POLITECNICO DI TORINO
Repository ISTITUZIONALE

Parallel Simulation of Very Large-Scale General Cache Networks

Original
Parallel Simulation of Very Large-Scale General Cache Networks / Tortelli, M., Rossi, D., Leonardi, E.. - In: IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. - ISSN 0733-8716. - ELETTRONICO. - 36:8(2018), pp. 1871-1886. [10.1109/JSAC.2018.2844938]

Availability:
This version is available at: 11583/2711472 since: 2018-12-14T13:54:12Z

Publisher:
IEEE

Published
DOI:10.1109/JSAC.2018.2844938

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

17 November 2026

Parallel Simulation of Very Large-Scale General Cache Networks

Michele Tortelli*, Dario Rossi*, Emilio Leonardi†
* Telecom ParisTech, Paris, France
† Politecnico di Torino, Torino, Italy

***Abstract*—In this paper we propose a methodology for the study of general cache networks, which is intrinsically scalable and amenable to parallel execution. We contrast two techniques: one that *slices the network*, and another that *slices the content catalog*. In the former, each core simulates requests for the whole catalog on a subgraph of the original topology, whereas in the latter each core simulates requests for a portion of the original catalog on a replica of the whole network. Interestingly, we find out that when the number of cores increases (and so the split ratio of the network topology), the overhead of message passing required to keeping consistency among nodes actually offsets any benefit from the parallelization: this is strictly due to the correlation among neighboring caches, meaning that requests arriving at one cache allocated on one core may depend on the status of one or more caches allocated on different cores. Even more interestingly, we find out that the newly proposed *catalog slicing*, on the contrary, achieves an ideal speedup in the number of cores. Overall, our system, which we make available as open source software, enables performance assessment of large-scale general cache networks, i.e., comprising hundreds of nodes, trillions contents, and complex routing and caching algorithms, in minutes of CPU time and with exiguous amounts of memory.**

I. Introduction

In today's Internet, content distribution plays a central role. Future Internet architectures, such as 5G Mobile Edge computing [6] and Information Centric Networks [30], further make caching an integral part of the network. Caching is attractive as it helps meeting 5G requirements of lower latency, while relieving traffic load at the same time, and thus also providing users with higher available bandwidth.

Yet the performance evaluation of cache networks is made challenging by many factors. To start with, the *correlation between miss streams of neighboring caches* (particularly true for general cache networks with arbitrary topologies, complex routing, cache decisions and replacement algorithms), either makes analytical models fail to capture system behaviors, or it sharply increases the complexity of their numerical solution. This makes event-driven simulation an appealing alternative. Additionally, the global and pervasive *scale of cache networks* further adds complexity to performance evaluation. Indeed, due to prohibitive CPU and memory requirements, simulating large-scale scenarios is hard. For instance, consider that just storing the description of a popularity law for realistic content catalogs [29] requires allocating a vector of 10^{12} 64-bits double precision floating points, i.e., 8 TB of RAM memory. While downscaling is appealing here, however ingenuity is needed as naïve approaches introduce tremendous biases in the results, rendering the evaluation useless (cfr Sec.II).

To get beyond these limits, we proposed a hybrid technique named *ModelGraft* [40], which is able to aggressively downscale the original system while fully preserving its properties. This technique relies on the assumption that caches (for a large class of replacement policies) are well approximated by a Time-To-Live (TTL)-based equivalent, whose eviction timer is set equal to the *characteristic time* (T_C) of the original system. This allows us to use *meta-objects* in the downscaled system, whose role is to represent *aggregate of objects*. We sample objects in a way that maintains structural properties of the original catalog. We implemented *ModelGraft* as an alternative hybrid engine into an already available and open-source simulator. We show that it consistently downscales the original system, gaining several orders of magnitude in terms of CPU time (100-300×) and memory occupancy (10,000×), with a very good accuracy on the cache hit rate (<2% error).

This work makes a significant step further, by proposing and implementing a parallel technique for the evaluation of general cache networks that achieves an ideal speedup. We show that this is non trivial due to the correlation between neighboring caches (i.e., miss-streams' propagation): therefore, partitioning the simulation of nodes over multiple cores incur a massive overhead associated to Message Passing Interface (MPI), with an overall slowdown of the execution time. Our original contribution is to recognize that the decoupling principle of the Che's approximation [8] yields to *dynamics of meta-objects that are completely independent from each other*. The original network can be well described by the aggregate of several independent networks, each comprising the *whole original topology, but simulating requests for portions of the original content catalog*. This constitutes a significant breakthrough, as it effectively makes each thread independent, thus resulting in very good scaling properties of the parallelization technique.

In more details, we design, implement and evaluate two techniques (network vs content slicing) for parallel simulation of cache networks, and additionally contrast two downscaling strategies. These techniques make a significant leap by reducing the simulation time by almost 10000x with respect to event-driven simulation, and by up to 70x with respect to our previous work ModelGraft [40]. We make our code, able to simulate trillion-scale objects scenarios in just one hour of CPU time, available at [1].

