \frac{L}{4N} \quad \ln((2N)=L) + \frac{L}{6N} + \frac{3}{2} \quad : \tag{42}$$

An accurate approximation for the value of L satisfying this equation for $L \quad 1.5N$ can be obtained by writing L as $fN/(1+a)$, using $\ln(1+a) \quad a$, and neglecting the $L/(6N)$ term, yielding the following equation for a:

$$a^2 + \quad 1 + \frac{f}{4} \quad a + \frac{f}{4} \quad \ln(2=f) + \frac{3}{2} \quad = 0: \tag{43}$$

Solving for a gives

$$L \quad \frac{2fN}{1 \quad f=4 + \sqrt{(1 + f=4)^2 \quad f(\ln(2=f) + 3=2 \quad)}}: \tag{44}$$

The relation $L \quad 1.5N$ corresponds to an upper bound on f of about 0.68. Substitution into (40) and (41) yields approximations for the insertion and hit rates, respectively. For small/moderate f (e.g., $f \quad 0.2$, so that the cache capacity is at most 20% of the objects), a simpler approximation for H is $(f/2)\ln(c/f)$ where c is a suitable constant such as 4.5. Note the considerable contrast between the scaling of hit rate with cache size for $\alpha = 1$ versus $\alpha = 0.5$. Also, when $\alpha = 0.5$ there is a bigger gap with respect to the hit rate when the cache is kept filled with the most popular objects. In this case, the hit rate is $\sum_{i=1}^{\lfloor C \rfloor} p_i \quad p\overline{C}/p\overline{N}$. When $C = fN$, this equals $p\overline{f}$.

3.3.3 Cache on 2nd Request, W=L, Zipf with $\alpha = 1$

Consider now the case of $W = L$, and a Zipf object popularity distribution with $\alpha = 1$. Applying (20) to (5), and (23) to (6) and (7), yields

$$A \quad \frac{(2\ln 2 \quad L=(N(\ln N +)))L}{\ln N + }; \tag{45}$$

$$H \quad \frac{\ln(L=(\ln N +)) + 2 \quad \ln 2}{\ln N + } \tag{46}$$

$$I \quad \frac{\ln 2 \quad L=(N(\ln N +))}{\ln N + }: \tag{47}$$

With respect to the range of values for L for which these approximations are accurate, note that, when $W = L$, (5), (6), and (7) include both $(1 \quad p_i)^L$ and $(1 \quad p_i)^{2L}$ terms. Therefore, when L is substantially smaller than $N(\ln N + \gamma)$, but $2L$ is not, the accuracy of these approximations is uncertain *a priori*, and requires experimental assessment. A similar issue arises in the case of $\alpha = 0.5$, and for *Cache on* k^{th} *request* with $k > 2$.

Equating the corresponding LRU cache capacity C to the approximation for the average number A of objects in the

cache as given in (45), solving for L, and then applying the approximation $\sqrt{1-x} \approx 1 - x/2 - x^2/8$ for small x yields:

$$L \approx \frac{C(\ln N + \gamma)(1 + C/(4(\ln 2)^2 N))}{2 \ln 2}. \quad (48)$$

If the cache capacity $C = N^\beta$ for $0 < \beta < 1$, substituting from (48) into the expression for H in (46) yields an approximation for the cache hit rate which for large N is very close to β:

$$H \approx \frac{\ln(4 \ln 2) - \ln(1 + 1/(4(\ln 2)^2 N^{1-\beta})) - (2\gamma)}{\ln N + \gamma}; \quad (49)$$

while substitution into the expression for I in (47) yields an approximation for the cache insertion rate.

3.3.4 Cache on 2nd Request, W=L, Zipf with $\alpha = 0.5$

For the case of $W = L$ and a Zipf object popularity distribution with $\alpha = 0.5$, applying (28) to (5), and (31) to (6) and (7), yields:

$$A \approx \frac{L^2}{2N} - \ln(N/(2L)) + \frac{L}{2N} + \frac{3}{2} - \gamma; \quad (50)$$

$$H \approx \frac{L}{N} - \ln 2 - \frac{L}{4N}; \quad (51)$$

$$I \approx \frac{L}{2N} - \ln(N/(2L)) + \frac{3L}{4N} + 1 - \gamma. \quad (52)$$

Suppose now that the corresponding LRU cache capacity $C = fN$ for some $f > 0$. Equating the cache capacity C to the approximation for the average number A of objects in the cache as given in (50), writing L as $f^{\frac{1}{2}} N/(1+a)$, and employing the approximations $\ln(1+a) \approx a$ and $1/(1+a) \approx 1 - a$ yields the following equation for a:

$$2a^2 + \left(3 + \frac{f^{\frac{1}{2}}}{2} - \gamma\right) a + \ln(2f^{\frac{1}{2}}) - \frac{f^{\frac{1}{2}}}{2} + \frac{1}{2} + \gamma = 0. \quad (53)$$

Solving for a gives

$$L \approx \frac{\sqrt{4f^{\frac{1}{2}} N}}{1 - \frac{f^{\frac{1}{2}}}{2} + \sqrt{\left(3 + \frac{f^{\frac{1}{2}}}{2} - \gamma\right)^2 - 8(\ln(2f^{\frac{1}{2}}) - \frac{f^{\frac{1}{2}}}{2} + \frac{1}{2} + \gamma)}}. \quad (54)$$

Substitution into the expressions for H and I in (51) and (52) yields approximations for the hit and insertion rates. For small/moderate f, a rough approximation for H is $cf^{1/2}$ where c is a suitable constant such as 0.7.

3.3.5 Cache on kth Request, W=L, Zipf with $\alpha = 1$

Consider now the case of $W = L$, and a Zipf object popularity distribution with $\alpha = 1$. Writing out $(1 - (1 - p_i)^L)^k$ as a polynomial in $(1 - p_i)^L$ and applying (20) yields, for $k \geq 3$, the following approximation for the average number of objects in the cache:

$$A \approx \frac{\sum_{j=2}^{k} (-1)^j \binom{k}{j} j \ln(j) - L}{\ln N + \gamma}. \quad (55)$$

Applying (23) the cache hit rate can be approximated by

$$H \approx \frac{\ln(L/(\ln N + \gamma)) + 2 - \sum_{j=2}^{k} (-1)^j \binom{k}{j} \ln(j)}{\ln N + \gamma}; \quad (56)$$

and, for $k \geq 3$, the cache insertion rate by

$$I \approx \frac{\sum_{j=2}^{k} (-1)^j \binom{k}{j} \frac{1}{1} \ln(j)}{\ln N + \gamma}. \quad (57)$$

Equating the corresponding LRU cache capacity C to the approximation for the average number of objects in the cache as given by expression (55), and solving for L, yields, for $k \geq 3$,

$$L \approx \frac{C(\ln N + \gamma)}{\sum_{j=2}^{k} (-1)^j \binom{k}{j} j \ln(j)}. \quad (58)$$

If the cache capacity $C = N^\beta$ for $0 < \beta < 1$, substituting from (58) into expression (56) yields, for $k \geq 3$, an approximation for the cache hit rate which for large N is very close to β, while the cache insertion rate can be approximated using expression (57).

3.3.6 Cache on kth Request, W=L, Zipf with $\alpha = 0.5$

For the case of $W = L$ and a Zipf object popularity distribution with $\alpha = 0.5$, applying (28) to the equation for A in (14) yields:

$$A \approx \begin{cases} (9 \ln 3 - 12 \ln 2 - L/N) L^2/(4N) & k = 3; \\ \sum_{j=2}^{k} (-1)^{j+1} \binom{k}{j} j^2 \ln(j) - L^2/(4N) & k \geq 4. \end{cases} \quad (59)$$

Applying (31) to the equation for H in (14) yields the following approximation for the cache hit rate for $k \geq 3$:

$$H \approx \frac{L}{2N} \sum_{j=2}^{k} (-1)^j \binom{k}{j} j \ln(j). \quad (60)$$

Applying (31) to the equation for I in (14) yields:

$$I \approx \begin{cases} (L/(2N)) 3 \ln 3 - 4 \ln 2 - \frac{L}{2N} & k = 3; \\ (L/(2N)) \sum_{j=1}^{k-1} (-1)^j \binom{k}{j} (j+1) \ln(j+1) & k \geq 4. \end{cases} \quad (61)$$

Suppose now that the corresponding LRU cache capacity $C = fN$ for some $f > 0$. Equating the cache capacity C to the approximation for the average number of objects in the cache given in (59) for $k \geq 4$, and solving for L, yields

$$L \approx \frac{2f^{\frac{1}{2}} N}{\sum_{j=2}^{k} (-1)^{j+1} \binom{k}{j} j^2 \ln(j)}. \quad (62)$$

Substitution into (61) ($k \geq 4$ case) and (60) yields approximations for the insertion and hit rates. Note that the approximation for H simplifies in this case to $cf^{1/2}$ where c is a k-dependent constant.

