ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks

Michele Tortelli[1], Dario Rossi[1] and Emilio Leonardi[2]

[1]Télécom ParisTech, Paris, France
[2]Politecnico di Torino, Torino, Italy

Abstract

Large scale deployments of general cache networks, such as Content Delivery Networks or Information Centric Networking architectures, arise new challenges regarding their performance evaluation for network planning. On the one hand, analytical models can hardly represent all the detailed interaction of complex replacement, replication, and routing policies on arbitrary topologies. On the other hand, the sheer size of network and content catalogs makes event-driven simulation techniques inherently non-scalable.

We propose a new technique for the performance evaluation of large scale caching systems that intelligently integrates elements of stochastic analysis within a MonteCarlo simulative approach, that we colloquially refer to as *ModelGraft*. Our approach (i) leverages the intuition that complex scenarios can be mapped to a simpler equivalent scenario that builds upon Time-To-Live (TTL) caches; it (ii) significantly downscales the scenario to lower computation and memory complexity, while, at the same time, preserving its properties to limit accuracy loss; finally, it (iii) is simple to use and robust, as it autonomously converges to a consistent state through a feedback-loop control system, regardless of the initial state.

Performance evaluation shows that, with respect to classic event-driven simulation, ModelGraft gains over *two orders of magnitude* in both CPU time and memory complexity, while limiting accuracy loss below 2%. In addition, we show that ModelGraft extends performance evaluation well beyond the boundaries of classic approaches, by enabling study of Internet scale scenarios with content catalogs comprising *hundreds of billions* objects.

1 Introduction

Caching systems, from their simplest single-cache incarnation to more complex general networks, have attracted remarkable attention over the years. Different approaches have been proposed to study their performance, from exact analytical models [21] with a high computational cost, to refined approximations able to

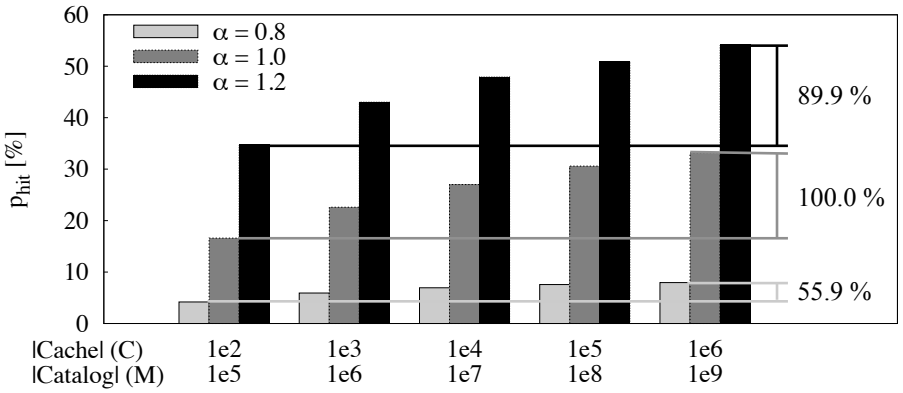


Figure 1: Hit ratio variability - 4-level binary tree, variable α, fixed $C/M = 0.01\%$ ratio, with proportional variation of C and M.

reduce the computational cost while preserving their accuracy within acceptable bounds [11, 19, 12, 22, 36, 10, 13, 35, 29, 26]. However, with Content Distribution Networks (CDNs) first, and then with the advent of a new networking paradigm, namely Information Centric Networking (ICN) [32], caches become the atomic part of globally deployed networks. Under CDN/ICN architectures, several factors like content replacement algorithms, cache decision policies, and forwarding strategies, interact with each other at a global scale: such intricate dependencies heavily influence the performance of the whole system, and thus make it hard to rely only on pure analytical models to reliably predict network Key Performance Indicators (KPIs).

It follows that, especially as a first step to assess the performance of new complex protocols, event-driven simulation techniques represent an appealing alternative: indeed, simulative techniques make it simpler to describe algorithmic interactions between all the different entities with the desired level of fidelity. At the same time, large-scale simulations require massive computational resources (both CPU and memory), to the point that the very same fidelity of simulation results may be compromised by the intrinsic scalability limit of the technique itself.

To show why this may be the case, we provide in Fig. 1 an illustrational example. Whereas it is well known that Internet-scale catalogs approaches trillion objects [30], these scales are hardly achievable in simulation: as such, simulation-based studies typically resort to naïve downscaling of the scenario under investigation, to comply with memory constraints and CPU-time budgets. Using the mean hit ratio as KPI, Fig. 1 contrasts the performance of a four-level binary tree where the Zipf exponent is varied $\alpha \in \{0.8, 1, 1.2\}$, and the ratio between catalog cardinality M and cache size C is kept constant to $C/M = 0.01\%$, while both M and C are jointly downscaled. Results in Fig. 1 clearly show that barely downscaling the simulated scenario by

linearly reducing the size and cardinality of all the components does not preserve its original properties at all: the relative error between the smallest vs largest scenario is between 50% and 100%. This fact has profound implications: indeed, rather typically, crucial parameters of the scenario under investigation such as the Zipf exponent α (or Mandelbrot Zipf plateau q) are measured over real Internet catalogs such as YouTube [8] (or the BitTorrent ecosystem [16]) that are not only growing, but already had a catalog in the order of hundred millions objects about 10 years ago [8]. Clearly, Fig. 1 also implies that merely applying catalog parameters inferred from large-scale measurements to small-scale simulations does not make any sense, as it induces excessive distortion of the KPI to make results of practical relevance.

Inspired by *hybrid models* proposed over the years in different domains (see Sec. 9) we argue that *grafting* components of stochastic modeling into simulative techniques can increase the scalability of the resulting technique by orders of magnitude, without compromising its accuracy at the same time. Specifically, our methodology exploits a synergy between stochastic analysis of Least Recently Used (LRU) caches [10, 26] and MonteCarlo approaches based on Time-To-Live (TTL) caches [19, 12, 27, 29]. In particular, our intuition consists in using the characteristic time T_C of Che's approximation [10] for LRU caches as the TTL parameter in MonteCarlo simulations: as a consequence, the complexity is significantly reduced by simulating TTL-based caches in place of their more complex LRU counterpart, and by subsampling the original catalog in a way that preserves its key statistical properties. Given that T_C in complex scenarios is not known a priory, our system uses a feedback loop to iteratively converge to the correct T_C value, even when the initial guess is wrong by two orders of magnitude.

We develop this intuition further to build a fully integrated system, making the following contributions:

- we propose a novel hybrid methodology for the performance evaluation of cache networks, that reduces CPU time and memory usage by over two orders of magnitude, while limiting accuracy loss to less than 2%;
- we implement the methodology as an alternative simulation engine, so that users can seamlessly switch between event-driven vs ModelGraft simulation, on the very same scenario;
- we make the technique (and scenarios) available as open source in the latest release of ccnSim [1].

In the remainder of this paper, we first provide background information about the building blocks leveraged by our methodology, and about the modeling intuition behind them (Sec. 2). We next provide a succinct overview of our proposal (Sec. 3), followed by an in-depth description of each component (Sec. 4–6). We next validate the technique through a performance evaluation (Sec. 7) involving both very-large

and Internet-scale scenarios, before showcasing results related to an extensive sensitivity analysis (Sec. 8) on all the main parameters of the ModelGraft workflow. In the end, we cover related work (Sec. 9) and conclude the paper (Sec. 10).

2 Modeling intuition

In this section we first introduce background material on Che's approximation [9] (Sec. 2.1), and then formalize our intuition about the equivalence between LRU caches and (opportunely configured) TTL-based caches [19, 12, 27, 29] (Sec. 2.2), which constitute a fundamental building block of our methodology.

2.1 Background

Che's approximation [9], conceived for a LRU cache, is essentially a mean-field approximation which greatly simplifies interactions between different contents inside a cache. In particular, it consists in confusing the *cache eviction time* $T_C(m)$ for content m (i.e., the time since the last request after which object m will be evicted from the cache, unless the object is not requested again in the meantime), with a *constant characteristic time* T_C, which does not depend on the content itself, being a property of the whole cache. As a consequence, content m is considered to be in the cache at time t, if and only if, at least one request for it has arrived in the interval $(t - T_C, t]$. Supposing an Independent Request Model (IRM), with content catalog of size M, and request process for content m to be Poisson of rate λ_m, the probability $p_{in}(m)$ for content m to be in a LRU cache at time t can be expressed as:

$$p_{in}(\lambda_m, T_C) = 1 - e^{-\lambda_m T_C} \tag{1}$$

Denoting with $\mathbb{1}_{\{\mathrm{A}\}}$ the indicator function for event A, we have that by construction, cache size C must satisfy:

$$C = \mathbb{E}[C] = \mathbb{E}\left[\sum_m \mathbb{1}_{\{m \text{ in cache at t}\}}\right] = \sum_m p_{in}(\lambda_m, T_C). \tag{2}$$

It follows that the characteristic time T_C can be computed by numerically inverting (5), which admits a single solution [9]. Finally, KPIs of the system, such as the *cache hit probability* p_{hit}, can be computed using the PASTA property as:

$$p_{hit} = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = \sum_m \lambda_m p_{in}(\lambda_m, T_C) / \sum_m \lambda_m \tag{3}$$

As far as a single cache is concerned, Che's approximation was originally proposed for LRU caches [9], but it has been extended in more recent times to FIFO or Random replacement [13], LRU caches with probabilistic insertion [26], possibly depending on complex cost functions [4], and renewal request models [23].

As far as networks of caches are concerned, however, further approximations are required as an alternative approach to the computationally and algorithmically challenging characterization of the miss streams at any node in the network [26, 29]. In arbitrary networks with shortest path [36], or more complex routing policies [38], it has been shown that inaccuracies can potentially cascade, with significant degradation of the accuracy with respect to simulation [39]. Finally, analytical approaches often assume stationary conditions, thus lacking in characterizing transient periods, although a model has been recently proposed only for a single cache [14]. All these reasons thus justify the quest for an hybrid approach, such as the one we propose in this work.Che's approximation [9], conceived for a LRU cache, is essentially a mean-field approximation which greatly simplifies interactions between different contents inside a cache. In particular, it consists in confusing the *cache eviction time* $T_C(m)$ for content m (i.e., the time since the last request after which object m will be evicted from the cache, unless the object is not requested again in the meantime), with a *constant characteristic time* T_C, which does not depend on the content itself, being a property of the whole cache. As a consequence, content m is considered to be in the cache at time t, if and only if, at least one request for it has arrived in the interval $(t - T_C, t]$. Supposing an Independent Request Model (IRM), with content catalog of size M, and request process for content m to be Poisson of rate λ_m, the probability $p_{in}(m)$ for content m to be in a LRU cache at time t can be expressed as:

$$p_{in}(\lambda_m, T_C) = 1 - e^{-\lambda_m T_C} \tag{4}$$

Denoting with $\mathbb{1}_{\{\mathrm{A}\}}$ the indicator function for event A, we have that by construction, cache size C must satisfy:

$$C = \mathbb{E}[C] = \mathbb{E}\left[\sum_m \mathbb{1}_{\{m \text{ in cache at t}\}}\right] = \sum_m p_{in}(\lambda_m, T_C). \tag{5}$$

It follows that the characteristic time T_C can be computed by numerically inverting (5), which admits a single solution [9]. Finally, KPIs of the system, such as the *cache hit probability* p_{hit}, can be computed using the PASTA property as:

$$p_{hit} = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = \sum_m \lambda_m p_{in}(\lambda_m, T_C) / \sum_m \lambda_m \tag{6}$$

As far as a single cache is concerned, Che's approximation was originally proposed for LRU caches [9], but it has been extended in more recent times to FIFO or Random replacement [13], LRU caches with probabilistic insertion [26], possibly depending on complex cost functions [4], and renewal request models [23].