In the rest of this paper we motivate this research (Sec.II) and recall ModelGraft [40] (Sec.III). We next detail the different alternatives we implement to parallelize the evaluation of cache networks (Sec.IV–Sec.V). We then thoroughly assess

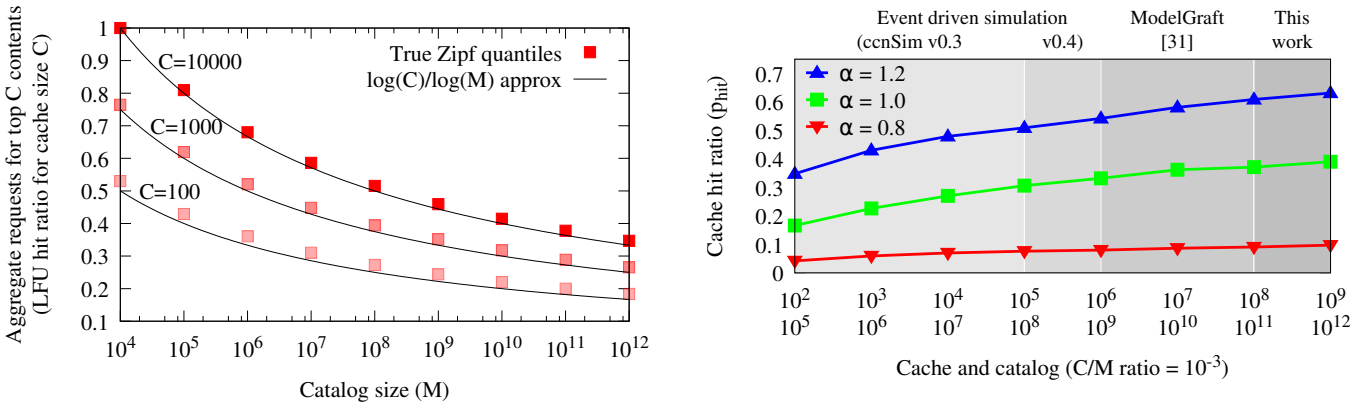


Fig. 1. Motivation: Cache network performance misestimation for (a) fixed cache size and downscaled catalog vs (b) jointly downscaled cache and catalog.

their performance (Sec.VI-Sec.VII), and finally review related work (Sec.VIII) and summarize our findings (Sec.IX).

II. Motivation

The need for approaches such as the one proposed in this paper clearly emerges by gauging the distortion entailed by naïve downscaling of important system parameters. In the case of cache networks, two options naturally arise: either (i) shrinking the catalog size M, while keeping other parameters, like the cache size C, unchanged, or (ii) jointly shrinking both catalog and cache size, thus keeping their ratio C/M unchanged. As we shall see, none of these options leads to meaningful results in practice.

A. Downscaling catalog M

First, note that important catalog properties, such as Zipf's popularity slope α, are gathered from real systems with large catalog size M. It is quite intuitive that if α is measured over a catalog of M objects, its use into a system of size $M' \ll M$ introduces a distortion of the gathered performance. [1]

For instance, the percentage of aggregated requests involving the first C most popular objects, out of a catalog of size M, is depicted in Fig. 1-(a) for $\alpha = 1$: it can be seen that downscaling the catalog size while keeping the cache size constant increases the relative volume of requests directed to the head of the catalog, *translating into a significant overestimation of the cache hit*. Downscaling the catalog from $M{=}10^{12}$ to $M'{=}10^4$ in a system with a stationary catalog and a single cache operating with Least Frequently Used (LFU) replacement policy, which statically places the C most popular objects in a cache of size $C < M$, would imply a relative error of 200%, 350% and 500% for caches storing $C = 100$, 1000 or 10000 objects respectively.

More generally, the quantile of the Zipf distribution for an object of rank C, i.e., the sum of frequencies for objects having rank $\leq C$ can be written as

$$P(C) = \frac{\sum_{k=1}^{C} 1/k^{\alpha}}{\sum_{k=1}^{M} 1/k^{\alpha}}. \quad (1)$$

[1]Traffic is synthetically generated in our simulator. We refer the reader to [43] for a discussion on the necessity of synthetic traffic models and how to generate realistic traffic traces.

which corresponds to the aggregate rate of requests for the most popular C objects, and in the case of LFU, also corresponds to the hit rate h of a cache of size C. In case $\alpha = 1$, numerator and denominator of 1 correspond to the Harmonic numbers $H_C = \sum_{k=1}^{C} 1/k$ (and H_M), which are known to be well approximated asymptotically by a logarithmic function, since

$$\lim_{X \to \infty} H_X - \ln(X) = \gamma, \quad (2)$$

where $\gamma \approx 0.57$ is the Euler-Mascheroni constant. Hence, in the large C and M regime, we have:

$$h = P(C) \approx \ln(C)/\ln(M). \quad (3)$$

It follows that, for a given cache size C in the LFU model, the cache hit error in the downscaled system goes as:

$$\frac{\ln(M) - \ln(M')}{\ln(M)}, \quad (4)$$

where $M' < M$ is the downscaled catalog size. Thus, overestimation error slowly decreases as M' increases toward the true catalog size M.

B. Jointly downscaling catalog M and cache C

Yet, compensating the aforementioned bias is not trivial: for instance, *an opposite behavior leading to a significant underestimation of the cache hit-ratio* happens when catalog cardinality and cache size are jointly downscaled in order to keep their ratio constant. This can be noticed from Fig. 1-(b), which reports the cache hit ratio, p_{hit}, of a 15-nodes binary tree of depth four for different Zipf's skews $\alpha \in \{0.8, 1, 1.2\}$. Results refer to different combinations of catalog M and cache C sizes; specifically the cache to catalog ratio has been fixed to $C/M = 0.1\%$. The figure also highlights the scale achievable with different performance evaluation methodologies (i.e., event-driven simulation, *ModelGraft* [40], and the parallel technique proposed in this work).

In particular, it clearly appears that p_{hit} increases at each joint increment of C and M (translating into a potential error of 135% between the biggest and the smallest scenario with $\alpha = 1$, as an example), which is tied to the increasing ratio

between the cache size C and the different percentiles of the catalog when jointly increasing both C and M.