Equating the corresponding LRU cache capacity $C = fN$ to the approximation for the average number of objects in the cache given in (59) for $k = 3$, writing L as $2f^{\frac{1}{2}} N/(1+a)$, and employing the approximation $1/(1+a) \approx 1 - a + a^2$ for the L/N term, yields the following equation for a:

$$(1 + 2f^{\frac{1}{2}})a^2 + 2(1 - f^{\frac{1}{2}})a + 1 + 2f^{\frac{1}{2}} - 9 \ln 3 + 12 \ln 2 = 0. \quad (63)$$

Solving for a gives, for $k = 3$,

$$L \approx \frac{2f^{\frac{1}{2}}(1 + 2f^{\frac{1}{2}})N}{3f^{\frac{1}{2}} + \sqrt{(f^{\frac{1}{2}} - 1)^2 - (1 + 2f^{\frac{1}{2}})(1 + 2f^{\frac{1}{2}} - 9 \ln 3 + 12 \ln 2)}}. \quad (64)$$

Substitution into (61) ($k = 3$ case) and (60) yield approximations for the cache insertion and hit rates, respectively. As with *Cache on 2^{nd} request*, a rough approximation for H is $cf^{1/2}$ for a constant c.

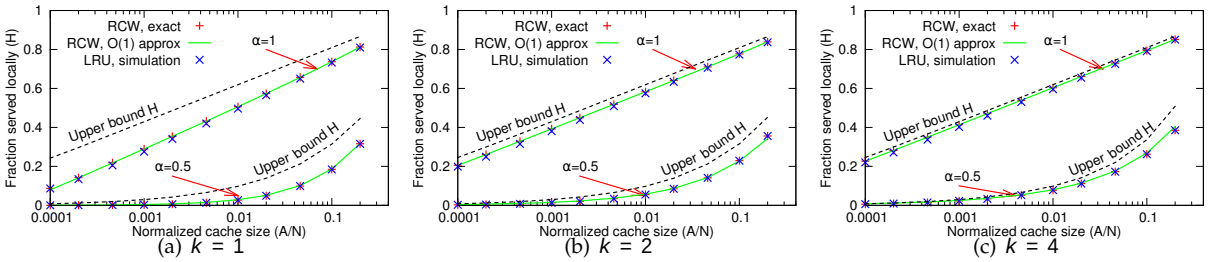


Fig. 1. Performance of *Cache on k^{th} request* (N = 100; 000); dashed lines show hit rate when cache is kept filled with the bCc most popular objects.

3.4 Validation and Performance Results

We have used simulations to validate the exact RCW models. However, since the simulation results for RCW ("RCW sim") and the exact analytic results ("RCW exact") end up being the same (and the simulations just validate the exact expressions) we only show one of the two in the following figures. Instead, these following figures focus on the comparisons between the simulated values of LRU (i.e., "LRU sim") and the exact (or simulated) RCW values (again "RCW exact" has the same values as "RCW sim") and $O(1)$ approximations (i.e., "RCW approx.").

Figure 1 compares our exact RCW cache hit rate results (red '+' markers), using $W{=}L$, with the results from simulations of corresponding fixed-capacity LRU caches (blue ' ' markers), for $N{=}100,000$ objects, different k, and over a large range of cache sizes ($A/N{=}0.0001$ corresponds here to $A{=}10$). Also shown in the figure are the *upper bound* hit rate (dashed black lines), corresponding to when the cache is kept filled with the bCc most popular objects, and $O(1)$ *approximations* (green lines) derived in Section 3.3 for the special cases of Zipf distributions with $\alpha{=}1$ and $\alpha{=}0.5$. We note that popularity skew typically is intermediate between these two cases.

For the simulations, we set the LRU cache size C equals A. To match use of $W{=}L$ in the case of the RCW caches, we assume an implementation of LRU with *Cache on k^{th} request* in which, when the cache is full (as it is in steady state), W is dynamically set to the number of requests since the "least recently requested" object currently in the cache was last requested. Similar to an RCW cache with $W{=}L$, this choice ensures that an object remains a "caching candidate" as long as it is requested at least as recently as the least recently requested object in the cache. For the simulation results reported here and in subsequent sections, each simulation was run for six million requests, with the statistics for the initial two million requests removed from the measurements.

The following observations stand out. First, for all cases (including larger k), the exact RCW results closely match the LRU simulation results. This shows that the RCW analysis (presented here) can be used as an effective method to approximate the performance of an LRU cache.

Second, the $O(1)$ approximations are relatively accurate, for both cases where caching is quite effective ($\alpha{=}1$) and largely ineffective ($\alpha{=}0.5$). For relative insertion rates (not shown), the $O(1)$ diverge somewhat more, but errors remain within 10% for all cases except for the cases of (i) small cache sizes ($A/N < 0.001$) when $k{=}4$ and $\alpha{=}0.5$, and (ii) large cache sizes ($A/N > 0.1$) when $k{=}4$ and $\alpha{=}0.5$. Here, it is important to note that accurate approximations

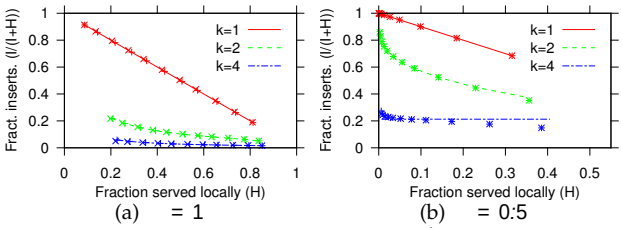


Fig. 2. Tradeoff curves for different *Cache on k^{th} request* policies. Here, N = 100; 000 and we use the following markers: "RCW, exact" (+), "RCW, approx" (line), and LRU ( ).

for insertion rates with larger k are a more difficult problem, since summations including larger powers of $(1 \quad p_i)$ need to be approximated (see equations (14)). This is particularly an issue for $\alpha{=}0.5$ since the basic summation approximations we use (Section 3.2) for $\alpha = 0.5$ place tighter constraints on the powers of $(1 \quad p_i)$ for which our basic summation approximations can be expected to be most accurate (powers substantially less than $2N$ rather than substantially less than $N(\ln N + \gamma)$ as is the case for $\alpha{=}1$; compare the assumptions used for equations (28) and (31) versus equations (20) and (23)). For $\alpha{=}1$, the relative insertion rate errors remain within 5%, and for $k{=}1$ (regardless of α) the errors are within 0.3%. When discussing these $O(1)$ approximations, it is also important to remember that the exact RCW results (which as per our first observation provide good results for all cases) can be used to approximate LRU.

Third, the gap in hit rate between the policies and the upper bound is substantial with $k{=}1$ (regular LRU), narrows with $k{=}2$, and is almost eliminated with $k{=}4$, leaving little room for further hit rate improvements.

While comparing the different figures suggest that further increasing k beyond $k{=}2$ yields only small additional improvements in the hit rates, it should be noted that larger improvements are seen in the insertion fraction and that these improvements continue (at least with respect to relative rather than absolute differences) as k is increased. To illustrate this, Figure 2 shows the tradeoff between hit rate (on x-axis) and insertion fraction (y-axis) for different k. Note that with selective cache insertion policies (i.e., larger k), the same hit rate can be achieved with a much lower insertion fraction.

4 Dynamic Instantiation Analysis

In the following, we present an exact analysis for the cache hit rates and insertion rates during the transient periods for the cases of *Cache on k^{th} request* when $k = 1$ (Section 4.1) and $k \quad 2$ (Section 4.2), respectively, and derive the corresponding $O(1)$ approximations (Section 4.3). Performance results are presented in Section 4.4.