As far as networks of caches are concerned, however, further approximations are required as an alternative approach to the computationally and algorithmically challenging characterization of the miss streams at any node in the network [26, 29]. In arbitrary networks with shortest path [36], or more complex routing policies [38], it

has been shown that inaccuracies can potentially cascade, with significant degradation of the accuracy with respect to simulation [39]. Finally, analytical approaches often assume stationary conditions, thus lacking in characterizing transient periods, although a model has been recently proposed only for a single cache [14]. All these reasons thus justify the quest for an hybrid approach, such as the one we propose in this work.

2.2 Intuition

Observe that, given the *characteristic time* T_C under Che's approximation, the dynamics of the different contents become completely decoupled. As a consequence, we can resort to simpler caching than LRU to ease the analysis of complex and large cache networks. One such alternative is constituted by Time-to-Live (TTL) based caches [19, 12, 27, 29]: contents are evicted from the cache after a pre-configured *eviction time* T'_C since the last request, unless a cache hit happens in the meantime (which resets the object timer).

Observation 1. *As experimentally shown in [9], and remarked in [12, 26], the dynamics of a LRU cache with characteristic time T_C, fed by an IRM process associated to a catalog $\mathcal{M}$ with cardinality M, and request rates λ_m drawn from a distribution Λ, become indistinguishable from those of a TTL based cache with deterministic TTL parameter set equal to T_C, and operating on the same catalog*:

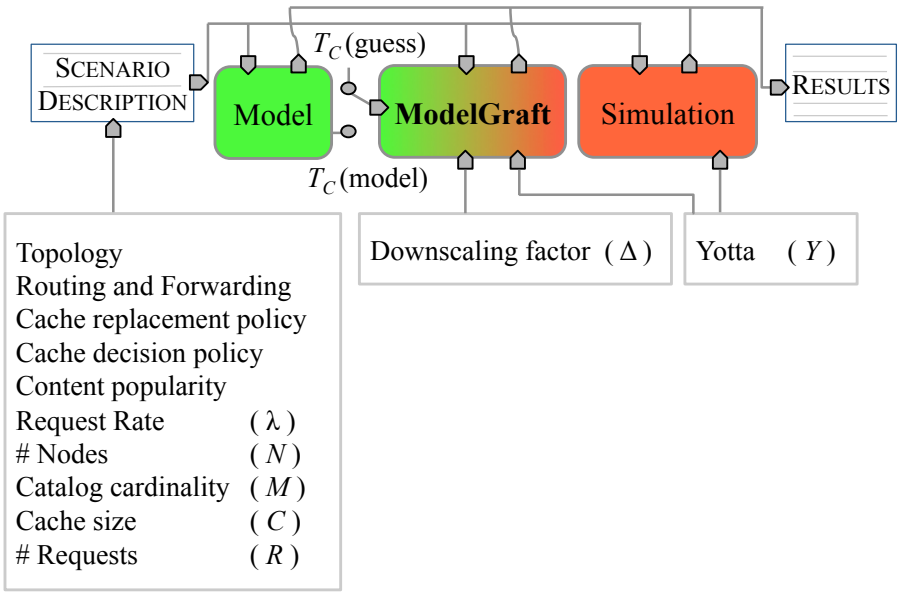
$$p_{hit}^{TTL}(T_C) = \mathbb{E}_\Lambda[1 - e^{-\lambda_m T_C}] = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = p_{hit}^{LRU}(T_C) \tag{7}$$

Specifically, (7) equals the average hit probability of the original LRU system to that of its TTL-based equivalent [12, 26]. Leveraging further on this intuition, we argue:

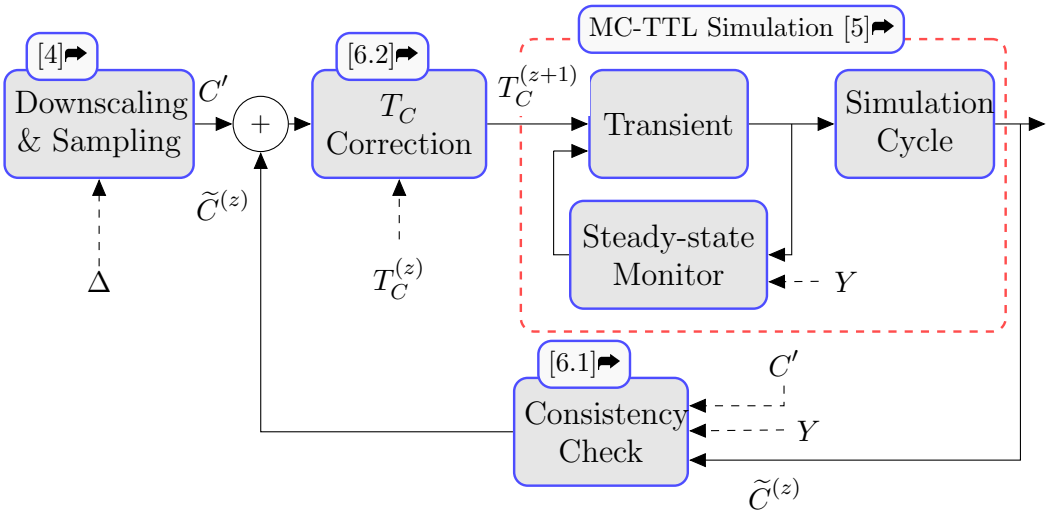
Observation 2. *Large-scale LRU networks can be analyzed through a downscaled system associated to a catalog $\mathcal{M}'$ with cardinality $M' \ll M$, where each cache is replaced by its TTL equivalent with an eviction time T'_C set equal to the characteristic time T_C of the original LRU cache. KPIs of the original network are, by construction, recovered by averaging system performance over multiple MonteCarlo realizations of the downscaled one, each lasting for a duration δT, and where rates λ'_m for individual objects of the downscaled catalog $\mathcal{M}'$ are uniformly and independently drawn from Λ. Thus, expanding (7):*

$$\begin{aligned} \mathbb{E}_{\Lambda'}\left[p_{in}(\lambda'_m, T_C)\right] &= \frac{\mathbb{E}_\Lambda\left[\lambda'_m p_{in}(\lambda'_m, T_C)\right]}{\mathbb{E}_\Lambda[\lambda'_m]} \\ \frac{\sum_m \lambda_m P_t(\lambda'_m = \lambda_m) p_{in}(\lambda_m, T_C)}{\sum_m \lambda_m P_t(\lambda'_m = \lambda_m)} &= \frac{\sum_m \lambda_m p_{in}(\lambda_m, T_C)}{\sum_m \lambda_m} \end{aligned} \tag{8}$$

where $P_t(\lambda'_m = \lambda_m)$ represents the probability that we have $\lambda'_m = \lambda_m$ at a generic time instant t. Observe that the top-right expression admits a simple physical



(a) High level view of the integrated simulation workflow



(b) Details of the ModelGraft MonteCarlo TTL-Based workflow

Figure 2: ModelGraft overview: (a) integration in ccnSim and (b) synoptic of the ModelGraft workflow.

interpretation as the ratio between the average hit-rate and the average rate of requests at the cache.

Given that, under the previous assumptions, contents have decoupled dynamics, we can significantly *downscale the system*, thus reducing both memory and CPU

complexity on the one hand, and *accurately represent* complex interactions and correlations among different caches, at the same time.

Observation 3. *A practical approach is to let $\delta T \to 0$, and a convenient approximation is to re-extract λ'_m at every new request, so to satisfy (8).*

We would like to remark that, in this case, in order to ensure that at a given arbitrary time t, $P_t(\lambda'_m = \lambda_m) = 1/\mathcal{M}$, we have to extract the new value for λ'_m at every arriving request from Λ non uniformly. In particular, it can easily shown, as consequence of classical renewal arguments, that the probability of extracting λ_m as a new value for λ'_m must be set equal to $\frac{\lambda_m}{\sum_m \lambda_m}$, i.e., more formally, $P(\lambda'_m = \lambda_m) = \frac{\lambda_m}{\sum_m \lambda_m}$.

The remainder of this paper illustrates, describes, and validates in greater details the methodology that is built upon these observations.

3 ModelGraft overview

ModelGraft performs MonteCarlo simulations of an opportunely downscaled system, where LRU caches are replaced by their Che's approximated version, implemented in practice as TTL-based caches. Before dwelving into its details, it is worth to both placing it into a broader context, as well as illustrating, at high level, each of its building blocks.

We implement ModelGraft as a simulation engine available in the latest version of ccnSim [1]. As illustrated in Fig.2(a), starting from a unique scenario description, users can analyze the performance of cache networks via either an *analytical model* [26], if available (left), a classic *event-driven* simulation engine (right), or via the *ModelGraft* engine (middle).

ModelGraft depends on a single additional parameter, namely the downscaling factor Δ, which can be easily tuned according to guidelines in Sec.6.2. As introduced in Sec. 2.2, ModelGraft requires input T_C values for each cache in the network. One option could be to bootstrap ModelGraft with *informed guesses* of T_C gathered via, e.g., analytical models (notice the $T_C(model)$ switch in Fig.2(a)), which would however limit the interest of the methodology (i.e., only cases covered by the model could be considered). A more interesting approach, used by default in ModelGraft, is instead to start from *uninformed guesses* of T_C (notice the default wiring to the $T_C(guess)$ switch in Fig.2(a)), and let the system iteratively correct the T_C value. In other words, ModelGraft is intrinsically conceived as an *auto-regulating* system, so that, by design, it achieves accurate results even when the input T_C values, that the user does not even need to be aware of, largely differ from the correct ones.

Details are exposed in Fig.2(b), which shows each of the blocks that are thoroughly described in the following sections. In a nutshell, ModelGraft starts with the configuration of the *downscaling and sampling* process (Sec. 4), before entering

the *MonteCarlo TTL-based (MC-TTL) simulation* (Sec. 5). During the MC-TTL phase, statistics are computed after a transient period (Sec. 5.1), where an *adaptive steady-state monitor* tracks and follows the dynamics of the simulated network in order to ensure that a steady-state regime is reached without imposing a fixed threshold (e.g., number of requests, simulation time, etc.) a priori (Sec. 5.2). Once at steady-state, a downscaled number of requests are simulated within a *MC-TTL cycle* (Sec. 5.3), at the end of which the monitored metrics are provided as input to the *self-stabilization* block (Sec. 6): a *consistency check* decides whether to end the simulation (Sec. 6.1), or to go through a T_C *correction* phase (Sec. 6.2) and start a new simulation cycle.

4 Downscaling and sampling

4.1 Design

The ModelGraft workflow starts with a proper *downscaling* of the original scenario. The only controlling parameter is the *downscaling factor* $\Delta \gg 1$: a catalog comprising $M' = M/\Delta$ objects, attracting $R' = R/\Delta$ total requests, will be simulated at steady-state in the equivalent TTL-based system (Sec. 5.3). Moreover, when $T'_C = T_C$, a TTL cache will store, on average, $C' = C/\Delta$ contents at steady-state. Indeed, adapting (5) to the downscaled scenario (i.e., $M' = M/\Delta$), we can compute the expected cache size as:

$$\mathbb{E}[C'] = \sum_{n=1}^{M'} p_{in}(\lambda'_n, T_C) = C/\Delta. \tag{9}$$

However, in order to avoid pitfalls caused by a naïve downscaling process (recall Fig. 1), we need to ensure that the downscaled catalog preserves the main features of the original one, e.g., its popularity distribution. While our methodology is not restricted to a specific popularity law, in what follows we develop the case where object popularity follows a Zipfian probability distribution with exponent α – which is also the most interesting case from a practical viewpoint. Hence, we denote with Λ the aggregate arrival rate of all objects in the catalog, and with $\lambda_n = \Lambda n^{-\alpha} / \sum_{k=1}^{M} k^{-\alpha}$ the rate for the n-th object in the original catalog.