The net effect of these over- or under-estimation errors is that simulation results, gathered over naïvely downscaled versions of the original system, are not more quantitatively accurate than uninformed guesses: this testifies the need for approaches such as our previous single-core technique [40], as well as the parallel technique we propose in this paper. Of course, other properties of the evaluation scenario, such as topological [34] and workload [42] aspects are still important, and any devised technique should correctly support these aspects (see Sec. III-B).

III. Background on ModelGraft

A. Modeling background

Che's approximation [8], conceived for a LRU cache, is essentially a mean-field approximation which greatly simplifies the analysis of the interactions between different contents inside a cache. The approximation consists in replacing the *cache characteristic time* $T_C(m)$ for content m, i.e., the (random) time since the last request after which object m will be evicted from the cache (due to arrival of requests for other contents), with a *constant eviction time* T_C (independent from the content itself, and rather a property of the whole cache).

Notice that under LRU, content m is considered to be in the cache at time t, if and only if, at least one request for m has arrived in the interval $(t - T_C, t]$. Supposing a catalog with cardinality M, for which requests are issued following an Independent Reference Model (IRM) with aggregate rate $\Lambda = \sum_m \lambda_m$, the probability $p_{in}(m)$ for content m to be in a LRU cache at time t can be expressed as:

$$p_{in}(\lambda_m, T_C) = 1 - e^{-\lambda_m T_C}. \tag{5}$$

Denoting with $\mathbb{1}_{\{\mathrm{A}\}}$ the indicator function for event A, and considering a cache of size C, we have, by construction, that $C = \sum_m \mathbb{1}_{\{m \text{ in cache at t}\}}$. Averaging both sides, we obtain:

$$C = \sum_m \mathbb{E}\left[\mathbb{1}_{\{m \text{ in cache at t}\}}\right] = \sum_m p_{in}(\lambda_m, T_C). \tag{6}$$

It follows that the characteristic time T_C can be computed by numerically inverting (6), which admits a single solution [8].

Our previous work [40] leveraged the intuition that the analysis of complex and large cache networks can be greatly simplified by replacing every LRU cache with a simpler Time-to-Live (TTL) cache [18, 13, 26, 27], where contents are evicted upon the expiration of a pre-configured *eviction timer* T'_C, which, for each content, is set upon the arrival of the last request if the content is not in the cache, and reset at every subsequent cache hit. As experimentally shown in [8], and remarked in [13, 25], the dynamics of a LRU cache with characteristic time T_C, fed by an IRM process with a catalog of cardinality M become indistinguishable from those of a TTL cache with deterministic eviction timer T'_C set equal to T_C (i.e., $T'_C = T_C$), and operating on the same catalog.

Yet, TTL caches are impractical since their size is not bound a priori, so that simulating TTL caches would require more memory than with classic LRU ones [41]. We thus proposed a technique to opportunely downscale the system to avoid the aforementioned biases, that we identified as *ModelGraft* [40]. As the main aim of this work is to parallelize its workflow, to make this paper self-contained we provide a high-level description of *ModelGraft*, referring the reader to [40, 41] for details, analytical proofs, and experimental results.

TABLE I
Scope of *ModelGraft* applicability

Workload	IRM, Dynamic [21]
Forwarding	SP, NRR [37], LoadBalance [36]
Cache decision	LCE, LCP [5], 2-LRU [25], CoA [4]
Cache replacement	LRU [8], FIFO [25], RND [14]

B. ModelGraft goals and scope

ModelGraft achieves very low memory and CPU complexity with respect to classic event-driven simulation, while at the same time yielding to performance that accurately represent those of the original system. Additionally, it retains simulation simplicity, as it is implemented and readily available as a simulation engine of ccnSim-0.4 [1]. Finally, it retains simulation flexibility, as it can be seamlessly applied to complex scenarios involving several schemes, summarized in Tab. I, that represent the current state of the art.

While in this work we limit our attention to networks of LRU caches, it is worth pointing out that *ModelGraft* can be applied whenever the original system admits a *mean-field approximation*, such as the one introduced by Che for LRU caches [8], and later extended to a fairly large class of cache networks [14, 25]. In particular, mean-field approximations exist for most *cache replacement strategies*, such as LRU, RANDOM, and FIFO, under both IRM and non-IRM traffic patterns with dynamic popularities [16] fit over real traces [42] (a theoretical justification is given in [21]). In addition, a fair variety of *cache decision policies* are supported, including classic Leave Copy Everywhere (LCE), Leave Copy Probabilistically (LCP) [5], k-LRU (whose behavior has been proven to converge to LFU [21]), and newer Cost-Aware (CoA) [4] and Latency-Aware Caching (LAC) [7] schemes. Finally, *forwarding strategies* are not limited to Shortest Path (SP), but state-of-the-art techniques including Load Balancing [36] and Nearest Replica Routing (NRR) [11], are equally supported over arbitrary graphs [34].

It follows that a *parallelization of the ModelGraft technique* would inherit the same wide scope of applicability of the underlaying simulation engine. On the one hand, a ccnSim [1] implementation would readily benefit the population of its users (at time of writing, ccnSim has been downloaded over 4,000 times). On the other hand, it would be beneficial for the technique to be generally applicable to other existing simulators, such as those overviewed in [39].