4.1 Cache on 1st Request

Consider now the case where the cache is allocated for only a portion of the time period, and is initially empty when instantiated. With *Cache on 1^{st} request*, after the first L requests following instantiation, the cache will have the occupancy probabilities derived earlier for the "always-on" case in Section 3, and so for requests following the first L requests the analysis in Section 3 can be used. The average insertion rate over the first L requests (the transient period) is given by the expression for the average number A of objects in cache from (1), divided by L. Denoting the average hit rate during the transient period by $\bar{H}_{\text{transient}}$, this gives:

$$H_{\text{transient}} = 1 \quad \frac{A}{L} = 1 \quad \frac{N \quad \sum_{i=1}^{N}(1 \quad p_i)^L}{L}; \tag{65}$$

and from equation (1) for H, assuming that $\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \quad L$,

$$H_{t_a:t_d} = \frac{L H_{\text{transient}} + \int_{t_a}^{t_d} (t)\mathrm{d}t \quad L \quad 1 \quad \sum_{i=1}^{N} p_i(1 \quad p_i)^L}{\int_{t_a}^{t_d} (t)\mathrm{d}t}: \tag{66}$$

Finally, for *Cache on 1^{st} request*, $\bar{I}_{\text{transient}}$ and $\bar{I}_{t_a:t_d}$ are given simply by $1 \quad \bar{H}_{\text{transient}}$ and $1 \quad \bar{H}_{t_a:t_d}$.

4.2 Cache on kth Request ($k \quad 2$)

As described in Section 2, *Cache on k^{th} request* requires maintenance of state information regarding "caching candidates" and all currently cached objects. We assume that when a cache using *Cache on k^{th} request* is deallocated, the state information of both types is transferred to the upstream system to which requests will now be directed. The upstream system maintains and updates this state information when receiving requests that the cache would have received had it been allocated, and transfers it back when the cache is instantiated again. Therefore, although the cache is initially empty when instantiated, it can use the acquired state information to selectively cache newly requested objects, caching a requested object not present in the cache, whenever that object should be in (or be put in) the cache according to its state information. Note that after the first L requests following instantiation, the cache will have the cache occupancy probabilities derived earlier for the "always-on" case, and so for requests following the first L requests the analysis in Section 3 can be used.

Note that over the transient period consisting of the first L requests, no objects are removed from the cache. The average insertion rate during the transient period $\bar{I}_{\text{transient}}$ is therefore given by the average number A of objects in cache (from (5) for k=2 and (11) for general k), divided by L. Under the assumption that $\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \quad L$, it is then straightforward to combine $\bar{I}_{\text{transient}}$ with the always-on insertion rate from Section 3 to obtain $\bar{I}_{t_a:t_d}$.

The average hit rate during the transient period is given by one minus the average transient period insertion rate, minus the average probability that a requested object is not present in the cache and should not be inserted. Recall that the cache receives up-to-date state information when instantiated, and a requested object is cached according to this state information. Therefore, a requested object is not present in the cache and should not be inserted, if and only if it would not be in the cache and would not be inserted into the cache on this request with an always-on cache. The probability of this case is equal to one minus the hit rate for an always-on cache minus the insertion rate for an always-on cache. The above implies that the average hit rate during the transient period, $\bar{H}_{\text{transient}}$, is given by the always-on cache hit rate (in (12)) plus the always-on cache insertion rate (in (13)) minus the average transient period insertion rate $\bar{I}_{\text{transient}}$; i.e., $\int_{t_a}^{t_d} \bar{H}_{\text{transient}} = H + I \quad A/L$. Under the assumption that $\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \quad L$, it is then straightforward to combine $\bar{H}_{\text{transient}}$ with the always-on hit rate (in (12)) to obtain $\bar{H}_{t_a:t_d}$.

4.3 Dynamic Instantiation Approximations

We next derive and analyze our easy-to-compute $O(1)$-approximations for the transient period. A summary of these results are provided in Table 2.

4.3.1 Cache on 1st Request

For the special case of a Zipf object popularity distribution with $\alpha = 1$, using (32) to substitute for A in $\bar{H}_{\text{transient}} = 1 \quad \frac{A}{L}$ yields

$$H_{\text{transient}} \quad \frac{\ln(L=(\ln N +)) + 2 \quad 1 \quad L=(2N(\ln N +))}{\ln N + }: \tag{67}$$

The ratio of the average cache hit rate over the transient period to the hit rate once the cache has filled can yield substantial insight into the impact of the transient period on performance. In this case, from the above expression and expression (34) this ratio is approximately $1 \quad (1 \quad L/(2N(\ln N + \gamma)))/(\ln N + \gamma)$.

For $\alpha = 0.5$, using (39) to substitute for A yields

$$H_{\text{transient}} \quad \frac{L}{4N} \quad \ln((2N)=L) + \frac{L}{6N} + \frac{3}{2} \quad : \tag{68}$$

In this case the ratio of the average hit rate over the transient period to the hit rate once the cache has filled (given in (41)) is between 0.5 and 0.7 (for $0 < L < 2N$), substantially smaller than for $\alpha = 1$.

4.3.2 Cache on kth Request ($k \quad 2$)

For $k = 2$, $W = L$, and a Zipf object popularity distribution with $\alpha = 1$, applying the expressions for H and I in (46) and (47), and using the expression for A in (45) to substitute for A in the transient period insertion rate A/L, yields an approximation for the average hit rate during the transient period of

$$\begin{aligned} H_{\text{transient}} \quad & \frac{\ln(L=(\ln N +)) + 2 \quad \ln 2}{\ln N + } \\ & + \frac{\ln 2 \quad L=(N(\ln N +))}{\ln N + } \quad \frac{2\ln 2 \quad L=(N(\ln N +))}{\ln N + } \\ = \ & \frac{\ln(L=(\ln N +)) + 2 \quad 2\ln 2}{\ln N + }: \end{aligned} \tag{69}$$

The ratio of the cache hit rate over the transient period to the hit rate once the cache has filled (given in (46)) is therefore approximately $1 \quad (\ln 2)/(\ln(L/(\ln N+\gamma))+2\gamma \quad \ln 2)$. In contrast, for $\alpha = 0.5$ and $k = 2$, applying the expressions for the hit and insertion rates in (51) and (52), and the expression for A in (50) to substitute for A in the transient period insertion rate A/L, yields an approximation for the average hit rate over the transient period of

$$H_{\text{transient}} \quad \frac{L}{N} \quad \ln 2 \quad \frac{L}{8N} \quad \frac{1}{4} \quad : \tag{70}$$

From comparison with the hit rate expression in (51), the ratio of the average cache hit rate over the transient period

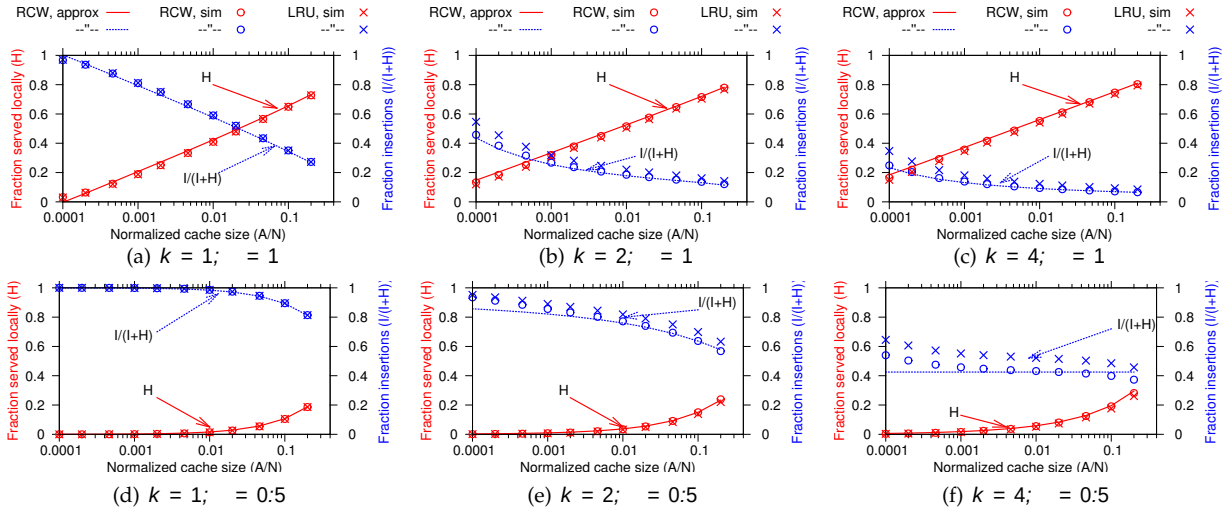


Fig. 3. Transient period results. Performance of *Cache on k^{th} request* (N = 100,000) during transient period.

to the hit rate once the cache has filled is between about 0.64 and 0.72 (considering here $0 < L < N$), substantially smaller than for $\alpha = 1$. Results similar in nature are obtained for k 3, applying (56), (57), and (55) for $\alpha = 1$, and (60), (61), and (59) for $\alpha = 0.5$.