The proposed approach, sketched in Fig. 3, consists in splitting the original catalog into a number of M' bins having the same cardinality Δ, i.e., $|\mathcal{M}_n| = \Delta$, where $\mathcal{M}_n$ refers to the n-th bin, with $n \in [1, M']$. In ModelGraft, each bin of the original catalog is represented by a single "meta-content" in the downscaled system, i.e., the active catalog comprises M' meta-contents. The key idea is to let each meta-content n to be requested with an *average request rate*, $\bar{\lambda}'_n$, which closely approximates the average request rate of the contents within the respective bin in the original catalog. More formally, for the n-th meta-content, with $n \in [1, M']$, it

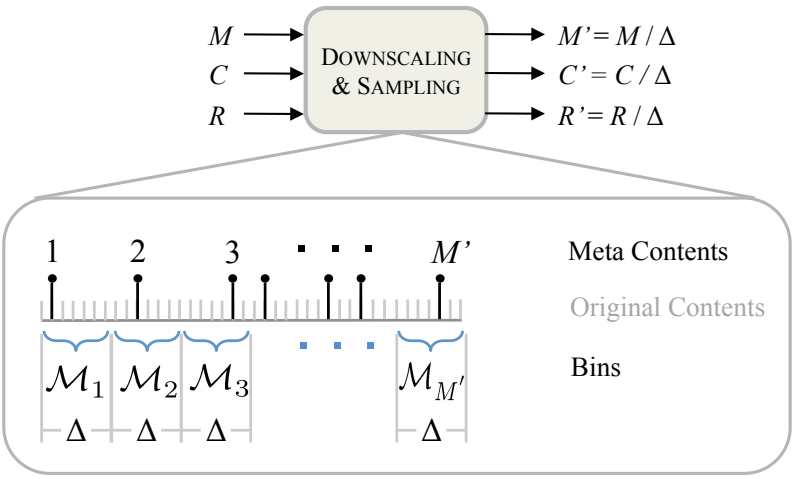


Figure 3: Downscaling and sampling process.

is required that:

$$\bar{\lambda'}_n = \frac{1}{\Delta} \sum_{m=(n-1)\Delta+1}^{n\Delta} \lambda_m, \tag{10}$$

where the interval $[(n-1)\Delta+1, n\Delta]$ represents contents of the original catalog that fall within the n-th bin. This design can be achieved by (i) instantiating M' parallel request generators, i.e., one per each meta-content, each of which is identified by a fixed $n \in [1, M']$; (ii) for any given meta-content n, varying its instantaneous request rate at each new request, as suggested in Observation (3), so that its average complies with (10).

4.2 Implementation

As already anticipated in section 2.2, it is easy to see that the simplest implementation of the above requirements boils down to bind the probability according to which, rate λ'_n for n-th meta-content is sampled at every new request i.e., $P(\lambda'_n = \lambda_m)$ with the popularity distribution of the Δ contents inside the respective bin $m \in [(n-1)\Delta+1, n\Delta]$:

$$P(\lambda'_n = \lambda_m) = \frac{\lambda_m}{\sum_{j=(n-1)\Delta+1}^{n\Delta} \lambda_j} \approx \frac{1}{\Delta} \frac{\lambda_m}{\bar{\lambda'}_n}. \tag{11}$$

While the above requirement (10) is met, a significant downside of this naïve implementation is its space complexity. Indeed, since it is based on the classic inverse transform sampling, this approach would require to store M' Cumulative Distribution Functions (CDFs) having each a size Δ, resulting in an overall memory

allocation equal to $M'\Delta = M$ elements, i.e., as in the original scenario. Given that M is the dominant factor driving the overall memory occupancy, it is clear that such a simple implementation is not compatible with our goals.

We therefore resort to a better sampling technique called Rejection Inversion Sampling [18], which is an acceptance-rejection method that efficiently generates random variables from a monotone discrete distribution (in this case Zipf's distribution) without allocating memory-expensive CDFs, and which is characterized by a $\mathcal{O}(1)$ runtime complexity. Originally proposed in [18] for $\alpha > 1$, this technique has been recently extended to all non-negative exponents $\alpha > 0$ [3]. Without thoroughly discussing all its details, we provide a brief overview of its main steps. First of all, the object population (that we assume being the interval $[1, \Delta]$ for sake of simplicity) is divided into two parts: the *head*, composed by only the first element, and the *tail*, consisting in the remaining objects. Then, as in any acceptance-rejection method, a *hat* function $h(x)$ and its *integral* $H(x)$ are defined. At this point, the algorithm [3] iterates through the following steps:

1. Extract a uniformly distributed random variable U from the area under the hat function.

2. Extract, by inversion, the element X to test, and limit its range to $[1, \Delta]$:

$$X \leftarrow H^{-1}(U), \quad K \leftarrow \lfloor X + 1/2 \rfloor \tag{12}$$

3. Return K if:

$$K - X \leq s \;\; \| \;\; U \geq H(K + 1/2) - h(k) \tag{13}$$

It can be shown [3] that the probability of selection and acceptance for both the head and an element from the tail are proportional to the probability mass function of Zipf's distribution.

Recall now that power-law distributions (and hence Zipf's one) exhibit a *scale-independent* (or *self-similar*) property, according to which the scale exponent α is preserved independently of the level of observation. Hence, by means of *rejection inversion sampling*, we can consider a single interval [1,Δ] (i.e., with the same cardinality of one bin) from which extracting indexes that follow a Zipf's distribution with exponent α, and that will be used to vary the request rates of the M' meta-contents at each new request, as pointed out in Sec. 4.1. Indeed:

Observation 4. *If the request generator associated to the n-th bin, with $n \in [1, M']$, needs to schedule the next request rate for the n-th meta-content, an index $t \in [1, \Delta]$ is extracted with the aforementioned technique: to satisfy condition (10), the relative request rate has to be computed as $\lambda'_n = \lambda_{(n-1)\Delta+t}$.*

Proof. In classic *event-driven simulation*, the average request rates for contents of the original catalog $\mathcal{M}$ inside an ideal bin n of size Δ would be:

$$\begin{aligned}\mathbb{E}[\lambda_n] &= \frac{1}{\Delta}\sum_{i=(n-1)\Delta+1}^{n\Delta}\lambda_i = \frac{\Lambda}{\Delta}\sum_{i=(n-1)\Delta+1}^{n\Delta}P(i) \qquad (14)\\ &= \frac{\Lambda}{\Delta}\sum_{i=(n-1)\Delta+1}^{n\Delta}\left(\frac{\frac{1}{i^\alpha}}{\sum_{j=1}^{M}\frac{1}{j^\alpha}}\right)\\ &= \frac{L}{\Delta}\sum_{i=(n-1)\Delta+1}^{n\Delta}\frac{1}{i^\alpha}. \qquad (\text{with } L=\tfrac{\Lambda}{\sum_{j=1}^{M}\frac{1}{i^\alpha}})\end{aligned}$$

Consider, now, the design of ModelGraft: a single random number generator which extracts, at each request, and for each n-th meta-content (where $n \in [1, M']$), a random number from a Zipf's distribution with exponent α and cardinality Δ, i.e., equal to the cardinality of a single bin. It follows that, the probability for the n-th meta-content to be requested with a rate λ_m, where $m \in [(n-1)\Delta+1; n\Delta]$, is given by the joint probability of two events: $P(\mathcal{M}_n)$, which is the probability of scheduling a request for the n-th bin (given by the sum of the $P(\lambda_i)$ of that bin, i.e., the aggregate request rate it represents), and $P(t)$, which represents the probability of extracting an index t from a Zipf's distribution with exponent α and cardinality Δ. The final request rate for the n-th meta-content will be, then, $\lambda_m = \lambda_{(n-1)\Delta+t}$, provided that a request for the n-th bin is being scheduled. As a consequence, it results that:

$$\begin{aligned}\bar{\lambda'}_n &= \frac{\Lambda}{\Delta}\sum_{i=1}^{\Delta}P(i\cap\mathcal{M}_n) = \frac{\Lambda}{\Delta}\sum_{i=1}^{\Delta}P(i|\mathcal{M}_n)P(\mathcal{M}_n)\\ &= \frac{\Lambda}{\Delta}\sum_{i=1}^{\Delta}P(i)P(\mathcal{M}_n)\\ &= \frac{\Lambda}{\Delta}\sum_{i=1}^{\Delta}\left[\frac{\frac{1}{i^\alpha}}{\sum_{j=1}^{\Delta}\frac{1}{j^\alpha}}\sum_{k=(n-1)\Delta+1}^{n\Delta}\left(\frac{\frac{1}{k^\alpha}}{\sum_{z=1}^{M}\frac{1}{z^\alpha}}\right)\right]\\ &= \frac{\Lambda}{\Delta}\sum_{k=(n-1)\Delta+1}^{n\Delta}\left(\frac{\frac{1}{k^\alpha}}{\sum_{z=1}^{M}\frac{1}{z^\alpha}}\right)\sum_{i=1}^{\Delta}\left[\frac{\frac{1}{i^\alpha}}{\sum_{j=1}^{\Delta}\frac{1}{j^\alpha}}\right]\\ &= \frac{L}{\Delta}\sum_{k=(n-1)\Delta+1}^{n\Delta}\frac{1}{k^\alpha} \qquad (\text{with } L=\tfrac{\Lambda}{\sum_{z=1}^{M}\frac{1}{z^\alpha}})\\ &= \mathbb{E}[\lambda_n],\end{aligned}$$

where the first step follows from Bayes' theorem, the second step from independence between the two events, and the latest steps from algebra. ■

5 MC-TTL Simulation

5.1 Transient

Once the scenario is properly downscaled, ModelGraft starts the warm-up phase of the first MC-TTL *simulation cycle* (with initial uninformed guesses for T_C). Given that the duration of the warm-up can be affected by many parameters, ModelGraft monitors KPIs in order to automatically adapt the duration of the transient period, thus guaranteeing their statistical relevance. For instance, transient duration can be affected by forwarding policies (e.g., where shorter average paths under ideal Nearest Replica Routing can reduce the transient with respect to shortest path [38]), as well as from cache decision policy (e.g., like Leave Copy Probabilistically (LCP) [5], where the reduced content acceptance ratio with respect to Leave Copy Everywhere (LCE) is expected to yield longer transient durations).

5.2 Steady-state monitor

TThe convergence of a single node i is effectively monitored using the Coefficient of Variation (CV) of the *measured hit ratio*, $\bar{p}_{hit}(i)$, computed via a *batch means* approach. In particular, denoting with $p_{hit}(j,i)$ the j-th sample of the measured hit ratio of node i, node i is considered to enter a steady-state regime when:

$$CV_i = \frac{\sqrt{\frac{1}{W-1}\sum_{j=1}^{W}\left(p_{hit}(j,i) - \bar{p}_{hit}(i)\right)^2}}{\frac{1}{W}\sum_{j=1}^{W}\bar{p}_{hit}(j,i)} \leq \varepsilon_{CV}, \tag{15}$$

where W is the size of the sample window, and ε_{CV} is a user-defined convergence threshold. To avoid biases, new samples are collected only if (i) the cache has received a non-null number of requests since the last sample, and (ii) its state has changed, i.e., at least a new content had been admitted in the cache since the last sample. To exemplify why this is important, consider that with a LCP(p) cache decision policy, where new contents are probabilistically admitted in the cache, the reception of a request is correlated with the subsequent caching of the fetched content only in 1 out of $1/p$ cases.