C. ModelGraft components and workflow

In a nutshell, *ModelGraft* combines elements of stochastic analysis within a simulative MonteCarlo approach of opportunely downscaled systems, where LRU caches are replaced

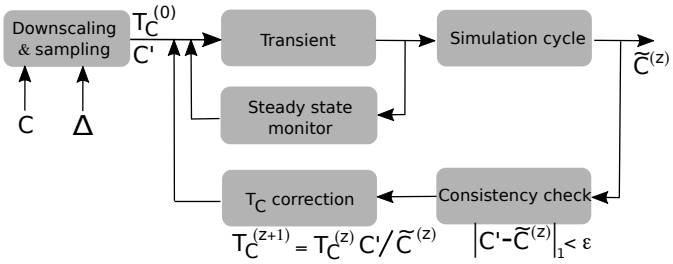


Fig. 2. Synoptic of *ModelGraft* workflow (simplified view of [40])

by their Che's approximated version, implemented in practice as TTL caches. Crucial ingredients are (i) the use of the so called characteristic time T_C of LRU caches as Time-To-Live parameter of their TTL counterparts; (ii) an effective downscaling of the catalog, which preserves crucial properties of its stationary probability distribution; (iii) the use of a rejection inversion sampling technique as random number generator to cope with memory explosion, and (iv) a self-regulation loop that guarantees accuracy by ensuring convergence to a consistent state. The *ModelGraft* workflow, as depicted in Fig.2, starts with a *downscaling and sampling* of the scenario, before entering a MonteCarlo TTL-based simulation phase. During the MonteCarlo TTL phase, statistics are computed after a transient period, where an *adaptive steady-state monitor* tracks 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. Once at steady-state, a MonteCarlo TTL *simulation cycle* performs a number of requests, downscaled by a factor of Δ. The monitored variable is then sent to the *self-stabilization* block: a *consistency check* decides whether to end the simulation, or to go through a T_C *correction* phase and start a new simulation cycle.

In particular, *ModelGraft* automatically infers the correct TTL parameter, i.e., T_C, by starting from uninformed guesses and automatically correcting the value according to the following observation. Despite TTL-based caches have not a fixed size, their average measured size $\tilde{C}^{(z)}$ at the end of the z-th step should be equal to the target downscaled cache $C' = C/\Delta$. As such, when all caches in the network are within a small error ϵ from their expected size, the results of the downscaled system are consistent with those that would have been gathered in the original one. Otherwise, *ModelGraft* exerts a controller action on the characteristic time of the next step $T_C^{(z+1)}$, to compensate for the observed cache size discrepancy, $T_C^{(z+1)} = T_C^{(z)} C'/\tilde{C}^{(z)}$. In practice, convergence requires only a handful of cycles even when the input T_C differ by orders of magnitude from the correct ones (see Sec. VII-B). Notice that cache size C does not need to be homogeneous across nodes. Finally, we may be wondering: up to which point can we enlarge safely the *downscaling factor* Δ, in our simulations? The answer is rather simple: indeed, convergence is guaranteed whenever the downscaled cache size can be reliably measured (e.g., $C' \approx 10$) so that a rule of thumb identifies the largest $\Delta \approx C/10$ as the most effective and robust setting. These properties make *ModelGraft* a hybrid simulation engine that can be seamlessly applied to any event-driven scenario.

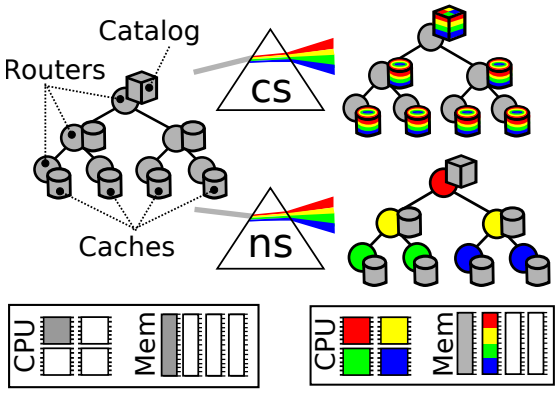


Fig. 3. Slicing alternatives for parallel simulations: Catalog Slicing (CS) vs Network Slicing (NS). Both CS and NS allow to use multiple CPU cores, at the price of a larger memory occupancy.

IV. Parallel Cache Network Simulation: High-level Design

In practice, the *ModelGraft* downscaling technique shifts the bottleneck for evaluation of large-scale scenarios from RAM memory to CPU execution time: the main contribution of this work is to break this CPU bottleneck by exploiting *parallelism* in the evaluation of cache networks. In doing so, we explore a broad design space, that we overview at high-level in this section and review in greater detail in Sec.V. Specifically, Sec.IV-A introduces two complementary strategies for parallelization, namely *Network Slicing* (NS) vs *Catalog Slicing* (CS): the latter technique yields a close-to-ideal multi-threading speedup, and is the main contribution of this work. Furthermore, Sec.IV-B introduces two strategies for the catalog downscaling and its mapping to multiple-cores, that respectively *spatially* or *temporally* split the request generation process: again, the latter technique yields to both an improvement in the results accuracy as well as a further speedup, and it is an additional contribution of this work.

A. Network vs Catalog Slicing

We illustrate these strategies with the help of Fig. 3, where we depict network nodes, caches, and content catalog. The left part symbolizes the *single-threaded* (ST) case, where the whole simulation is executed by a monolithic process, thus leaving the available resources (such as additional CPU cores and RAM memory) unexploited. The right part illustrates the *multi-threaded* (MT) strategies, with colors highlighting the primary resource to share over the different threads. As parallel execution increases the RAM usage, downscaling techniques are clearly required (Sec.IV-B).