4.4 Transient Period Performance Results

Figure 3 shows sample results for the transient period when using *Cache on k^{th} request* with different $k = 1, 2, 4$ and Zipf with $\alpha = 0.5$ or 1. In all experiments, we used $W{=}L$ and show results only for the transient period itself. For the analytic expressions, we used the $O(1)$ approximations for each metric. For the simulations, we start with an empty cache, and simulate the system until the system reaches steady-state conditions. At that time, we empty the cache and begin a new transient period. This is repeated for 2,000 transient periods or until we have simulated 6,000,000 requests, whichever occurs first, and statistics are reported based on fully completed transient periods. With these settings, each data point was calculated based on at least 17 transient periods. (This occurred with $A/N{=}0.2$, $k{=}4$, and $\alpha{=}1$.) To improve readability, as in prior figures, confidence intervals are not included. However, in general, the confidence intervals are tight (e.g., 0.0016 for the data point mentioned above).

For the RCW simulations, the system maintains the same state information and operates in the same way as described for our analysis assumptions. As in the steady-state simulations, for the corresponding LRU cache, the capacity C was set equal to A. However, rather than dynamically setting W to the number of requests since the "least recently requested" object currently in the cache was last requested (corresponding to $W = L$), as was done for the steady-state simulations, we dynamically inflate W by one for each request being made during the transient period. This ensures that no per-object counters are reset during the transient phase, regardless of how long the transient period is. The cache reaches steady state conditions when it is filled completely. Otherwise the implementation is the same as for the RCW simulations, including that objects are added to the cache whenever their counter reaches k.

The transient results very much resemble the steady-state results. For example, the tradeoff curves in Figure 3 are very similar to those observed in Figure 1, and the analytic approximations again nicely match the simulated RCW values for most instances (which themselves nicely match the exact analysis results). Most importantly, there is a very good match for all hit rate results (red curves/markers: RCW approximations, RCW simulations, and LRU simulations); the metric that we will use in the optimization models (Section 5) and the evaluation thereof (Section 6). Substantive differences between the RCW simulations and analytic approximations are observed only for the insertion fraction metric (shown in blue) when using very small cache sizes (e.g., A/N less than 0.001) when $\alpha{=}0.5$. When $k{=}4$ and $\alpha{=}0.5$, we also observe some noticeable differences in the insertion fraction between RCW and LRU. This may suggest that when k is large, RCW is a worse approximation for LRU (as we compare them) during transient periods than during steady state. The main reason for this is related to the difficulty in selecting a good W for the LRU cache during the transient period. By ensuring that all requests made during the transient period increase a per-object request counter, our implementation of the transient period operation of the LRU cache provides a clear implementation of a *Cache on k^{th} request* policy. However, with $k > 1$, this implementation ends up adding objects to the cache at a higher rate than the RCW cache. This results in larger transient-period insertion fractions than with RCW when $k > 1$, with the difference becoming larger for larger k. A less aggressive policy may be to freeze W during the transient period (based on the last observed value before the cache was released) and only use our current policy for the initial transient period (when W is still unknown). However, this approach can become sensitive to the value that W has at the moment when a cache is released. Cache performance comparisons of this and other alternative LRU policy variations provide interesting future work but are outside the scope of this paper. Yet, for the purpose of this paper, for all considered k and α, we find the approximations sufficiently accurate to justify using them for our optimization of dynamic cache instantiation. Again, in the following sections, we will leverage the (more accurate) hit rate results.

5 Optimization Models for Dynamic Instantiation

Consider now the problem of jointly optimizing the capacity C of a dynamically instantiated cache, and the interval over which the cache is allocated, so as to minimize the cache cost subject to achieving a target fraction of requests $H_{\min}$ ($0 < H_{\min} < 1$) that will be served locally:

$$\text{minimize} \quad (t_d - t_a)(C + b); \tag{71}$$

$$\text{subject to} \quad \frac{H_{t_a:t_d}}{\int_0^T \lambda(t)\mathrm{d}t} \quad H_{\min}.$$

Note that a smaller cache has the advantages of a shorter transient period until it fills and lower cost per unit time, while a larger cache has the advantage of a higher hit rate once filled. For convenience, in the following we assume that $\lambda(t) > 0, \forall t$.

In this section, using the above optimization formulation, we derive a cost lower bound (Section 5.1), present bounds specialized to Zipf popularity distributions (Section 5.2), and present both policy-based cost optimizations (Section 5.3) and their corresponding approximate cost optimizations (Section 5.4). These results are then used in the performance evaluation of different dynamic instantiation solutions presented in Section 6.

5.1 Lower Bound

A lower bound on cost can be obtained by using an upper bound for the average hit rate over the cache allocation interval. One such bound can be obtained by assuming that there is a hit whenever the requested object is one that has been requested previously, since the cache was allocated. We apply this bound to obtain a lower bound on the duration of the cache allocation interval. Another bound is the hit rate when the bCc most popular objects are present in the cache. We apply this bound to the more constrained optimization problem that results from our use of the first bound.

Denote by $\bar{H}_R$ the average hit rate over the first R requests after the cache has been allocated. At best, request r, $1 \quad r \quad R$ is a hit if and only if the requested object was the object requested by one or more of the $r \quad 1$ earlier requests, giving:

$$H_R \quad \frac{1}{R}\sum_{r=1}^{R}\sum_{i=1}^{N} p_i(1-(1-p_i)^{r-1}) = 1 \quad \frac{1}{R}\left(N - \sum_{i=1}^{N}(1-p_i)^R\right). \tag{72}$$

Since this is a concave function of R, we can bound the average hit rate over the cache allocation interval by setting $R = \int_{t_a}^{t_d}\lambda(t)\mathrm{d}t$, the expected value of the number of requests within this interval. Applying this bound to the hit rate constraint in (71) yields

$$\frac{\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t}{\int_0^T \lambda(t)\mathrm{d}t} \quad 1 \quad \frac{1}{R}\left(N - \sum_{i=1}^{N}(1-p_i)^R\right) \quad H_{\min}; \tag{73}$$

implying that

$$\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t + \sum_{i=1}^{N}(1-p_i)^{\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t} \quad \int_0^T \lambda(t)\mathrm{d}t \quad H_{\min} + N. \tag{74}$$

Given that we choose t_a and t_d as the beginning and end, respectively, of a time interval with the largest average request rate, the left-hand side is a strictly increasing function of $t_d \quad t_a$, as can be verified by taking the derivative with respect to $\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t$, noting that this derivative is minimized for minimum $\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t$ (which is at least one, in the region of interest), and using the fact that $\ln(x)$ is a convex function. Therefore, for any particular workload this relation defines a lower bound D_l for the interval duration $t_d \quad t_a$.

Applying now the upper bound on hit rate from when the bCc most popular objects are present in the cache, gives the following optimization problem:

$$\text{minimize} \quad (t_d - t_a)(C + b); \tag{75}$$

$$\text{subject to} \quad \frac{\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t}{\int_0^T \lambda(t)\mathrm{d}t} \quad \sum_{i=1}^{bCc} p_i \quad H_{\min}; \quad D_l \quad t_d \quad t_a \quad T.$$

Solution of this problem yields a lower bound on cost. Specializations of this problem for the cases of Zipf popularity distributions with α = 1 and 0.5 are developed next.