At network level, denoting with N the total number of nodes in the network, and given a tunable parameter $Y \in (0,1]$, we consider the whole system to enter steady-state when:

$$CV_i \leq \varepsilon_{CV}, \qquad \forall i \in \mathcal{Y}, \tag{16}$$

where $|\mathcal{Y}| = \lceil YN \rceil$ is the *set of the first YN nodes satisfying condition* (15). The rationale behind this choice is to avoid to unnecessarily slow down the convergence of the whole network by requiring condition (15) to be satisfied by all nodes: indeed,

due to particular routing protocols and/or topologies, there are nodes that have low traffic loads (hence, long convergence time), and, at the same time, a marginal weight in network KPIs. A sensitivity analysis of the impact that Y has on system performance is presented in Sec. 8.3, with the aim to show that, even by excluding some nodes, the accuracy of the gathered KPIs is not affected, and, at the same time, the time of convergence is reduced.

5.3 Simulation cycle

For the original system, the *duration of a simulation cycle* T at steady-state is computed as $T = R/(\Lambda N_C)$, where R is the target number of requests, $\Lambda = \sum_{i \in M} \lambda_i$ is the aggregate request rate per client, and N_C the number of clients. In MC-TTL simulations, instead, the total request rate per each client is $\Lambda' = \sum_{n \in M'} \lambda'_n \approx \Lambda/\Delta$. Keeping the simulated time $T' = T$ constant, it follows that the number of simulated events per each cycle of a MC-TTL simulation is $R' = R/\Delta$ – with an expected significant reduction of the CPU time required to simulate a cycle.

6 Self-stabilization

As described in Sec. 3, one of the desirable properties of our hybrid methodology is a *self-contained* design that allows to simulate large scale networks even in the absence of reliable estimates of characteristic times T_C. This is achieved through a feedback loop, which ensures that our methodology *self-stabilizes* as a result of the combined action of two elements: a measurement step, referred as *consistency check*, and a *controller action*, where inaccurate T_C values are corrected at each iteration.

6.1 Consistency check

The *consistency check* is based on Eq. (9), according to which a TTL cache of the downscaled system, with downscaling factor Δ, and $T'_C = T_C$, would store, on average, $C' = C/\Delta$ contents at steady-state. Moreover, it is worth highlighting that, unlike LRU caches having a fixed size (and where the oldest content is selected for eviction), TTL caches are supposed with *infinite size* [12] (as old contents remain soft-state in the cache for a fixed TTL, but are not otherwise evicted due to cache size limits). Considering that there exists a strong correlation between the eviction time T_C and the number of cached contents, it follows that we can consider the *measured cache size* $\widetilde{C}$ as our controlled variable.

In particular, for each TTL cache we maintain an online average of the number of stored contents as:

$$\widetilde{C}_i^{(z)}(k+1) = \frac{\widetilde{C}_i^{(z)}(k)\ t(k) + {B_i}^{(z)}(k+1)\ [t(k+1) - t(k)]}{t(k+1)}, \tag{17}$$

where $\widetilde{C}_i^{(z)}(k)$ is the online average of the cache size of the i-th node at k-th measurement time during z-th simulation cycle, and ${B_i}^{(z)}(k+1)$ is the actual number of contents stored inside the TTL cache of the i-th node at the $(k+1)$-th measurement time during the z-th simulation cycle. Samples for the online average are clocked with *miss* events and collected with a probability 1/10, so that samples are geometrically spaced, in order to avoid oddities linked to periodic sampling [6].

At the end of each MC-TTL simulation cycle (i.e., after the simulation of R' requests), the *consistency check* block evaluates the accuracy of the measured cache size $\widetilde{C}^{(z)}$, with respect to the target cache C', by using the following expression:

$$\frac{1}{YN}\sum_{i\in\mathcal{Y}}\frac{\left|C'-\widetilde{C}_i^{(z)}(k_{end})\right|}{C'}\leq\varepsilon_C, \tag{18}$$

where $\widetilde{C}_i^{(z)}(k_{end})$ is the online average of the measured cache size of i-th node at the end of the z-th simulation cycle, C' is the target cache, supposed to be equal for all the nodes without loss of generality, and ε_C is a user-defined consistency threshold. For coherence, measures are taken on those $|\mathcal{Y}|=YN$ nodes that have been marked as stable in Sec. 5.2. If condition (18) is satisfied, the MC-TTL simulation ends, otherwise a new MC-TTL cycle needs to be started: T_C values are corrected (as in the next subsection), all the caches are flushed, and the online average measurements are reset.

6.2 T_C correction

The controller action leverages the direct correlation that exists between the *target cache size* $C'=C/\Delta$ expected for a TTL cache at steady-state, and its characteristic time $T_C'=T_C$, that is expressed through equations (4)-(5)-(7)-(18). Intuitively, there exists a direct proportionality between C' and T_C: i.e., the average number of elements $\widetilde{C}$ stored in a TTL cache, with TTL=T_C, grows as T_C grows.

Therefore, if the consistency check block reveals that the measured cache size $\widetilde{C}$ of a particular node is *smaller* than its target cache $\widetilde{C}<C'$, it means that the respective T_C value provided as input is actually *smaller* than the actual T_C, thus suggesting an increment in the next step. Viceversa, for a $\widetilde{C}>C'$, the T_C of the correspondent node should be decreased.

As a consequence of this relationship, we employ a *proportional* (P) controller to compensate for T_C inaccuracies. That is, if condition (18) is not satisfied at the end of z-th simulation cycle, T_C values are corrected, before starting the next cycle, as:

$${T_{Ci}}^{(z+1)}={T_{Ci}}^{(z)}\left(\frac{C'}{\widetilde{C}_i^{(z)}}\right), \tag{19}$$

where ${T_{Ci}}^{(z)}$ is the TTL value assigned to the i-th node during the z-th simulation cycle. In practice, (19) guarantees a fast convergence towards the right T_C values

(see Sec. 8.1), avoiding at the same time any divergence of the control action (provided that measures on $\widetilde{C}$ are taken at steady-state). This allows ModelGraft to guarantee considerable gains, even when multiple simulation cycles are necessary due to significantly inaccurate input T_C values.

There is an important condition worth highlighting: i.e., the controller needs to react on *reliably measurable* quantities, as opposite to *noisy measures* – which happens whenever the terms of the ratio are very small $C' \approx \widetilde{C}_i^{(z)} \approx 1$, as errors in their estimation amplify in their ratio $C'/\widetilde{C}_i^{(z)}$. In particular, this translates into a very simple practical guideline: i.e., it introduces a lower bound to the target cache size of the downscaled system $C' = C/\Delta \geq 10$, which practically upper bounds the maximum downscaling factor to $\Delta \leq C/10$ (see Sec. 8.2 for further details).

7 Results

We now validate the ModelGraft engine against classic event-driven simulation in very-large scale scenarios (Sec. 7.1), and we then project ModelGraft gains to Internet-scale one, which are prohibitively complex for classic event-driven simulations (Sec. 7.2). To simplify the analysis of ModelGraft gains, in this section we limitedly provide *accurate* T_C values extracted from event-driven simulations[1]. We, instead, defer the evaluation of ModelGraft self-stabilization capabilities to Sec. 8, where T_C guesses, which are possibly inaccurate by several orders of magnitude, are provided as input. All the results presented in this section have been obtained by executing both event-driven and ModelGraft simulations on the same commodity hardware, i.e., an Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM.

7.1 ModelGraft validation: Very-large Scale Scenario

To evaluate the accuracy of ModelGraft, we consider the largest scenario we can investigate via event-driven simulation gathered via ccnSim – which was already shown to be among the most scalable ICN software tools [39]. To stretch the boundaries reachable by event-driven simulation even further, we integrate the rejection inversion sampling in ccnSim – so that, to the best of our knowledge, *we are the first to evaluate the performance of ICN networks when the content catalog is in the order of billions.*

The validation scenario represents an ICN-access tree network [28], where the topology is a N=15-nodes 4-level binary tree depicted in Fig. 4(a). A single repository, connected to the root node, stores a $M = 10^9$ objects catalog, where objects follow a Zipf popularity distribution with exponent $\alpha = 1$. An overall $R = 10^9$ requests are injected at each leaf nodes, at a rate of $\Lambda = 20\,\text{req/s}$ per leaf.

[1] Reported for completeness in Tab. 4 and discussed in Sec. 8.1

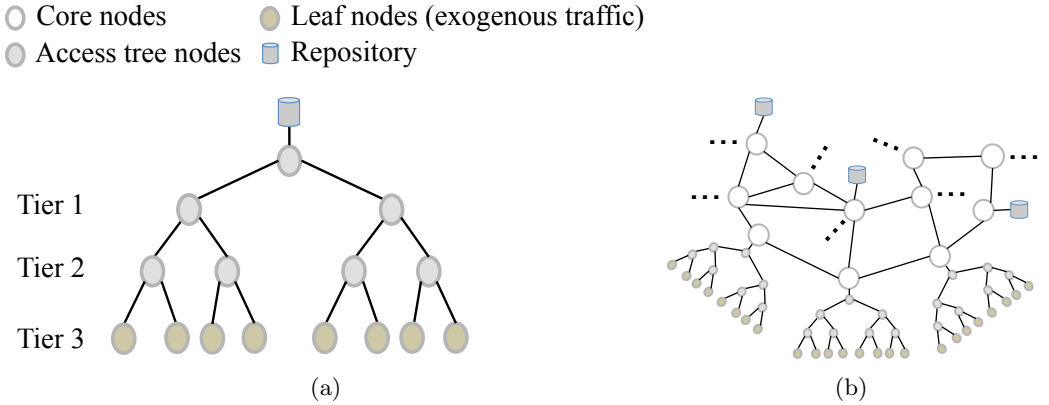


Figure 4: Network Topologies: (a) 4-level binary tree, (b) CDN-like.

The cache size of each node is fixed at $C = 10^6$, resulting in a cache to catalog ratio of $C/M = 0.01\%$. Three different cache decision policies are considered for the comparison: (i) LCE, where fetched contents are always cached in every traversed node; (ii) LCP(1/10), that probabilistically [5] admits contents in the cache (configured so that one over ten fetched contents is cached on average); (iii) 2-LRU [20, 26], where cache pollution is reduced by using an additional cache in front of the main one, with the purpose of caching only the names of requested contents: the fetched contents will be stored in the main cache only in case of a hit event in the first cache. According to our rule of thumb $C' = C/\Delta \geq 10$ (anticipated in Sec. 6.2 and substantiated in Sec. 8.2), the maximum downscaling factor is $\Delta = 10^5$. Additionally, equations (16) and (18) are computed considering $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$ (see Sec. 8 for a sensitivity analysis on ε_{CV} and ε_C): in other words, we test convergence of 75% of the caches in the network, by requiring the coefficient of variation of the hit rate to be below $5 \cdot 10^{-3}$, and we iterate ModelGraft simulations until the measured average cache size of those nodes is within 10% of the expected size $C' = C/\Delta$. A sensitivity analysis to the above parameters is provided in Sec. 8.