NS Under *Network Slicing (NS)*, network topology is cut into multiple slices, whose simulation is assigned to different threads: thus, each thread simulates requests for the whole catalog over a portion of the network only. Two main factors

can however limit the efficacy of NS: first, the maximum number of multiple threads which can run in parallel is upper-bounded by the size of the network $N \ll M$. Second, and most importantly, the propagation of miss streams throughout the network (which influence the arrival processes at different nodes) translates into a strong correlation between neighboring caches. As such, the overhead tied to message passing is expected to grow with the *degree of parallelism* P (i.e., the number of parallel threads). For the sake of clarity, consider the simple hierarchical tree topology reported in Fig. 3: since leaves are independent, they can efficiently run on independent threads. However, already at the first level above the leaves, the request process becomes correlated with the incoming miss streams from the leaves: this forces the exchange of synchronization messages to keep consistency in the network state. As a consequence, we may expect NS gain to remain limited in practice.

CS Under *Catalog Slicing (CS)*, the catalog is cut into multiple slices, whose simulation is assigned to different threads: thus, each thread simulates a portion of requests over the whole network. The CS strategy is one of the main contributions of this work and leverages the following intuition. Recall that the *constant characteristic time* T_C of the early introduced stochastic model does not depend on the content itself: as individual contents become de facto independent in the approximation, we argue that it should be possible to shard requests for independent contents over multiple cores. Our proposal, complementary to the classical NS approach outlined above, is thus *to let each thread have a full view of the network (as nodes are correlated), but only a partial view of the catalog (as contents are independent).* In turn, given that contents are independent, we do not face additional overhead tied to synchronization of network state across threads, unlike in the NS gain, from which we expect sizeable multi-threading gains. To the best of our knowledge this intuition is unexplored (cfr. [17, 20] in Sec.VIII-B), despite it appears quite intuitive in hindsight.

B. Content slicing: Binning and mapping strategies

It is easy to recognize that CS technique alone would face a memory bottleneck. First, as the topology is entirely replicated for each thread, the amount of memory required to simulate the cache space C for each of the N nodes increases with the parallelism degree P. Second, the catalog $M \gg NC$ still represents the dominant factor for memory occupancy. Techniques to properly downscale the original scenario, as well as to map it to the multi-threaded execution are thus required: Fig.4 illustrates the *binning* and *mapping* strategies that we consider in this work and describe next.

PRE One option is to fix binning *a priori (PRE)*. Under a downscaled catalog of cardinality $M' = M/\Delta$, each meta-content $m \in [1, M']$ is requested with a rate $\bar{\lambda}'_m$, equal to the average request rate for contents in the correspondent bin $[(m-1)\Delta + 1, m\Delta]$ of the original catalog. This preserves the peculiarities of the original catalog, i.e., it guarantees

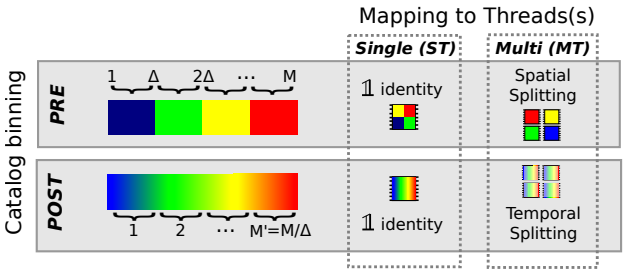


Fig. 4. Content slicing (CS): synopsis of Binning (PRE vs POST) and Mapping (ST vs MT) strategies. The original *ModelGraft* [40] technique can be identified as CS-PRE-ST in the terminology of this paper. Mapping strategies are tightly coupled to the binning in the multi-threaded MT execution: PRE binning *spatially splits* the catalog, whereas POST binning *temporally splits* the requests' process.

that ratios between the request rates of meta-contents exactly mimics the original ratios between aggregate rate of the corresponding bins.

Under PRE binning, a number of M' independent generators are needed in order to be mapped to one (ST) or more (MT) cores. In single-threaded simulation, all generators are mapped to the same core; the *ModelGraft* [40] technique, indeed, can be identified as CS-PRE-ST according to the notation of this paper. In the CS-PRE-MT multi-threaded execution, instead, each of the M' bins has to be assigned to one of the P threads. Practically, the downscaled request process R' is *spatially split* [20] into non-overlapping portions z_p of the downscaled catalog, each of which is *spatially mapped* to a different thread p, i.e.:

$$R' = \sum_{p=1}^{P} R'_p = \sum_{p=1}^{P} \sum_{i \in z_p} \bar{\lambda}'_i T_{end}, \tag{7}$$

where R'_p is the number of requests generated by assigning the z_p meta-contents to core p, the catalog is partitioned into M' disjoint sets $z_p \cap z_q = \emptyset,\ \forall p \neq q$, and T_{end} is the scheduled end of the entire simulation. Intuitively, to maximize the efficiency of the execution, it would be desirable to distributed the load among threads. Equivalently, the objective is to minimize the *makespan*, i.e., the total time after all cores have finished processing assigned jobs, which can be seen as a particular instance of the "multiprocessor scheduling" (or job shop scheduling with atomic instances):

$$\begin{aligned} &\min \max_p Y_p(z_p) \\ &\text{s.t.} \sum_{p=1}^{P} \sum_{i \in z_p} Y_p(z_p) = R' \\ &z_p \cap z_q = \emptyset,\ q \neq p \end{aligned} \tag{8}$$

where $Y_p(z_p) = \sum_{i \in z_p} \bar{\lambda}'_i T_{end}$ can be considered as the *load* of core p. We remark that the CS-PRE-MT mapping, which assigns each of the M' meta-contents to one thread to avoid overlapping, requires the preliminary solution of a NP-hard optimization problem – whose computational time, even for sub-optimal greedy heuristics, can grow large enough to offset parallel execution gains of CS-PRE-MT.