5.2 Bounds for Zipf Popularity Distributions

5.2.1 Zipf with = 1

Consider now the special case of a Zipf object popularity distribution with parameter $\alpha = 1$, and denote the normalization constant $\sum_{i=1}^{N} 1/i$ by Ω. In the case that $R < (N+1)(\ln(N+1)+\gamma)$, we have

$$\begin{aligned}\sum_{i=1}^{N}(1-p_i)^R &= \sum_{i=1}^{N}\left(1-\frac{1}{i}\right)^{(i)\frac{R}{i}} \quad \sum_{i=1}^{N} e^{-R/(i)} \\ &< \int_1^{N+1} e^{-R/(x)}\mathrm{d}x < \int_1^{N+1} e^{-R/(x(\ln(N+1)+))}\mathrm{d}x \\ &< (N+1)\left(-\frac{R}{\ln(N+1)+}\right) \quad \ln\left(\frac{R}{\ln(N+1)+}\right) + 2 \quad 1;\end{aligned} \tag{76}$$

where the second last inequality uses $\Omega < \ln(N+1)+\gamma$, and the last inequality follows from the Taylor series expansion (as in (16) in the Section 3.2) under the assumption that $R < (N+1)(\ln(N+1)+\gamma)$. Using $R = \int_{t_a}^{t_d}\lambda(t)\mathrm{d}t$, under the assumption that $R = \int_{t_a}^{t_d}\lambda(t)\mathrm{d}t < (N+1)(\ln(N+1)+\gamma)$, we can substitute into (74) to obtain:

$$\frac{\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t}{\ln(N+1)+} \quad \ln\left(\frac{\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t}{\ln(N+1)+}\right) + 2 \quad 1 \quad \int_0^T \lambda(t)\mathrm{d}t \quad H_{\min} \quad 1. \tag{77}$$

When the right-hand side of this relation is positive, which it is for parameters of interest, the left-hand side must be a strictly increasing function of $t_d \quad t_a$. Under the assumption that $\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t < (N+1)(\ln(N+1)+\gamma)$, a lower bound D_l^0 for $t_d \quad t_a$ can therefore be obtained from this relation for any particular workload of interest (setting $D_l^0 = 1$ if no value for $t_d \quad t_a \quad T$ satisfies this relation). Denoting by D_l^{00} the maximum value of $t_d \quad t_a$ such that $\int_{t_a}^{t_d}\lambda(t)\mathrm{d}t$ $(N+1)(\ln(N+1)+\gamma)$, with $D_l^{00} = 1$ if this still holds for $t_d \quad t_a = T$, a lower bound D_l for $t_d \quad t_a$ is given by:

$$D_l = \min\ D_l^0; D_l^{00}\ . \tag{78}$$

Also, for the special case of a Zipf object popularity distribution with parameter $\alpha = 1$ and $C \quad N$, $\sum_{i=1}^{bCc} p_i < (\ln(C+1)+\gamma)/(\ln N+\gamma)$. Applying this bound to the hit rate constraint in (75) yields the following optimization problem:

$$\text{minimize} \quad (t_d - t_a)(C + b); \tag{79}$$

subject to

$$\frac{\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t}{\int_0^T \lambda(t)\mathrm{d}t} \left(\frac{\ln(C+1)+\gamma}{\ln N+\gamma} \right) \leq H_{\min};$$
$$C \leq N; \quad D_l \leq t_d - t_a \leq T;$$

where D_l is given by (78). This optimization problem can be further specialized to any particular workload of interest by specifying, as a function of the duration $D = t_d - t_a \leq T$, the average request rate that the cache would experience should it be allocated for the interval of duration D, within the time period under consideration, with the highest average request rate. It is then straightforward to solve the optimization problem to any desired degree of precision. The computational cost of evaluating the optimization function and checking the constraints is $O(1)$, and it is feasible to simply search over all choices of C and the duration $t_d - t_a$ of the cache allocation interval, at some desired granularity, to find the choices that satisfy the constraints (should any such choices exist) with lowest cost.

5.2.2 Zipf with $\alpha = 0.5$

For a Zipf object popularity distribution with $\alpha = 0.5$, the normalization constant $\Omega = \sum_{i=1}^{N} 1/\sqrt{i}$. In the case that $R < 2(N+1)$ we have

$$\sum_{i=1}^{N} (1-p_i)^R = \sum_{i=1}^{N} 1 - \frac{1}{(\sqrt{i})} \left(\left(\sqrt{i} \right) \right)^{\frac{R}{(\sqrt{i})}}$$
$$\sum_{i=1}^{N} e^{\frac{R}{(\sqrt{i})}} < \int_1^{N+1} e^{R/\left(\left(\sqrt{x} \right) \right)} \mathrm{d}x < \int_1^{N+1} e^{R/(2\sqrt{x}-\sqrt{N+1})} \mathrm{d}x$$
$$< N + 1 - R + \frac{R^2}{4(N+1)} \ln \frac{2(N+1)}{R} + \frac{R}{6(N+1)} + \frac{3}{2}; \tag{80}$$

where the second last inequality uses $\Omega < 2\sqrt{N+1}$, and the last inequality follows from the Taylor series expansion (as in (25) in the Section 3.2) under the assumption that $R < 2(N+1)$. Using $R = \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t$, under the assumption that $R = \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t < 2(N+1)$, we can substitute into (74) to obtain:

$$\frac{\left(\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \right)^2}{4(N+1)} \ln \frac{2(N+1)}{\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t} + \frac{\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t}{6(N+1)} + \frac{3}{2} \leq \int_0^T \lambda(t)\mathrm{d}t \leq H_{\min} - 1. \tag{81}$$

The left-hand side of this relation is a strictly increasing function of $t_d - t_a$. Under the assumption that $\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t < 2(N+1)$, a lower bound D_l^0 for $t_d - t_a$ can therefore be obtained from this relation for any particular workload of interest (setting $D_l^0 = 1$ if no value for $t_d - t_a \leq T$ satisfies this relation). Denoting by D_l^{00} the maximum value of $t_d - t_a$ such that $\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \leq 2(N+1)$, with $D_l^{00} = 1$ if this still holds for $t_d - t_a = T$, a lower bound D_l for $t_d - t_a$ is given by $D_l = \min[D_l^0, D_l^{00}]$.

Also, for the special case of a Zipf object popularity distribution with parameter $\alpha = 0.5$ and $C \leq N$,

$$\sum_{i=1}^{C} p_i < \frac{2\sqrt{C+\frac{1}{2}} - \gamma - \frac{1}{2}}{2\sqrt{N+1} - 2} = \frac{\sqrt{C+\frac{1}{2}} - \gamma - \frac{1}{2}}{\sqrt{N+1} - 1}; \quad C \leq N. \tag{82}$$

Applying the bound in (82) to the hit rate constraint in (75) yields the following optimization problem:

$$\text{minimize} \quad (t_d - t_a)(C+b); \tag{83}$$

subject to

$$\frac{\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t}{\int_0^T \lambda(t)\mathrm{d}t} \left(\frac{\sqrt{C+\frac{1}{2}} - \gamma - \frac{1}{2}}{\sqrt{N+1} - 1} \right) \leq H_{\min};$$
$$C \leq N; \quad D_l \leq t_d - t_a \leq T;$$

As before, it is straightforward to specialize this optimization problem to any particular workload of interest, and to then solve it to any desired degree of precision.

5.3 Policy-based Cost Optimizations

Cache on 1^{st} request: For an LRU cache using this policy, equating the cache capacity C to the average occupancy A of an RCW cache and applying (66) yields:

$$\text{minimize} \quad (t_d - t_a)(C+b); \tag{84}$$

$$\text{subject to} \quad C = N - \sum_{i=1}^{N} (1-p_i)^L; \quad L \leq \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t;$$
$$\frac{L - C + \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t - L \left(1 - \sum_{i=1}^{N} p_i (1-p_i)^L \right)}{\int_0^T \lambda(t)\mathrm{d}t} \leq H_{\min}.$$

Similarly as for the lower bound, specializations of this problem for the cases of Zipf popularity distributions with $\alpha = 1$ and 0.5 are developed in Section 5.4.

Cache on k^{th} request: Similarly, equating the capacity C to the average occupancy A of an RCW cache and applying the Section 4.2 analysis yields the optimization problem:

$$\text{minimize} \quad (t_d - t_a)(C+b); \tag{85}$$

subject to

$$C = \sum_{i=1}^{N} \frac{1 - (1-p_i)^L}{1 - (1-p_i)^L + \frac{(1-p_i)^L (1 - (1-(1-p_i)^W)^k)}{(1-p_i)^W (1-(1-p_i)^W)^k}}; \quad L \leq \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t;$$
$$\frac{\int_{t_a}^{t_d} \lambda(t)\mathrm{d}t \sum_{i=1}^{N} p_i \frac{1-(1-p_i)^L}{1-(1-p_i)^L + \frac{(1-p_i)^L (1-(1-(1-p_i)^W)^k)}{(1-p_i)^W (1-(1-p_i)^W)^k}}}{\int_0^T \lambda(t)\mathrm{d}t}$$
$$+ \frac{L \sum_{i=1}^{N} p_i \frac{(1-p_i)^L}{1-(1-p_i)^L + \frac{(1-p_i)^L (1-(1-(1-p_i)^W)^k)}{(1-p_i)^W (1-(1-p_i)^W)^k}} - C}{\int_0^T \lambda(t)\mathrm{d}t} \leq H_{\min}.$$

Section 5.4 develops specializations of this problem for $W = L$ and the cases of Zipf popularity distributions with $\alpha = 1$ and 0.5.