Tab. 1 reports mean values (computed over 10 different runs) for three KPIs: mean hit ratio p_{hit}, CPU *time*, and *memory* occupancy. Relative *gains* are also highlighted for CPU and memory footprint (i.e., $KPI^{Simulation}/KPI^{ModelGraft}$), as well as the *accuracy loss* with respect to simulation (i.e., $|p_{hit}^{Simulation} - p_{hit}^{ModelGraft}|$). From the table, sizable gains emerge, which are due to the combination of several factors. First of all, (i) ModelGraft brings significant improvements in terms of reduction of both CPU time and memory occupancy; indeed, the relative gains with respect to the classic event-driven approach are always about two orders of magnitude large, regardless of the cache decision policy. Considering the LCE case as an example, ModelGraft requires only 38 MB of memory and 211 sec of CPU time, compared to 6.4 GB and 11 hours under classic simulation. Additionally, we can

Table 1: ModelGraft validation, accurate initial T_C
(4-level binary tree, $M = R = 10^9, C = 10^6, \Delta = 10^5, Y = 0.75$)

Cache Decision Policy	Technique	p_{hit}	Loss	CPU time	Gain	Mem [MB]	Gain
LCE	Simulation	33.2%	**1.8%**	11.4 h	**194x**	6371	**168x**
	ModelGraft	**31.4%**		**211 s**		**38**	
LCP(1/10)	Simulation	35.4%	**1.4%**	7.3 h	**90x**	6404	**168x**
	ModelGraft	**34.0%**		**291 s**		**38**	
2-LRU	Simulation	37.0%	**0.9%**	10.8 h	**97x**	8894	**234x**
	ModelGraft	**36.1%**		**402 s**		**38**	

also highlight the fact that, despite the aforementioned gains, (ii) the discrepancy between p_{hit} measured by ModelGraft vs the one gathered through the event-driven approach remains always under 2%. Finally, it is interesting to point out the absence of either an (iii) accuracy vs. speed trade-off, as one would typically expect [31], or of a (iv) memory vs. CPU trade-off [17], i.e., cases where an algorithm either trades increased space (e.g., cached results) for decreased execution time (i.e., avoid computation), or viceversa. ModelGraft thus stands in a rare win-win operational point where both CPU and memory usage are significantly relieved, at a price of a furthermore very small accuracy loss.

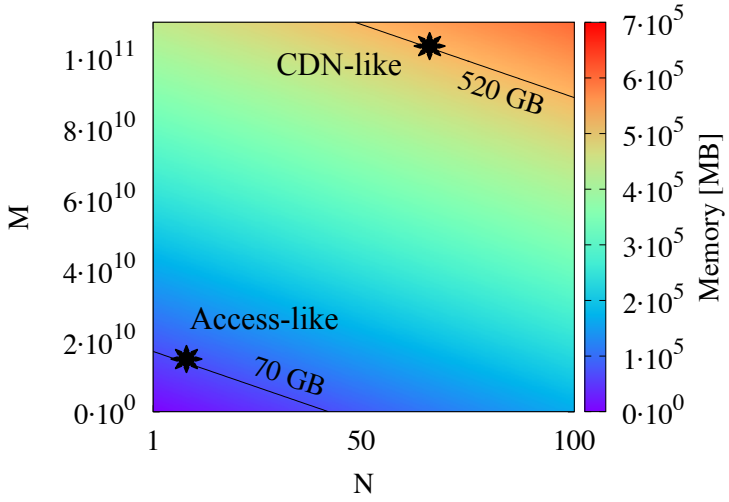


Figure 5: Estimated memory occupancy for event-driven simulation (Mem_{ED}) based on predictive model in (20).

Table 2: Internet-scale scenarios: ModelGraft results and projected gains vs event-driven simulation.

Topology	Parameters	Technique	p_{hit}	CPU time (#Cycles)	Gain	Mem	Gain
Access-like (*N* = 15)	M = 10^{10} R = 10^{10} C = 10^6 Δ = 10^5 Y = 0.75	Simulation (estimate)	n.a.	4.5 days	**270x**	70 GB	**~1500x**
		ModelGraft	**31.4%**	**24 min (1 cycle)**		**45 MB**	
CDN-like (*N* = 67)	M = 10^{11} R = 10^{11} C = 10^7 Δ = 10^6 Y = 0.75	Simulation (estimate)	n.a.	50 days	**96x**	520 GB	**~16700x**
		ModelGraft	**34.0%**	**12.5 h (3 cycles)**		**31 MB**	

7.2 Gain projection: Internet-scale Scenario

We finally employ the validated ModelGraft engine to venture scenarios that are prohibitively complex for classic event-driven simulation, due to both CPU and memory limitations. Indeed, our aim is to investigate Internet-scale scenarios, whose content catalogs are estimated to be in the order of $\mathcal{O}(10^{11})$ – $\mathcal{O}(10^{12})$ [30], i.e., *two orders of magnitude larger than those considered in the previous section.*

We consider two characteristic examples: an Access-like scenario, depicted in Fig. 4(a), represented by a 4-level binary tree with a single repository connected to the root, serving a catalog with cardinality $M = 10^{10}$. Cache size is $C = 10^6$, which limits the maximum downscaling to $\Delta = 10^5$ (see Sec. 8.2). The second scenario, depicted in Fig. 4(b), models, instead, a more complex CDN network, where three repositories, serving a catalog with cardinality $M = 10^{11}$, are connected to backbone nodes which are interconnected as the classic Abilene network, and where an access tree is further attached to each backbone node. In this scenario, we let the cache size be $C = 10^7$, which allows to increase the downscaling factor to $\Delta = 10^6$. In both cases, considering the lack of ground truth, we provided T_C guesses as input.

As before, we set $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$, and we run experiments on the same Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM. Although we cannot instrument event-driven simulations at such large-scale, due to both physical memory limits (a hard constraint), as well as time budget (a soft constraint), we can *estimate* the expected memory occupancy and CPU times by means of a predictive model created by fitting and cross-validating several smaller scenarios (more than 50 in our case). Despite pertaining only to the specific implementation of ccnSim (and so being of limited interest), estimates are useful to project ModelGraft gains.

As for the *memory occupancy* (Mem_{sim}) of the optimized event-driven module

of ccnSim, we resorted to fitting the following formula:

$$Mem_{sim} = \beta_{cache} NC + \beta_{catalog} M + \beta_{fix} \quad \text{[Bytes]}, \tag{20}$$

where N is the number of nodes, C the cache size, M the catalog cardinality, while the three constants β_{cache}, $\beta_{catalog}$, and β_{fix}, take into account the memory cost for cache entries, catalog representation, and simulator environment, respectively. The fitting resulted in $\beta_{cache} = 165$, $\beta_{catalog} = 4$, and $\beta_{fix} = 20 \cdot 10^6$, with asymptotic standard errors of 3.28%, 0.02%, and 0.03%, respectively, showing that the (non)-scalability of classic event-driven simulation are mainly influenced by the cardinality of the catalog, i.e., $\mathcal{O}(M)$. In our specific case, Mem_{sim} estimates are shown in Fig. 5 by means of a contour plot, with isobars at 70 and 520 GB: to simulate the selected CDN-like scenario we would need almost 520 GB of RAM and 50 days to complete a single run. As for the CPU *time*, indeed, it resulted a linear variation with the total number of requests, i.e., $\mathcal{O}(R)$.

ModelGraft does not have to face with neither the presented estimates (prohibitive memory and CPU budgets), nor with drawbacks (e.g., message passing overhead) that classic parallelization approaches may suffer from [37], as it clearly appears from results reported in Tab. 2.

Consider the Access-like scenario first: even though the rejection inversion sampling technique optimizes the memory allocation of both simulation and ModelGraft, a mapping between seed copies (i.e., $\mathcal{O}(M)$) and respective repositories still needs to be stored somewhere; this justifies the substantial increment in memory allocation for the event-driven approach, with respect to previous scenarios with $M < 10^{10}$. On the contrary, the memory footprint of ModelGraft increases only slightly, since the mapping scales as $M' \ll M$: this leads, in the end, to projected memory gains which are significantly higher (i.e., more the three orders of magnitude) than the ones shown in Tab. 1. Regarding CPU time, instead, relative gains are similar (i.e., higher than two orders of magnitude) to Tab. 1, since, in this case, our initial T_C guesses were accurate enough to let ModelGraft converge after one cycle.

If we shift the attention to the CDN-like scenario, instead, we notice that memory gains increase by one further order of magnitude, with respect to the Access-like scenario: a larger downscaling factor, i.e., $\Delta = 10^6$, amplifies the difference between simulation and ModelGraft (same considerations about scalability of content/repository mappings hold also in this case). Conversely, CPU time gains are reduced due to the higher number of MC-TTL cycles needed to end the simulation: specifically, given that our initial T_C guesses were not accurate enough, ModelGraft took three cycles to converge, reducing the respective gains. However, a CPU time reduction of 96× remains very significant.

Table 3: Sensitivity analysis parameters

Variable parameters

Parameter	*Default*	*Range*	*Sec.*
T_C	T_C^{sim} (Tab. 4)	$[T_C^{sim}/(100u), T_C^{sim}u]$	[8.1]➡
Δ	10^5	$10^2, 10^3, 10^4, 10^5, 10^6$	[8.2]➡
Y	0.75	$1, 0.95, 0.9, 0.75, 0.5$	[8.3]➡
α	1	$0.8, 1, 1.2$	[8.4]➡
ε_{CV}	0.005	$\{0.005, 0.01, 0.05, 0.1\}$	[8.5]➡
ε_C	0.1	$\{0.05, 0.1, 0.5\}$	[8.5]➡

Fixed parameters

Parameter	*Value*
Catalog Cardinality (M)	10^9
Cache Size (C)	10^6
Number of Requests (R)	10^9
Cache Decision Policy	$LCE, LCP(1/10)$
Topology	4-level binary tree, NDN Testbed

8 Sensitivity analysis

To thoroughly test the correctness of ModelGraft, we conducted an extensive sensitivity analysis by varying all the main parameters involved in its workflow. Tab. 3 presents an overview of them, and, at the same time, it summarizes the adopted approach: that is, we considered the scenario presented in Sec. 7.1 as a baseline, with certain *fixed* parameters (bottom part), while we let the remaining ones, i.e., those with the highest impact on ModelGraft performance, *vary*, as reported in the upper part of Tab. 3.

Results related to each parameter will be extensively discussed in the following: a strong evidence of the *self-stabilization* property will be provided by simulating different scenarios with variable input T_C (Sec. 8.1); then, we will investigate on how the variation of the downscaling factor Δ may impact on the reduction of the space/time complexity, and on the accuracy of the gathered KPIs (Sec. 8.2); accuracy and consistency that could be influenced, also, by the Y parameter (Sec. 8.3). In the end, we validate the ModelGraft approach with different scenarios by varying the skewness of Zipf's distribution (Sec. 8.4), before concluding with considerations about the variability of the remaining parameters (Sec. 8.5). Each section is ended with a concise *remark*, in order to sum up lessons learned and useful guidelines.