POST Another option is to infer binning *a posteriori (POST)*. In other words, the POST strategy relies on the inversion rejection sampling technique to build a single generator which extracts random numbers in the interval $[1, M]$ of the original non-downscaled catalog, following the original Zipf distribution with exponent α. Binning is done after the extraction of the random number m: the correspondent request rate λ_m is computed and associated to the correspondent meta-content m' according to the bin it falls into, i.e., $m' = \lfloor m/\Delta \rfloor + 1$.

The POST binning strategy can be applied in the single-threaded execution case: interestingly, the CS-POST-ST execution is not only more accurate, but it also achieves a $2\times$ speedup with respect to the CS-PRE-ST technique originally used in [40] (see Sec.VII).

The CS-POST-MT mapping strategy further leverages the fact that, due to Che's approximation [8], contents' dynamics can be considered both spatially as well as *temporarily* independent. As such, instead of *spatially/deterministically* splitting the request process as in CS-PRE-MT, under CS-POST-MT the request process is *temporally/randomly split* over multiple threads. Formalizing, we can re-write equation (7) as:

$$R' = \sum_{p=1}^{P} R^p = \sum_{p=1}^{P} \sum_{i=1}^{M'} \frac{\bar{\lambda}'_i T_{end}}{P}, \tag{9}$$

where the inner sum now extends over the whole downscaled catalog M': this clearly simplifies mapping, as now each thread can potentially generate requests for all the meta-contents, but along a limited amount of time (i.e., T_{end}/P). Temporal splitting [17] used by SC-POST-MT also entails load-balancing among threads, without incurring NP-hard complexity as in the spatial SC-PRE-MT case, from which we expect significant practical gains.

V. Parallel Cache Network Simulation: Implementation Details

In this section we introduce the components involved in the parallel simulation workflow, that we implement in the ccnSim [35], an open source cache network simulator [1] built over Omnet++ and written in C++. In particular, we use the Akaroa2 [9] software suite to parallelize the single-threaded workflow (early depicted in Fig.2) with a master-slave architecture (illustrated in Fig. 5 and described in the following). The for lack of space, the detailed software structure is omitted in this description but is available in GitHub [2].

A. Akaroa2

Akaroa2 [3, 9] is a licensed software written in C, offering all the fundamental master-slave network programming APIs. At high level, Akaroa2 assists in instantiating *multiple independent replications* of the same scenario on different cores: observations of the monitored KPIs are regularly sent to a central unit, i.e., the *master* thread, which is in charge of consolidating the statistics and eventually stopping the simulation when the desired level of accuracy has been reached. If P independent copies of the simulation are instantiated in parallel, measures are taken P times the rate of a single instance, thus potentially leading to a proportional speedup with the number of parallel cores.

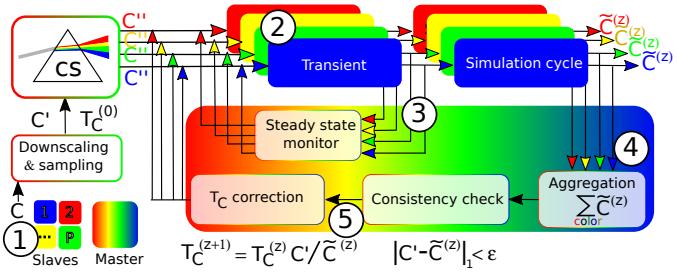


Fig. 5. Synoptic of parallel workflow: (1) Instantiation of parallel downscaled scenario; (2) Distributed collection of samples for centralized (3) steady-state monitoring (inner loop). Finally, (4) KPI consolidation and (5) optional T_C correction (outer loop).

There are three main components in Akaroa2. The *akmaster* process, which coordinates all the other processes, to which it is connected in a logical star topology. The *akslave* processes, used by the *akmaster* to launch and communicate with a *simulation engine*: there must be one *akslave* process on each host, and a single *akslave* process can launch and communicate with multiple *simulation engines* on the same host. The *akrun* process is used to instantiate a simulator run (i.e., selecting the engine, the scenario, the parallelism degree P), and to communicate with *akmaster* to choose and instruct the necessary *akslave* processes for that run.

In general, *akmaster* simply collects local estimates from the simulation engines, calculates global estimates, and decides when to stop the simulation. In our specific case, we needed to extend the Akaroa2 framework APIs to report a richer set of statistics (i.e., miss and hit vectors as opposite to just scalar values) and to integrate and centralize all the decisional logic (e.g., steady-state monitor, stability check, etc.) inside the master process. To let Akaroa fit our parallel simulation design, we modified 17 of its core component, that we release as patches at [2].

B. Parallel Workflow

Limiting our attention to the Content slicing (CS) technique, we now describe each block of the parallel workflow illustrated in Fig. 5. For the sake of clarity, each of the P worker threads (notice that there are $N_H < P$ *akslave* processes, where N_H is the number of physical hosts) is represented with a solid color, whereas the master process is represented with a blended color. With respect to the Network slicing (NS) technique, CS limits communication overhead both because (i) it is limited to slave and masters only, unlike in NS, but especially because (ii) communication is purposely designed to infrequently collect independent statistics from slaves for consolidation at the master, as opposite to frequently exchanging messages to synchronize simulation state as in NS. To simplify the exposition, in the following, we assume all caches in the network to be of the same size, but we remark that our approach naturally extends to the more general case.