5.4 Approximate Cost Optimization

Cache on 1st Request: For the special case of a Zipf object popularity distribution with parameter $\alpha = 1$, applying (34) and (37) this becomes:

$$\text{minimize} \quad (t_d - t_a)(C+b); \tag{86}$$

subject to

$$L = \frac{N(\ln N + \gamma)((1-\gamma)\ln N + \gamma - 1)}{((1-\gamma)\ln N + \gamma)((1-\gamma)\ln N - \gamma + \ln((1-\gamma)\ln N + \gamma))};$$
$$= \frac{\ln C}{\ln N}; \quad L \leq \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t;$$
$$\frac{L - C + \int_{t_a}^{t_d} \lambda(t)\mathrm{d}t - L \frac{\ln(L/(\ln N+\gamma)) + 2 - L/(N(\ln N+\gamma))}{\ln N + \gamma}}{\int_0^T \lambda(t)\mathrm{d}t} \leq H_{\min}.$$

Similarly, applying (41) and (44) yields the corresponding optimization problem for $\alpha = 0.5$. As before, it is feasible to simply search over all choices of C and the duration $t_d - t_a$

of the cache allocation interval, at some desired granularity, to find the choices that satisfy the constraints (should any such choices exist) with lowest cost.

Cache on kth Request: For the special case of $W = L$ and a Zipf object popularity distribution with parameter $\alpha = 1$, applying the expressions for H and I in (46) and (47), and (48), yields the following optimization problem for *Cache on* 2^{nd} *request*:

$$\text{minimize} \quad (t_d \quad t_a)(C + b); \tag{87}$$

subject to

$$L = \frac{C(\ln N + \quad)(1 + C=(4(\ln 2)^2 N))}{2 \ln 2}; \quad L \int_{t_a}^{t_d} (t)\mathrm{d}t;$$

$$\frac{L \quad \frac{\ln 2 \quad L=(N(\ln N + \quad))}{\ln N + } \quad C}{\int_0^T (t)\mathrm{d}t + \frac{\int_{t_a}^{t_d} (t)\mathrm{d}t}{\int_0^T (t)\mathrm{d}t} \quad \frac{\ln(L=(\ln N + \quad)) + 2 \quad \ln 2}{\ln N + }} \quad H_{\min}:$$

Similarly, applying (56), (57), and (58) yields the corresponding optimization problem for $k \quad 3$, while applying the expressions for H and I in (51) and (52), and (54) ($k = 2$), and (60), (61), (62), and (64) ($k \quad 3$) yield the corresponding optimization problems for $\alpha = 0.5$.

6 Dynamic Instantiation Performance

For an initial model of request rate variation, we use a single-parameter model in which the request rate increases linearly from a rate of zero at the beginning of the time period to a rate λ_{high} half-way through, and then decreases linearly such that the request rate at the end of the period is back to zero. Default parameter settings (each used unless otherwise stated) are T = 1440 min. (24 hours), $\lambda_{\text{high}} = 20$ req./min., $b = 500$ (and so for a cache capacity of 1000 objects, for example, the size-independent portion of the cache cost contributes half of the total), $H_{min} = 0.4$, $N = 100,000$, and a Zipf object popularity distribution with $\alpha = 1$.

Figures 4(a), (b), and (c) show the ratio of the minimal cost for a dynamically instantiated cache using different cache insertion policies (using $W{=}L$ for the *Cache on* k^{th} *request* policies) to the cost lower bound, as obtained from numerically solving the optimization models of Section 5, as a function of the cost parameter b, the hit rate constraint H_{min}, and the peak request rate λ_{high}, respectively. As *baseline* comparisons, we also include the cost ratios for *Cache on* 1^{st} *request* ($k = 1$) and *Cache on* 2^{nd} *request* ($k = 2$) for the baseline case of a permanently allocated LRU cache with capacity C chosen so as to yield a hit rate of H_{min} (as calculated from our RCW analysis). In each figure, all other parameters are set to their default values.

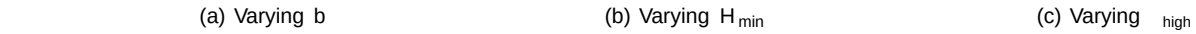
Note that in these results: (1) unless b is very small (in which case, it is most cost-effective to permanently allocate a small cache), H_{min} is large, or λ_{high} is too small for a dynamically instantiated cache to fill, dynamic cache instantiation can yield substantial cost savings; (2) *Cache on* k^{th} *request* for $k \quad 2$ provides a better cost/performance tradeoff curve compared to *Cache on* 1^{st} *request*; and (3) there is only modest room for further improvement through use of more complex cache insertion and replacement policies.

The results also provide some interesting observations with regards to a permanently allocated LRU cache (with optimized cache capacity C). For example, in Figure 4(b) the curve for the *baseline with* $k = 1$ goes down first, since for very low values of H_{min} it is very inefficient to permanently allocate a cache. As H_{min} increases from these very low values, permanently allocating a cache becomes more reasonable. Eventually, however, the baseline curve starts to rise. In this region, permanently allocating a cache is reasonable. However, the required cache capacity C for the *baseline with* $k = 1$ begins to dominate the parameter b (which gives the portion of the cost that is independent of cache size) in the cost expression, and so the inefficiency of a *Cache on* 1^{st} *request* LRU cache in comparison to a cache in which the most popular objects are always kept in the cache becomes more and more apparent (as C gets bigger and bigger relative to b), causing the baseline curve to rise.

The potential benefits of dynamic cache instantiation (and of caching itself) are strongly dependent on the popularity skew. When object popularities follow a Zipf distribution with $\alpha{=}0.5$, with our default parameters it is not even possible to achieve the target fraction of requests H_{min} to be served locally, using dynamic cache instantiation. This is partly due to the fact that for $\alpha{=}0.5$, caching performance is degraded much more severely when in the transient period than for $\alpha{=}1$ (e.g., results in Section 4), and partly due to the fact that a larger cache is required to achieve a given hit rate. The impact of the popularity skew can be clearly seen by comparing the results in Figures 5(a) and (b), which use $N{=}10,000$ instead of the default value of 100,000 so as to allow the hit rate constraint to be met over a significant range of values, even when $\alpha{=}0.5$. In addition to the poorer performance of dynamic cache instantiation that is seen in Figure 5(a), note also the increased gap with respect to the lower bound, and the poorer performance of *Cache on* 2^{nd} *request* relative to *Cache on* 1^{st} *request* (compared to the relative performance seen in Figure 5(b)). (Results for *Cache on* k^{th} *request* for $k = 3, 4$ are not shown in Figure 5(a), since the required value of L becomes too large for all but the smallest cache sizes.)

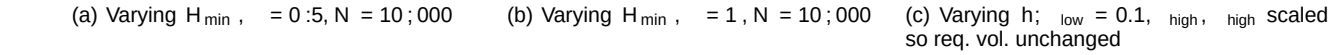
The significant impact of N can be seen by comparing Figures 4(b) and 5(b), which both use $\alpha{=}1$ and differ only in the value of N. Here, we note that smaller N (Figure 5(b)) results in a smaller cache size C being needed to achieve a given hit rate H_{min}. This decreases the relative importance of C compared to that of b in the cost expressions, and correspondingly compresses the gaps between the curves with dynamic cache instantiation but different cache insertion policies.