Table 4: T_C values for validation scenario
(4-level binary tree, $M = R = 10^9, C = 10^6, Y = 0.75$)

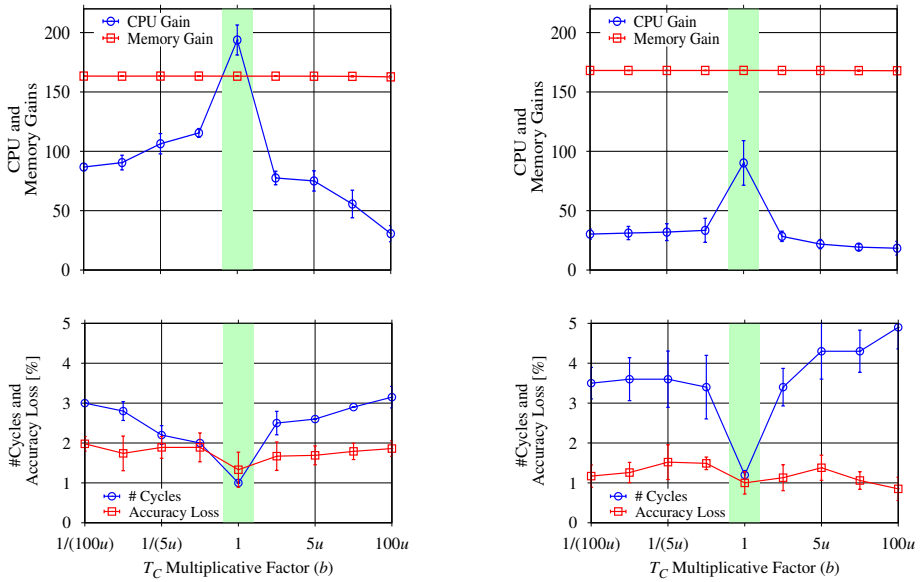
Level		LCE	LCP(1/10)	2-LRU (Name/Main)
0 (Root)	T_C values [s]	$16.7 \cdot 10^3$	$16.7 \cdot 10^4$	$20.0 \cdot 10^3$ / $76.4 \cdot 10^4$
1		$32.5 \cdot 10^3$	$31.4 \cdot 10^4$	$38.1 \cdot 10^3$ / $12.5 \cdot 10^5$
2		$63.0 \cdot 10^3$	$56.9 \cdot 10^4$	$71.9 \cdot 10^3$ / $20.4 \cdot 10^5$
3 (Leaves)		$11.1 \cdot 10^4$	$88.3 \cdot 10^4$	$11.1 \cdot 10^4$ / $22.6 \cdot 10^5$
p_{hit}		33.2%	35.4%	37.0%

8.1 Input sensitivity (T_C)

The downscaling factor Δ is not the sole aspect that influences ModelGraft performance in terms of CPU time; correctness of T_C values provided as input, in fact, may impact on the number of MC-TTL cycles needed before consistency is stated, as seen in Sec. 7.2 for the CDN-like scenario, thus affecting the overall CPU time. Since in typical use cases ModelGraft is meant to target (i.e., those presented in Sec. 7.2) exact T_C values are not known a priori, it is of primary importance to (i) assess its performance in the presence of T_C *guesses*, and to (ii) try to grasp some recurrent trends that may limit the inaccuracy of those guesses at the same time.

To this end, we first show, in Tab. 4, example of T_C values computed by simulating the same very-large scenario, introduced in Sec. 7.1, by means of the classic event-driven approach, and by varying the cache decision policy. It can be clearly noticed that T_C values vary more according to the *cache decision policy* than to the *topological position* inside the network; in particular, the more conservative the cache decision policy (e.g., LCP or 2-LRU), the bigger the T_C values. Intuitively, this reflects the fact that if nodes regulate the storage of new (unpopular) contents (as for conservative cache decision policies), (popular) contents will be cached for longer periods (i.e., bigger T_C values), thus inevitably increasing the overall hit probability (as the last row of Tab. 4 shows). In the specific case reported in Tab. 4, T_C values can vary by up to one order of magnitude among different policies, thus suggesting that, even with more complex and random topologies, the order of magnitude of T_C values can be guessed starting from smaller scenarios and by combining simple and topology-independent information, like cache decision policy, cache size, catalog cardinality, and so on.

With this baseline in mind, our approach in testing the resilience of ModelGraft against input T_C variability consists in purposely introducing estimates errors, for each node, in a controlled fashion; in particular, we set $T_C^0(i) = b(i)T_C^{sim}(i)$, where

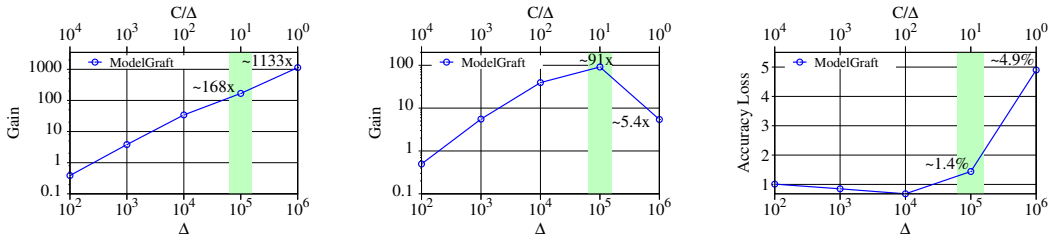


(c) LCE (d) LCP(1/10)

Figure 6: T_C sensitivity for very large scenario: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, LCE (a,c) and LCP(1/10) (b,d) cache decision policies.

$T_C^0(i)$ is the initial characteristic time for node i, $T_C^{sim}(i)$ is the accurate value taken from Tab. 4, and $b(i) \in [1/(Bu), Bu]$ is a multiplicative factor obtained by multiplying a bias value $B \in [1, 100]$ (equal for all the nodes), by a uniform random variable $u \in (0, 1]$. This means that we allow both *overestimating*, $b(i) > 1$, and *underestimating*, $b(i) < 1$, the accurate $T_C^{sim}(i)$. Notice that, in case of maximum bias (i.e., $B_{max} = 100$), $T_C^0(i)$ will differ from $T_C^{sim}(i)$ by up to two orders of magnitude, and the delta will be different for each node in the network due to the uniform variable u.

Fig. 6 reports the collected results related to four KPIs and two cache decision policies, that are LCE and LCP, as a function of the multiplicative factor b; in particular, CPU and *memory gains*, vs *accuracy loss* and number of MC-TTL *cycles*, are monitored. The exhaustive case set shown in Fig. 6, inspires the following observations: in the first place, (i) the presence of a feedback control loop efficiently nullifies the effects of the initial T_C variability, since the accuracy loss on p_{hit} remains always under 2%, regardless of the magnitude of the bias $b(i)$ (figs. 6(c) and 6(d)); this remarkably confirms ModelGraft to converge to the accurate system performance even for very harsh estimation of the initial T_C values, making the methodology very robust for practical purposes. Additionally, (ii) MC-TTL cycles



(a) Memory Gain (b) CPU Gain (c) Accuracy Loss (%)

Δ	Simulation	ModelGraft
10^2	6404	16549
10^3	6404	1674
10^4	6404	187
10^5	6404	38
10^6	6404	6

(d) Memory[MB]

Δ	Simulation	ModelGraft
10^2	26280	52810
10^3	26280	4741
10^4	26280	666
10^5	26280	287
10^6	26280	4841

(e) CPU time [s]

Δ	Simulation	ModelGraft
10^2	35.40	34.39
10^3	35.40	34.55
10^4	35.40	34.72
10^5	35.40	33.96
10^6	35.40	30.50

(f) p_{hit}[%]

Figure 7: Δ sensitivity - ModelGraft vs Simulation in 4-level binary tree, with $M = 10^9, R = 10^9, C = 10^6, Y = 0.75$, LCP(1/10) as cache decision policy: (a-c) Respective gains, (d-f) Tabular values.

always remain under a maximum threshold of five, regardless of the cache decision policy, and even in those extreme cases of under/over estimation, where the number of MC-TTL cycles increases with respect to ideal situations (i.e., the central highlighted bars of Fig. 6, where exact T_C estimates are provided as input). It follows that, (iii) while CPU gain is maximum in the absence of bias (about 200x, and 90x for LCE and LCP, respectively, when no iterations are required), the benefits brought by ModelGraft are, however, still significant (always higher than 50x, for both LCE and LCP) even for very large T_C biases (e.g., 100x overestimation or 1/100x underestimation). Finally, (iv) memory gain is, as expected, independent of the initial T_C bias.

Remark 1. *ModelGraft is* self-contained *and* self-stabilizing*: the accuracy of the gathered metrics is always guaranteed, regardless of input T_C values. Performance, in terms of memory and CPU time reduction, are maximum, i.e., more than two orders of magnitude, when exact T_C values are provided as input; moreover, even with very inaccurate T_C estimates, ModelGraft converges within few cycles, always guaranteeing more than one order of magnitude of memory and CPU time reduction w.r.t. event-driven simulation.*

8.2 Downscaling factor (Δ) sensitivity

ModelGraft is designed to combine two objectives that may conflict with each other: (i) maximizing performance, and (ii) minimizing accuracy losses. As seen previously,

the presence of a closed-loop control feedback helps neutralizing potential negative effects that T_C guesses may have on the accuracy of the gathered KPIs, and, at the same time, it minimizes losses in terms of CPU times. However, another key parameter surely impacts on ModelGraft performance, in terms of both CPU time and memory occupancy: that is, the *downscaling factor* Δ. In this subsection we present results related to a sensitivity analysis executed by simulating the same very-large of Sec. 7.1: LCP(1/10) is used as a cache decision policy, exact T_C estimates (Tab. 4) are provided as input, and the downscaling factor Δ is varied in the interval $[10^2, 10^6]$ (i.e., target cache $C' = C/\Delta$ is progressively decreased from 10^4 to 10^0).

Fig. 7 reports the three KPIs, i.e., memory gain, CPU gain, and accuracy loss, as well as tables with correspondent numerical values. First of all, inefficiencies of ModelGraft emerge when choosing a downscaling factor from either extremes of the interval (i.e., $\Delta = 10^2$ or $\Delta = 10^6$), as shown in Figures 7(a) and 7(b); indeed, when Δ is *too small*, i.e., $\Delta = 10^2$, both memory occupancy and CPU time exceed the respective values measured with the classic event-driven simulation. This is mainly due to the fact that TTL caches, having no size limit (as opposed to LRU ones), reach their target cache size $C' = C/\Delta$ only at steady-state, while peaks where $C' > C$ are common during the transient state, thus requiring the allocation of more memory space. Furthermore, the implementation of TTL caches does not require the ordered structure which characterizes LRU ones, thus inevitably degrading the lookup time, and, in the end, the total CPU time, as reported in Fig. 7(b).

On the contrary, when the downscaling factor is *too big*, e.g., $\Delta = 10^6 = C$, ModelGraft, despite a very low memory footprint, as shown in Fig. 7(a), starts presenting problems related to CPU time and accuracy, as reported in Figures 7(b) and 7(c). The causes of this performance degradation are mainly two: on one side, (i) the very small downscaled request rate $\Lambda' = \Lambda/\Delta$ slows down the collection of the W samples needed to compute Eq. 15 (Sec. 5.1), resulting in longer transient periods; on the other side, and most importantly, (ii) a very small target cache $C' = C/\Delta = 1$ may hamper the measurement step of the consistency check, Eq. 18, as anticipated at the end of Sec. 6.2, thus resulting in unjustified (i.e., when exact T_C estimates are provided as input) and additional MC-TTL cycles. As a consequence, the controller action will, in the end, diverge, thus inevitably increasing accuracy losses, as reported in Fig. 7(c) (in the specific case of Figures 7(b) and 7(c), ModelGraft simulations ended since the maximum number of cycles, set to 21 for practical reasons, had been reached; this means that inaccuracy may have been increased with additional cycles).