① *Initial resource allocation*. Before executing the parallel simulation of the cache network, it is important to downscale the scenario and to dimension the master-slave infrastructure: this boils down to setting $\Delta \approx C/10$, selecting the binning and mapping strategies, choosing a parallelism degree P, and instantiating a number of slave processes on the desired hosts. In practice, given N_C (resp. $2N_C$) the number of physical cores on a given host (resp. virtual cores under hyper-threading), it is advisable to spawn *akslaves* processes on at least $\lceil P/N_C \rceil$ hosts.

② *Distributed Collection of Samples*. Once the parallel simulation starts, slaves start collecting statistics (counters of hit and miss events) that they periodically communicate to the master. At high level, the purpose of statistic collection is to test for steady state using batch-means for cache hit rate over a window of W samples. Since samples are collected from P independent instantiations of the same scenario, the individual window size (i.e., the number of samples collected from each slave) can be set as:

$$W_P = \lfloor W/P \rfloor, \ with \ W_P \geq 10, \tag{10}$$

where W is the desired window size for the single-threaded case, and where the lower bound of 10 samples is introduced to avoid collecting vanishing samples at high parallelism degrees. In particular, every τ simulated seconds, each slave checks for the availability of a new valid[2] sample for each node of the simulated network. When a large majority of nodes YN, with $Y \in [3/4, 1)$, have collected W_P valid samples, a summary vector containing $2N$ W_P samples is sent to the master (where the factor of 2 is due to the need of counting both hit and misses in the distributed case). The parameter Y is useful to avoid being slowed down from nodes whose cache is rarely accessed (e.g., as the root of a deep tree, or nodes that are in peripheral positions under arbitrary topologies), and that thus only minimally affect system performance. Once the summary vector is sent, the slave waits for a message from the master, that will either signal the start of the *steady-state* phase, or confirm the *transient* state.

③ *Centralized Steady-State Monitoring*. During transient state, the master collects distributed samples from all slaves: data collected *during the transient* phase are used to check whether steady state starting conditions are met (with a batch means technique). In the current implementation, the master operates in *synchronous mode*, meaning that it waits for the summary vector from all the simulation engines before continuing with the aggregate computation, leading to possible idle time at slaves. Yet, in our experiments (Sec.VI), we see that synchronous mode is efficient in setups with tens (to hundred) of cores, such as the one at our disposal. Another possible design, which we leave for future work, is the *asynchronous mode*, which could avoid idle times at slaves: however, clocks between workers may drift, and it could still lead to resource wastage (i.e., the events simulated between the last batch signaled by the slave and the steady-state signal by the master). This mode may be more efficient in the many-core regimes, such as Graphical Processing Units (GPUs) with (possibly several) thousands of cores.

Once all P vectors are received for the sampling interval j, all valid[3] samples collected from different slaves and related to the same nodes i are aggregated. Denoting with $hit_k(j,i)$ the number of hit events collected for node i during slot j by slave k, we have that the $h_P(j,i)$ sample for the hit ratio of node i aggregated over the P slaves during j is:

$$h_P(j,i) = \frac{\sum_{k=1}^{P} hit_k(j,i)}{\sum_{k=1}^{P} hit_k(j,i) + \sum_{k=1}^{P} miss_k(j,i)} \tag{11}$$

from which the master computes the average hit ratio $\bar{h}_P(i)$ and the corresponding coefficient of variation $CV_P(i)$:

$$CV_P(i) = \frac{\sqrt{\frac{1}{W_P - 1} \sum_{j=1}^{W_P} \left(h_P(j,i) - \bar{h}(i) \right)^2}}{\frac{1}{W_P} \sum_{j=1}^{W_P} \bar{h}_P(j,i)} \tag{12}$$

Denoting with ε_{CV} a user-defined threshold, the master considers the whole system to enter in steady-state when:

$$CV_P(i) \leq \varepsilon_{CV}, \qquad \forall i \in \mathcal{Y}, \tag{13}$$

where $|\mathcal{Y}| = \lceil YN \rceil$ is the *set of the first YN nodes satisfying the convergence condition*. In case condition (13) is not verified, the master signals all slaves to continue the transient phase (inner loop of Fig. 5). When instead (13) holds, slaves are instructed to enter the *steady-state phase*, where $R'_P = R'/P$ total requests are issued. Data collected *during the steady steady* are used to generate steady state measurements and check convergence conditions for the outer loop.

④ *Centralized KPI Consolidation*. At the end of a steady state phase (outer loop of Fig. 5), whereas hit rate is computed as described in step ③, consolidation of the cache size depends on the binning/mapping strategies. In particular, in the CS-PRE-MT case, each of the P slaves runs a different "spatial" portion of the catalog: the expected cache size for each slave is $1/P$-th of the cache size $C' = C/\Delta$ of the single threaded case. It follows that for CS-PRE-MT, the measured cache size of a node i in the TTL-based system is consolidated by *summing the individual contribution of the different slaves*, i.e., $C'_P(i) = \sum_{k=1}^{P} C'_P(i,k)$. Conversely, in the CS-POST-MT case, each of the P slaves runs a "temporal" slice of the whole catalog: the cache size is thus expected to be invariant with respect to the single-threaded case $C'_P(i) \approx C'$, so that the cache size is consolidated over the P slaves by *averaging the individual slave contributions*, i.e., $C'_P(i) = \frac{1}{P} \sum_{k=1}^{P} C'_P(i,k)$.

[2] A sample is valid when the cache has received a non-null number of requests since the last sample and its state has changed, i.e., a new content has been accepted in the cache and an old one evicted from the cache.

[3] As early stated, some "peripheral" nodes (i.e., outside the first YN nodes collecting valid samples), have not completed their collection: the sample is thus marked as invalid by the slave and disregarded by the master.