The extent of rate variability also has a substantial impact. This is illustrated in Figure 5(c), for which our model of request rate variation is modified so that the minimum rate is λ_{low}, $0{<}\lambda_{\text{low}}{<}\lambda_{\text{high}}$, rather than zero, and with linear rate increase/decrease occupying only a fraction $1 \quad jhj$ of the time period, where h is a parameter between $\quad 1$ and 1. When $h{>}0$, the request rate is λ_{low} for the rest of the time period, while when $h{<}0$, the request rate is λ_{high} for the rest of the time period (and so during the fraction $1 \quad jhj$ of the time period the rate first decreases linearly to λ_{low} and then increases linearly back to λ_{high}), giving a peak to mean request rate ratio for $\quad 1{<}h{<}1$ of $2/(1 \quad h+(1+h)\lambda_{\text{low}}/\lambda_{\text{high}})$. Results are shown for varying h, with λ_{low} fixed at 10% of λ_{high}, and λ_{high} scaled for each value of h so as to maintain



(a) Varying b (b) Varying H_{min} (c) Varying α_{high}

Fig. 4. Ratio of the minimal cost to the lower bound versus b, H_{min}, and α_{high} (other parameters at defaults).



(a) Varying H_{min}, $\alpha = 0.5$, $N = 10,000$ (b) Varying H_{min}, $\alpha = 1$, $N = 10,000$ (c) Varying h; $\alpha_{low} = 0.1$, α_{high}, β_{high} scaled so req. vol. unchanged

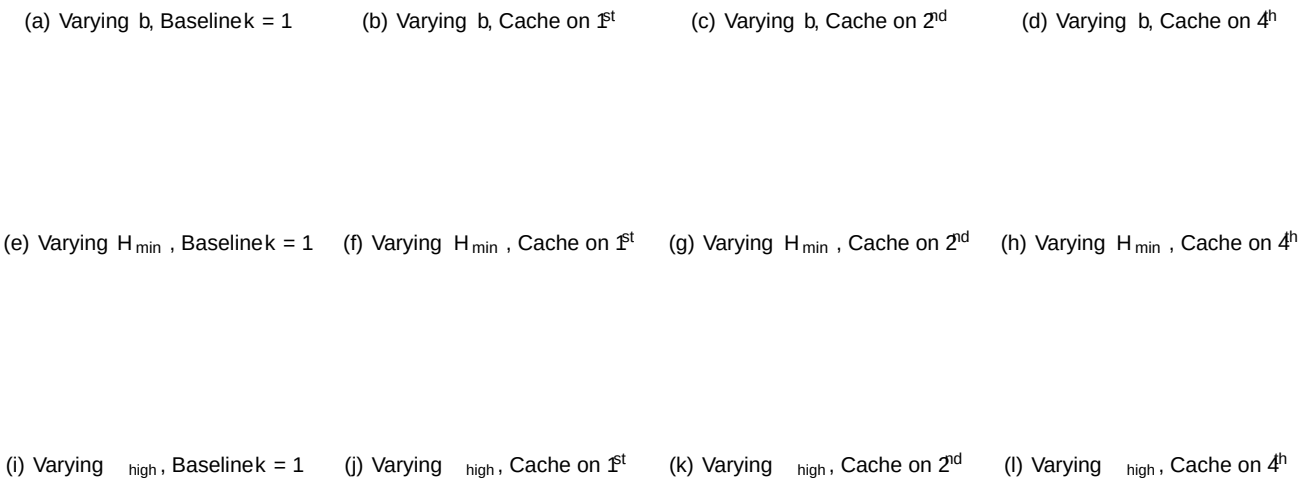
Fig. 5. Impact of popularity skew, number of objects, and request rate variability.

the same total request volume as with the default single-parameter model. Note that $h=1$ and $h=-1$ correspond to the same scenario, since in both cases the request rate is constant throughout the period. Note also that the lower bound becomes overly optimistic for h around 0.7; in this case the requests are highly concentrated, and the solution to the lower bound optimization problem is a large cache allocated for a short period of time (for which the upper bound on hit rate when the bCc most popular objects are present in the cache becomes quite loose). Most importantly, observe that when the pattern of request rate variation is such that there is a substantial "valley" ($h>0$) within the time period during which the request rate is relatively low, the benefits of dynamic instantiation are much higher than when there is a substantial "plateau" ($h<0$).

Finally, we look closer at the importance of optimizing both the cache size C and the time period that the cache is dynamically allocated (t_a to t_d). For these experiments we compare the optimized results presented in Figure 4 with results obtained if using a smaller subset of (average) cache sizes $C = 10, 20, 46, 100, 200, 460, 1000, 2000, 4600, 10000$. Here, we include both simulation-based results and results obtained using our RCW approximations. In particular, Figure 6 shows the optimal values for the above C configurations that are able to achieve an optimized cost that is less than five times greater than the lower bound. Here, the rows in the sub-figure matrix show the ratio of the minimal cost to the lower bound versus b, H_{min}, and α_{high}, respectively, and the columns show the results for four example policies (Baseline $k = 1$, Cache on 1st request, Cache on 2nd request, and Cache on 4th request). The figure clearly shows that the choice of C is important. For example, for each example policy (left-to-right) typically only a small subset of the C choices achieved close to optimal cost for that policy (red line). However, even with only this small subset of possible caches sizes (separated by roughly a factor 2) we are able to achieve close to the best performance with each policy. This is illustrated by the red markers associated with RCW simulated (+), LRU simulated ($\times$), and RCW approximation ($\diamond$) typically being close to the optimal policy version (red line).

We also note that the more cost-effective policies (that achieve closer to the lower bound) are less sensitive to the choice of C compared to Baseline, $k = 1$ (that never deallocates the cache). The instances of this policy with the biggest differences between the most cost-effective observed configuration (red markers) and the optimum configuration are primarily associated with instances where the best selected configuration (from the smaller subset of possible C values) results in a significantly higher hit rate than H_{min}. Of course, for these instances, a reduced cost (down to the optimal for that policy) could have been achieved by reducing the cache size down to the smallest cache size that achieves H_{min}. Even though our model is highly accurate for this policy (Sections 3.4 and 4.4) we also observe some instances where the observed differences between the best points (red markers) from analysis and simulations are substantial. These instances corresponds to cases where for the same C, the analytic value ended up slightly above H_{min} and the simulation values ended up slightly below H_{min} (and therefore not satisfying the optimization constraints), resulting in a different cache size C being used for the simulation points than for the analysis. Of course, with more fine grained cache sizes the best values would be much more similar and closer to the optimal configuration (red line).

For the simulation-based results presented here (similar to the analytic results), we used the simulation results from previous sections to obtain the cache hit rate for both transient and steady-state periods using each cache size C of interest. Then, we used knowledge of the workload to optimize t_a and t_d based on these values. Here, we simply find the largest t_a (with t_d chosen as $T - t_a$, by symmetry in the optimal solution for this workload) for which the expected hit rate is at least H_{min}. This approach can of course be taken for any simulated/operational system for which one can measure both steady-state and transient hit rates



(a) Varying b, Baseline $k = 1$ (b) Varying b, Cache on 1st (c) Varying b, Cache on 2nd (d) Varying b, Cache on 4th

(e) Varying H_{min}, Baseline $k = 1$ (f) Varying H_{min}, Cache on 1st (g) Varying H_{min}, Cache on 2nd (h) Varying H_{min}, Cache on 4th

(i) Varying $_{high}$, Baseline $k = 1$ (j) Varying $_{high}$, Cache on 1st (k) Varying $_{high}$, Cache on 2nd (l) Varying $_{high}$, Cache on 4th

Fig. 6. Demonstrating the effects of optimizing using only sample simulations (or analytic evaluations) based on a smaller set of cache size C rather than optimizing over all possible C, t_a, t_d. Here, we show the optimized values for the example con gurations that achieve a ratio of the minimal cost to the lower bound that is less than ve, when using one of the following (discrete) cache sizes $C = 10; 20; 46; 100; 200; 460; 1000; 2000; 4600; 10000$. For each x-value of interest, results for the best of these con gurations are shown using red markers, while results for the others are shown with black markers. Results from both RCW simulations (+) and LRU simulations ( ) are shown. We also include the best (over these C values) locally optimized delivery cost ratio when using the the RCW approximation ( ) and the global optimal value from Figure 4 (red line) for each policy, which is optimized across all possible C, t_a, t_d. The rows in the sub- gure matrix show the ratio of the minimal cost to the lower bound versus b, H_{min}, and $_{high}$ (other parameters at defaults). The columns show the results for four example policies.

and the number of requests that a transient period typically takes. We therefore argue that several of the optimization formulations that we use for optimized policies in this paper also could be applied when optimizing systems with harder-to-analyze policies as long as the system can be measured during both transient and steady-state for different cache sizes and the overall workload volume can be properly predicted over time.