It follows that, the only constraint that could limit the magnitude of the downscaling factor Δ comes from the cache size C; once passed a certain threshold, e.g., when $C' = C/\Delta = 1$, ModelGraft becomes inefficient and unstable. However, when the downscaling factor is chosen properly, i.e., when $C' \geq 10$, the best performance are guaranteed: a CPU time reduction of about 91x, and a memory reduction of

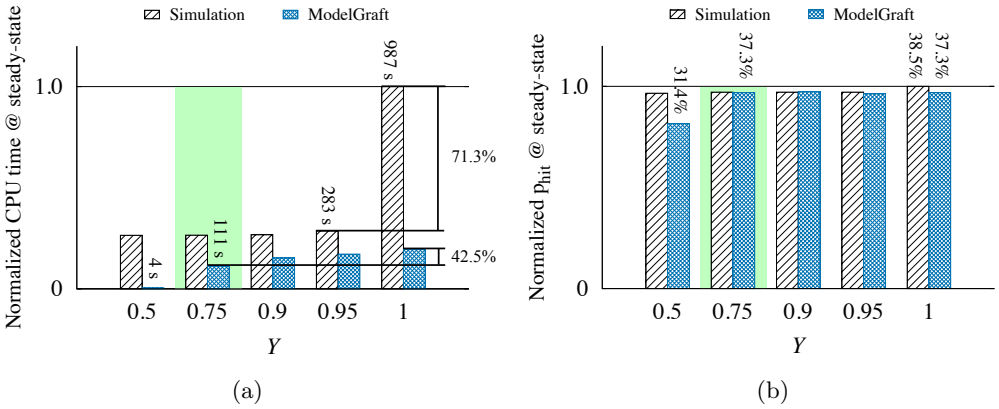


Figure 8: Y sensitivity - ModelGraft and event-driven simulation in very large scenario: NDN Testbed, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, LCE as cache decision policy - normalized (a) p_{hit} and (b) CPU time when the system converges to steady-state.

168x, as reported from Figures 7(a) and 7(b). Since we experienced the same trend for all the executed simulations, we can consider the aforementioned criterion for the dimensioning of Δ as a *de facto* rule.

Remark 2. *The downscaling factor Δ has to be chosen in accordance with the simulated scenario, and especially with the cache size C; when* too small*, memory occupancy and CPU time may be higher then the respective values for the event-driven simulation; when, instead,* too big*, both CPU and accuracy are degraded, due to instability introduced by unmeasurable quantities. It follows that* Δ should be set in the interval $(10^2, C/10]$*, and that maximum performance gains are expected for $\Delta = C/10$.*

8.3 Consistency (Y) sensitivity

Performance in terms of CPU time are also influenced by the Y parameter: since ModelGraft has an *adaptive steady-state monitor* which varies the length of the transient period according to the simulated scenarios, as already seen in Sec. 5.2, the Y parameter allows to tune the number of nodes that will be considered for the convergence computation. The subtle task here is reaching the correct balance between performance and accuracy: indeed, according to inequality (16), requiring the convergence of all network nodes may uselessly extend the transient period; for example, if we take into account the 4-level binary tree topology depicted in Fig. 4(a), it may be intuitive to think at removing the root from the computation, as the node with the highest convergence time and with the lowest impact on the

overall performance (since content requests will be progressively satisfied by nodes at lower levels). This aspect may be emphasized with more complex topologies and/or elaborate routing strategies, where some nodes may receive such a very derisory amount of traffic (thus being irrelevant in the overall network performance) that their convergence could be hardly detected.

We, then, investigate on this aspect by considering the same very-large scenario in terms of catalog cardinality, cache size, number of requests, and downsizing factor (that is, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$), but with LCE as cache decision policy, and the NDN Testbed [2] as our topology. We purposely vary the Y parameter in the interval $[0.75, 1]$, for both ModelGraft and classic event-driven simulation.

We monitor two KPIs in particular: the *convergence time*, that is the elapsed time after which the simulation is considered at steady-state, and the measured p_{hit} in that specific time instant; results are reported in figs. 8(a) and 8(b), respectively, which show normalized values w.r.t. maximum (Fig. 8(a)) and ground truth value of p_{hit} gathered from event-driven simulation (Fig. 8(b)). The first thing to notice is that, as long Y is reduced (i.e., fewer nodes contribute to the steady-state evaluation), the convergence time is reduced. For example, for the classic event-driven approach, the exclusion of only 5% of nodes (i.e., from $Y = 1$ to $Y = 0.95$) leads to a convergence time reduction of about 71%, while, for ModelGraft, the exclusion of the same amount of nodes lowers the convergence time of about 12% (percentage that increases up to 42.5% if $Y = 0.75$, as shown in Fig. 8(a)). At the same time, there exists almost no variation in the p_{hit} measured at stability, for both approaches. This indicates that, as long as Y is decreased, we may exclude nodes that have, indeed, the smallest impact on the monitored KPIs (i.e., peripheral nodes with a very small hit ratio compared to the mean one), and that uselessly extend the duration of the transient period.

However, as also highlighted in Fig. 8(b), some inaccuracies may be introduced if Y goes too low, e.g., $Y = 0.5$. In this case, since too many nodes are cut out from the convergence computation (i.e., 50%), the p_{hit} at stability, especially for ModelGraft, starts to diverge from the real value, thus introducing non-negligible accuracy losses.

Remark 3. *The exact setting of the Y parameter allows the dynamic steady-state monitor to state the convergence of the network* faster*. To avoid inconsistencies in the gathered metrics, a lower bound on Y should be considered, e.g., $Y \in [0.75, 1)$.*

8.4 Scenario (α) sensitivity

In the end, in order to extend the applicability of ModelGraft, we verify its consistency in different scenarios. In particular, we vary the Zipfian distribution by changing the exponent $\alpha \in \{0.8, 1, 1.2\}$, and we compare the two approaches by means of the same very-large scenario (Sec. 7.1), and by monitoring P_{hit}, CPU

time, and memory occupancy.

Table 5: α sensitivity - ModelGraft and event-driven simulation in very-large scenario: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5, Y = 0.75$, LCE.

α	Technique	p_{hit}	CPU time	Gain	Mem [MB]	Gain
0.8	Simulation	7.96%	22.8 h	**461x**	6356	**168x**
	ModelGraft	7.85%	179 s		38	
1	Simulation	33.2%	11.4 h	**194x**	6371	**168x**
	ModelGraft	31.4%	211 s		38	
1.2	Simulation	52.9%	4.2 h	**52x**	6393	**168x**
	ModelGraft	52.6%	286 s		38	

Tab. 5 reports numerical values and relative gains of ModelGraft w.r.t. event-driven simulation. As it is clearly noticeable, ModelGraft accuracy is confirmed also for $\alpha = 0.8$ and $\alpha = 1.2$, with losses of 0.1% and 0.3%, respectively. Interestingly, the CPU gain for the case $\alpha = 0.8$ is more than twice the one obtained with $\alpha = 1$ (i.e., 461× instead of 194×), making ModelGraft ending the simulation in 179 seconds, instead of almost 1 day of the event-driven simulation. For the case $\alpha = 1.2$, instead, the computational gain is smaller (52×): specifically, the CPU time of the classic event-driven approach was already smaller than the respective case with $\alpha = 0.8$, since a hit ratio higher than 50% translates in almost no propagation of content requests (i.e., generated events), since they are mostly satisfied by leaf caches. However, a reduction of 52× is still noteworthy. In the end, as expected, memory reduction (168×) is not influenced by the skewness of the catalog. The sensitivity on α adds, then, further flexibility to ModelGraft as a scalable method for the analysis of Internet-scale cache networks.

Remark 4. *The popularity distribution of the content catalog does not affect the accuracy and the performance of ModelGraft; even though different gains, in terms of CPU reduction, are associated to different skewness values, they are always higher than one order of magnitude.*

8.5 Convergence (ε_{CV}) and Consistency (ε_C) Sensitivity

Convergence and *consistency* thresholds, i.e., ε_{CV} and ε_C, respectively, might, also, influence both the execution time and the accuracy of ModelGraft.

As a consequence, in this section we investigate the performance variations of ModelGraft when varying such parameters. As shown in Tab. 3, the respective sensitivity concerns, in this case, $\varepsilon_{CV} \in \{0.005, 0.01, 0.05, 0.1\}$ and $\varepsilon_C \in \{0.05, 0.1, 0.5\}$.

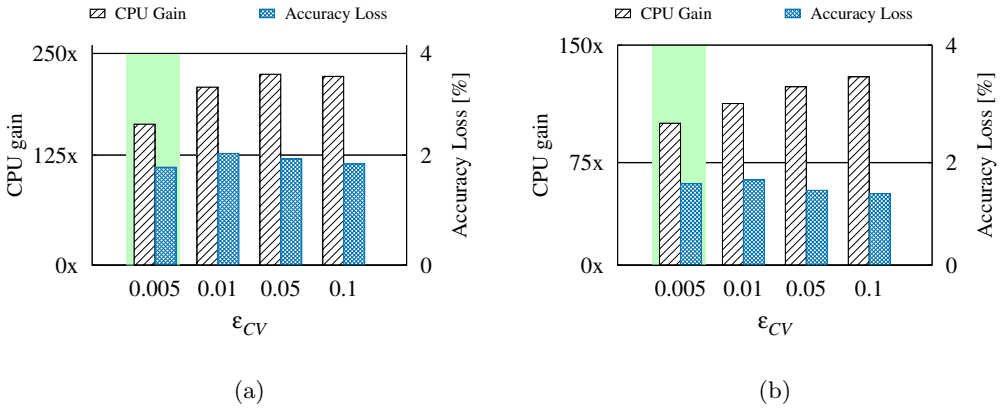


Figure 9: ε_{CV} sensitivity - ModelGraft in very large scenario: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, CPU gain and Accuracy Loss in (a) LCE and (b) LCP(1/10).

Fig. 9 reports results related to ε_{CV} sensitivity for both LCE and LCP(1/10): the same very large scale scenario introduced in Sec. 7.1 is used as a base for the comparison between ModelGraft and event-driven simulation, when jointly varying ε_{CV}. CPU gain and accuracy loss are considered as representative KPIs; in particular, they are both computed by comparing the result of the two techniques, when executed with the same ε_{CV} value. As expected, the bigger ε_{CV}, the faster ModelGraft goes to convergence for both cache decision policies, owing to a less stringent requirement on the coefficient of variation of the average hit probability. The result is a reduction of ModelGraft execution time, which is, at the same time, associated with an unvaried accuracy. However, we select $\varepsilon_{CV} = 0.005$ as the default value in order to compare ModelGraft against event-driven simulation in the most conservative case.

The effects of ε_C sensitivity, instead, are partially represented in Fig. 10, where the number of cycles and the accuracy loss are monitored for different values of ε_C, and for two cache decision policies, i.e., LCE and LCP(1/10). The first clear consequence is linked to *small* ε_C values: indeed, when ε_C is too small (e.g., $\varepsilon_C = 0.05$), ModelGraft could be forced to execute multiple MC-TTL cycles, even when it is not necessary. That is, considering that results reported in Fig. 10 are obtained by feeding ModelGraft with exact T_C values as input, the case for $\varepsilon_C = 0.05$ in Fig. 10, i.e., $\#Cycles >> 1$, should not happen.

On the opposite, the risk of considering milder conditions, i.e., large ε_C values, is that ModelGraft could end the simulation before the effective number of cycles which are needed to limit the accuracy loss within a certain threshold (i.e., less than 2%). When performed, then, an input sensitivity, like the one presented in Sec. 8.1, where we monitor all the KPIs when executing ModelGraft with $\varepsilon_C = 0.1$

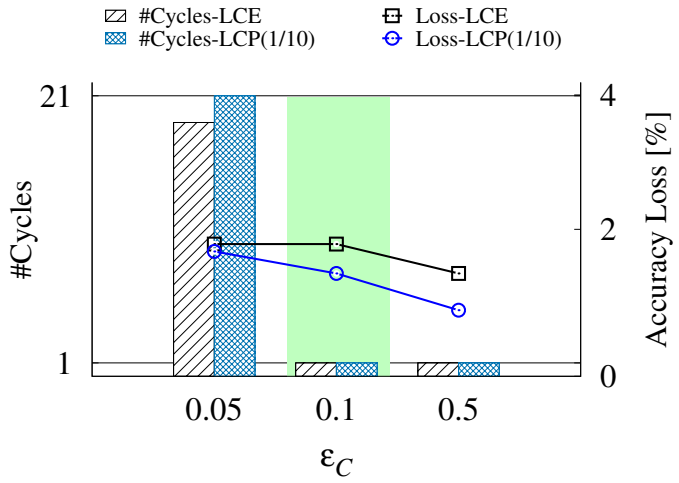


Figure 10: ε_C sensitivity - ModelGraft in very large scenario: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, LCP(1/10) as cache decision policy - # Cycles and Accuracy Loss.

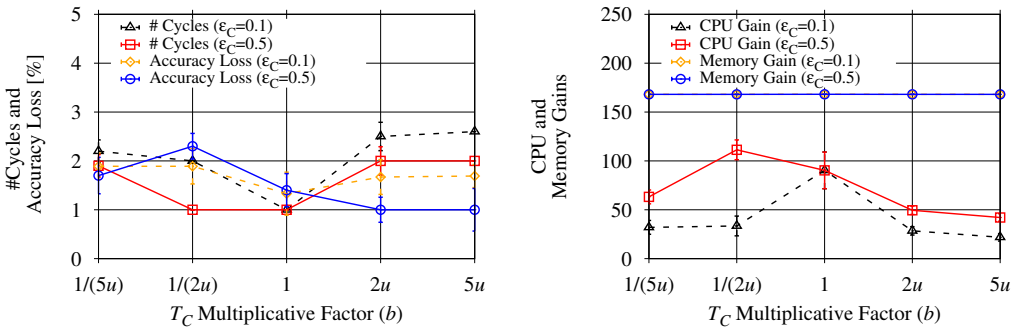
and $\varepsilon_C = 0.5$.

What emerges from Fig. 11 is that a bigger ε_C threshold is associated with a smaller number of cycles, especially when incorrect T_C values are provided as input to ModelGraft, and, consequently, with higher CPU gains. However, this phenomenon comes with a degradation of the accuracy, as for the case $b = 1/(2u)$ in Fig. 11(a), where the accuracy loss is higher than 2%. As a final result, we select $\varepsilon_C = 0.1$ as our default value, in order to avoid the risk of altering the accuracy of ModelGraft.

9 Related Work

Hybrid approaches have been considered to make it practical to study large scale networks even with commodity hardware, and at a reasonable time scale. From one side, the concept of inferring key aspects of large systems from the study of equivalent and scaled-down versions has been adopted in other several domains, from cosmology and biology, to the more closer communication networks, in the forms of large scale IP networks [34, 25], wireless sensor networks [24], and control theory [7]. On the other side, the design of efficient simulative techniques [15] has been preferred to the less scalable experimentation on real testbeds as a mean to empirically evaluate algorithmic solutions for large-scale and distributed systems.

What presented in this paper finds a strict correspondence with the work in [34], where the authors present a technique to scale large IP networks, in order to reduce



(a) (b)

Figure 11: ε_C sensitivity - ModelGraft with $\varepsilon_C \in \{0.1, 0.5\}$ in very large scenario: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, LCP(1/10) - (a) # Cycles and Accuracy Loss, (b) CPU and Memory gains.

the computational requirements of simulations and simplify performance prediction. Their idea consists in feeding a suitable scaled version of the system with a sample of the input traffic; in particular, they differentiate the scaling rule according to the type of TCP/UDP flows traversing the network: for IP networks with short and long flows they demonstrate, both analytically and with simulation, that their scaling technique leaves certain metrics, such as the distribution of the number of active flows and of their normalized transfer time, virtually unchanged in the scaled system. For networks with long-lived flows controlled by queue management schemes, instead, they demonstrate, through differential-equation models, that a different scaling approach leaves queuing delay and drop probability unchanged. In this latter case, the proposed approach drastically reduces the CPU time of *ns* simulations.

TCP networks are also considered in [25], where the authors propose a scalable model which is easily comparable with discrete event simulators due to its time-stepped nature. In particular, they refine a known analytical model [33] based on ordinary differential equations, and they solve it numerically using the Runge-Kutta method. Results show that their approach yields accurate results with respect to those of the original networks, and, at the same time, it is able to speedup the completion time of orders of magnitude with respect to packet level and discrete events simulators like *ns*.

This work is the first to apply these concepts to the study of cache networks. Clearly, caching dynamics are intrinsically different from system-level aspects of wireless sensor networks [24], or the steady state throughput of TCP/IP networks [34, 25]. Our methodology is not only novel, but also practical, as it is fully integrated in open-source tools [1].

An hybrid approach is also considered in the simulation framework described in [15], namely SimGrid. The purpose is to offer a versatile simulation framework for the performance evaluation of large scale and distributed systems (e.g., Grids, High Performance Computing (HPC), P2P, Clouds) which would preserve both accuracy and scalability at the same time. SimGrid offers several APIs which let users interface their code with its core; regardless of the API, the simulation application consists of a set of communication and computational activities which are executed on simulated hardware resources, defined in terms of compute capacities (e.g., CPU cycles per time unit). They are interconnected via a network topology that comprises network links and routing elements, defined by bandwidth capacities and latencies. Specifically, SimGrid simulates the execution of the aforementioned activities through a Max-Min optimization problem, which provides instantaneous resource shares, i.e., which resources are used by which activities. Moreover, a flow-level model is used at network level: an end-to-end communication is characterized only by the bandwidth associated to the relative *flow*.

10 Conclusion

This work proposes ModelGraft, an innovative hybrid methodology addressing the issue of performance evaluation of large-scale cache networks. The methodology grafts elements of stochastic analysis to MonteCarlo simulation approaches, retaining benefits of both methodology classes. Indeed, ModelGraft inherits simulation *flexibility*, in that it can address complex scenarios (e.g., topology, cache replacement, decision policy, etc.). Additionally, ModelGraft is implemented as a simulation engine to retain simulation *simplicity*: given its self-stabilization capability, ModelGraft execution is decoupled from the availability of accurate input T_C values, which is completely transparent to the users. Results presented in this paper finally confirm both the *accuracy* and the *high scalability* of the ModelGraft approach: CPU time and memory usage are reduced by (at least) two orders of magnitude with respect to the classical event-driven approach, while accuracy remain within a 2% band.

References

[1] ccnSim Simulator. `http://perso.telecom-paristech.fr/~drossi/ccnSim`.

[2] NDN Testbed web page. `http://named-data.net/ndn-testbed/`.

[3] Zipf distributed random number generator . `https://github.com/apache/commons-math/blob/master/src/main/java/org/apache/commons/math4/distribution/ZipfDistribution.java`.

[4] A. Araldo, D. Rossi, et al. Cost-aware caching: Caching more (costly items) for less (isps operational expenditures). 2015.

[5] S. Arianfar and P. Nikander. Packet-level Caching for Information-centric Networking. In *ACM SIGCOMM, ReArch Workshop*. 2010.

[6] F. Baccelli, S. Machiraju, et al. On optimal probing for delay and loss measurement. In *Proc of the ACM SIGCOMM IMC*. 2007.

[7] M. Branicky, V. Borkar, et al. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31, 1998.

[8] M. Cha, H. Kwak, et al. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *ACM IMC*. 2007.

[9] H. Che, Y. Tung, et al. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE JSAC*, 20(7):1305, 2002.

[10] H. Che, Z. Wang, et al. Analysis and design of hierarchical web caching systems. In *Proc of IEEE INFOCOM*. 2001.

[11] A. Dan and D. Towsley. An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. *SIGMETRICS Perform. Eval. Rev.*, 18(1):143, 1990.

[12] N. Fofack, P. Nain, et al. Performance evaluation of hierarchical TTL-based cache networks. *Elsevier Computer Networks*, 65:212 , 2014.

[13] C. Fricker, P. Robert, et al. A versatile and accurate approximation for LRU cache performance. In *Proc. of International Teletraffic Congress (ITC 24)*. 2012.

[14] N. Gast and B. V. Houdt. Transient and steady-state regime of a family of list-based cache replacement algorithms. In *Proc of ACM SIGMETRICS Conference*, pages 123–136. 2015.

[15] e. a. H. Casanova. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899, 2014.

[16] M. Hefeeda and O. Saleh. Traffic modeling and proportional partial caching for peer-to-peer systems. *IEEE/ACM Transactions on Networking*, 16(6):1447, 2008.

[17] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401, 1980.

[18] W. Hörmann and G. Derflinger. Rejection-inversion to generate variates from monotone discrete distributions. *ACM Trans. Model. Comput. Simul.*, 6(3):169, 1996.

[19] J. Jaeyeon, A. W. Berger, et al. Modeling TTL-based Internet caches. In *Proc. of IEEE INFOCOM*. 2003.

[20] T. Johnson, D. Shasha, et al. 2q: A low overhead high performance buffer management replacement algorithm. In *20th International Conference on Very Large Data Bases (VLDB)*, pages 439–450. 1994.

[21] W. King. Analysis of paging algorithms. In *In IFIP Congress*, pages 490–490. 1971.

[22] N. Laoutaris, H. Che, et al. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7), 2006.

[23] E. Leonardi and G. Torrisi. Least Recently Used caches under the Shot Noise Model. In *Proc. of IEEE Infocom*. 2015.

[24] P. Levis, N. Lee, et al. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proc. of ACM SenSys*. 2003.

[25] Y. Liu, F. Presti, et al. Scalable Fluid Models and Simulations for Large-scale IP Networks. *ACM Trans. Model. Comput. Simul.*, 14(3):305, 2004.

[26] V. Martina, M. Garetto, et al. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*. 2014.

[27] D. Berger et al. Exact Analysis of TTL Cache Networks: The Case of Caching Policies Driven by Stopping Times. In *Proc. of ACM SIGMETRICS Conference*, pages 595–596. 2014.

[28] S. Fayazbakhsh et al. Less Pain, Most of the Gain: Incrementally Deployable ICN. *SIGCOMM Comput. Commun. Rev.*, 43(4):147, 2013.

[29] N. Fofack et al. On the performance of general cache networks. In *Proc. of VALUETOOLS Conference*, pages 106–113. 2014.

[30] K. Pentikousis et al. Information-centric networking: Evaluation methodology. Internet Draft, https://datatracker.ietf.org/doc/draft-irtf-icnrg-evaluation-methodology/, 2015.

[31] M. Rosenblum et al. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel Distrib. Technol.*, 3(4):34, 1995.

[32] G. Xylomenos et al. A survey of information-centric networking research. *Communication Surveys and Tutorials, IEEE*, 16(2):1024, 2014.

[33] V. Misra, W. Gong, et al. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *SIGCOMM Comput. Commun. Rev.*, 30(4):151, 2000.

[34] R. Pan, B. Prabhakar, et al. SHRiNK: A Method for Enabling Scaleable Performance Prediction and Efficient Network Simulation. *IEEE/ACM Trans. Netw.*, 13(5):975, 2005.

[35] E. Rosensweig, D. Menasche, et al. On the steady-state of cache networks. In *Proc. of IEEE INFOCOM*. 2013.

[36] E. J. Rosensweig, J. Kurose, et al. Approximate Models for General Cache Networks. *IEEE INFOCOM*, pages 1–9, 2010.

[37] G. Rossini and D. Rossi. ccnSim: a highly scalable CCN simulator. In *Proc. of IEEE ICC*. 2013.

[38] G. Rossini and D. Rossi. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proc. of ACM SIGCOMM ICN*. 2014.

[39] M. Tortelli, D. Rossi, et al. ICN software tools: survey and cross-comparison. *Elsevier Simulation Modelling Practice and Theory (SIMPAT)*, 63:23 , 2016.