7 Related Work

Caching policies typically either evict objects from the cache when the cache becomes full (i.e., capacity-driven policies [18]–[23]) or based on the time since each individual object was last accessed or entered the cache (i.e., timing-based policies [22], [23]). In practice, use of capacity-based policies such as LRU have dominated. Unfortunatly, these policies are extremely hard to analyze exactly (e.g., [24], [25]). This prompted the development of approximations [20], [26], [27] and analysis of asymptotic properties [28]–[32]. Most recent work use that the performance of capacity-driven policies often can be well approximated using TTL-based caches [12]–[16], [33]. TTL-based models have also been used to analyze networks of caches [15], [16], [34]–[36], to derive asymptotically optimized solutions [37], for optimized server selection [38], for utility maximization [39], and for on-demand contract design [40].

Few papers (regardless of replacement policy) have modeled discriminatory caching policies such as Cache on k^{th} request. In our context, these policies are motivated by the risk of cache pollution in small dynamically instantiated caches, and more generally by the long tail of one-timers (one-hit wonders) observed in edge networks [9], [41]–[43]. Recent works include trace-based evaluations of Cache on k^{th} request policies [42], [43], simple analytic models for hit and insertion probabilities that (in contrast to us) ignore cache replacement [43], or works that have used TTL-based recurrence expressions to model variations of Cache on k^{th} request [16], [33], [44]–[46]. Of these works, only Carlsson and Eager try to minimize the delivery cost [33]. However, in contrast to us, they assume elastic TTL caches and only consider a single le.

Other related works have adapted the individual TTL values of each object so to achieve some objective [47], [48]. For example, Carra et al. [2] demonstrate how the individual TTL values of each object can be adapted (with constant overhead) so to closely track the optimal cache con gurations. However, these works only consider Cache on 1st request policies.

To simplify analysis, TTL-based approximations of LRU-based caches typically leverages the general ideas of a cache characterization time [12], [13], which in the simplest case corresponds to the (approximate) time that the object stays in the cache if not requested again. This time corresponds closely to our parameter L, with the important difference that the RCW caches use a request count window rather than a time window. This subtle difference makes our approach favorable when modeling xed capacity caches in systems with substantial request rate variations (e.g., according to a diurnal cycle). Furthermore, our RCW approach allows us to derive (i) exact expressions for general Cache on k^{th} request policies, popularity distributions, and transient periods, and

(ii) corresponding O(1) computational cost approximations. Such results, which we need for our optimization models, are not found in the TTL-based modeling literature.

For the case of an in nite Cache on 1st request cache with a nite request stream, Breslau et al. [26] provide exact hit rate expression for general popularity distributions, as well as approximate scaling properties for Zipf distributions. However, we did not nd these scaling relationships (focusing on orders) suf cient for our analysis and therefore developed more precise expressions for Zipf with = 1 and = 0:5.

The (general) idea of dynamically adapting the amount of dedicated resource based on time-varying wokloads is not new [49]–[52]. Within the context of cloud-based caching, some recent works consider how to scale resources based on time varying workloads and diurnal patterns [2]–[4]. For example, in addition to the work by Carra et al. [2] (discussed above), Sundarrajan et al. [3] use discriminatory caching algorithms together with partitioned caches to save energy during off-peak hours, Dan and Carlsson [4] optimize what objects to push to cloud storage based on diurnal demands and a basic cost model, and Cai et al. [5] considered the problem of how to scale the number of cache servers in a hierarchy of LRU caches. Others have considered how CDNs best collaborate with ISPs making available microdatacenters [53], how to build virtual CDNs on-the- y on top of shared third-party infrastructures [6], or have optimized the caching hierarchy of cloud caches based on the assumption that request rates are known and stationary (ignoring time-varying request loads) [54]. None of these works consider the problem of optimized cache instantiation.

8 Conclusions

In this paper we have taken a rst look at dynamic cache instantiation. For this purpose, we have derived new analysis results for what we term "request count window" (RCW) caches, including explicit, exact expressions for cache performance metrics for Cache on k^{th} request RCW caches for general k, and new O(1) computational cost approximations for cache performance metrics for Zipf popularity distributions with =1 and =0:5. These results are of interest in their own right, especially as the performance of RCW caches is shown to closely match the performance of the corresponding Cache on k^{th} request LRU caches. This contribution is particularly important since it is the rst to accurately capture the performance of LRU caches under time-varying workloads. We then applied our analysis results to develop optimization models for dynamic cache instantiation parameters, speci cally the cache size and the duration of the cache instantiation interval, for different cache insertion policies, as well as for a cost lower bound that holds for any caching policy. Our results show that dynamic cache instantiation using a selective cache insertion policy such as Cache on 2^{nd} request may yield substantial bene ts compared to a permanently allocated cache, and is a promising approach for content delivery applications. Finally, we note that the use of the independent reference model (used here) provides conservative estimates of the potential improvements using dynamic instantiation, since in practice short-term temporal locality, non-stationary object popularities, and high rates of addition of new content typically would make yesterday's cache contents less useful.

Acknowledgements

This work was supported by funding from the Swedish Research Council (VR) and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] N. Carlsson and D. Eager, "Optimized dynamic cache instantiation," in Proc. IFIP Networking, June 2020.
[2] D. Carra, G. Neglia, and P. Michiardi, "TTL-based cloud caches," in Proc. IEEE INFOCOM, 2019.
[3] A. Sundarrajan, M. Kasbekar, and R. Sitaraman, "Energy-ef cient disk caching for content delivery," in Proc. ACM e-Energy, 2016.
[4] G. Dan and N. Carlsson, "Dynamic content allocation for cloud-assisted service of periodic workloads," in Proc. IEEE INFOCOM, 2014.
[5] C. X. Cai, G. Liang, and U. C. Kozat, "Load balancing and dynamic scaling of cache storage against Zip an workloads," in Proc. IEEE ICC, 2014.
[6] S. Kuenzer et al., "Unikernels everywhere: The case for elastic CDNs," ACM SIGPLAN Notices, vol. 52, no. 7, pp. 15–29, 2017.
[7] M. McKeay, "The building wave of internet traf c," https://blogs.akamai.com/sitr/2020/04/the-building-wave-of-internet-traf c.html, 2020, [Online; accessed 30-June-2021].
[8] A. Feldman et al., "The lockdown effect: Implications of the covid-19 pandemic on internet traf c," in Proc. ACM IMC, 2020.
[9] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traf c characterization: A view from the edge," in Proc. IMC, 2007.
[10] P. Gill, M. Arlitt, N. Carlsson, A. Mahanti, and C. Williamson, "Characterizing organizational use of web-based services: Methodology, challenges, observations, and insights," ACM Transactions on the Web, vol. 5, no. 4, pp. 19:1–19:23, 2011.
[11] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn," in Proc. ICDCS, 2011.
[12] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," IEEE JSAC, vol. 20, 2002.
[13] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in Proc. ITC, 2012.
[14] G. Bianchi, A. Detti, A. Caponi, and N. B. Melazzi, "Check before storing: What is the performance price of content integrity veri cation in LRU caching?" ACM CCR, vol. 43, no. 3, pp. 59–67, 2013.
[15] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," Performance Evaluation, vol. 79, pp. 2–23, 2014.
[16] M. Garetto, E. Leonardi, and V. Martina, "A uni ed approach to the performance analysis of caching systems," ACM ToMPECS, vol. 1, 2016.
[17] F. Chen et al., "End-user mapping: Next generation request routing for content delivery," in Proc. ACM SIGCOMM, 2015.
[18] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," ACM Computing Surveys, vol. 35, no. 4, pp. 374–398, 2003.
[19] G. Barish and K. Obraczke, "World wide web caching: trends and techniques," IEEE Comm. Mag. vol. 38, no. 5, pp. 178–184, 2000.
[20] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in Proc. ACM SIGMETRICS, 1990.
[21] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating content management techniques for web proxy caches," ACM SIGMETRICS Performance Evaluation Review, vol. 27, no. 4, pp. 3–11, 2000.
[22] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet caches," in Proc. IEEE INFOCOM, 2003.
[23] O. Bahat and A. M. Makowski, "Measuring consistency in TTL-based caches," Performance Evaluation, vol. 62, no. 1, pp. 439–455, 2005.