⑤ *Centralized Consistency Check and TTL Correction*. The per-node measured TTL-based cache size $C'_P(i)$ is used to assess the quality of the results: while the size of a TTL-based cache is not upper-bounded a priori, however it is expected that it will match that of the approximated LRU one, i.e., $C'_P(i) \approx C'$. Once all steady state measures are received, the master checks, for all nodes, the consistency of the measured and expected cache sizes:

$$\frac{1}{YNP}\sum_{k=1}^{P}\sum_{i\in\mathcal{Y}}\frac{\left|C' - C'_P(i)\right|}{C'} \leq \varepsilon_C, \tag{14}$$

where C' is supposed to be equal for all nodes without loss of generality, $\varepsilon_C$is a user-defined consistency threshold and, for coherence, measures are taken on those $|\mathcal{Y}| = YN$ nodes that have been marked as stable in the previous phase.

If condition (14) is verified, the master signals all the simulation engines to stop the simulation. Otherwise, discrepancies between measured $C'_P(i)$ and expected size C' are due to imprecisions in the value of the characteristic time T_C used as Time-To-Live in the simulation: given that the consolidated cache size is the same as in the single-threaded case, the master can then exert the same controller action than in the original ModelGraft [40] (as reported in Fig.5, the input T_C values are corrected proportionally to the ratio between the $C'/C'_P(i)$) and execute a new parallel simulation cycle (in which case, statistics are discarded and counter resest).

VI. Results

In this section, we evaluate the performance of the proposed parallelization techniques. We first describe the experimental settings and scenarios (Sec.VI-A). We then contrast the multi-threading speedup under Network vs Content slicing techniques (Sec.VI-B) and then the PRE vs POST binning and mapping strategies (Sec.VI-B), showing merits and limits of each technique. Finally, we contrast accuracy, memory and CPU complexity of the parallel techniques to event-driven and monte-carlo simulations (Sec.VI-D).

A. Experimental settings

As a reference, we consider a large-scale scenario, depicted in Fig.6, describing a Content Distribution Network (CDN)-like topology, where 4-level binary trees representing access topologies are interconnected through a backbone of 11 nodes arranged as the classical Abilene2 topology. The network comprises a total of $N = 67$ nodes, equipped with LCE-LRU caches able to store $C = 10^7$ objects out of a catalog $\mathcal{M}$ comprising M content objects, with $M \in \{10^9, 10^{10}, 10^{11}, 10^{12}\}$, whose popularity distribution follows a Zipf trend with exponent $\alpha = 1$ (unless otherwise stated). We defer the analysis of more complex (e.g., non LCE-LRU) scenarios to Sec.VII.

We run our experiments using two clusters: one where resources are *shared* (but always available) and another where resources are *dedicated* (but is subject to reservation). The *shared cluster* comprises $N_S = 5$ servers equipped with 48 GB of RAM memory[4] and 2× NUMA nodes with a Xeon E5-2665 CPU operating at 2.40GHz. As there are $N_C = 8$ physical cores per NUMA node, each server features 32 virtual cores in Hyper-threading available for parallelization. The *dedicated cluster* comprises $N_S = 4$ servers equipped with 378 GB of RAM memory and 2× NUMA nodes with a Xeon E5-2690 CPU operating at 2.60GHz (with $N_C = 12$ physical cores per NUMA node, thus 48 virtual cores per server in Hyper-threading). To gather conservative results, we mostly use the shared cluster that operates at a lower frequency, and defer results of the dedicated cluster to Sec.VII.

[4]In reason of memory limits, the largest scenario attainable with event-driven simulation has a catalog of $M = 10^9$ objects

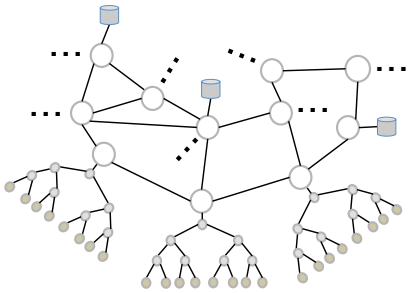


Fig. 6. CDN-like scenario: 4-level binary trees representing access topologies are interconnected through an Abilene2 backbone of 11 nodes, comprising a total of $N = 67$ nodes. LCE-LRU caches are able to store $C = 10^7$ objects out of a catalog comprising M content objects, with $M \in \{10^9, 10^{10}, 10^{11}, 10^{12}\}$.

As for the shared cluster, we cannot guarantee that our experiments run in isolation. To lessen the impact of uncontrolled workloads operating on the same server at the same time, we (i) poll each server load and select those that are instantaneously the least loaded, we (ii) repeat experiments 25 times, and (iii) measure the *minimum* execution time across repetitions. At the same time, since the only impact of the additional workload is to limit the achieved speedup, it follows that results we report in the following are conservative.

B. Network Slicing (NS) vs Content Slicing (CS): Quantitative Comparison

To assess the scaling properties of NS against CS techniques, we start by observing the *relative speedup* that a degree of parallelism P brings to each technique with respect to their monolithic single-threaded version (i.e., original ModelGraft [40]). Notice that we are not comparing, for the time being, against *event-driven simulation*, which is orders of magnitude slower. Formally, denoting with $T(P)$ the execution time of a simulation with P parallel threads running independently, we define the *relative speedup* as $S(P) = T(P)/T(1)$. For the sake of completeness, we report that for the scenario under consideration (i.e., CDN-like and $M = 10^{11}$) $T(1) = 4.82$ hours. Fig. 7 reports the speedup as a function of the degree of parallelism P for the NS vs CS techniques. The top x-axis additionally reports the number of servers N_S and cores N_C involved for any given P.

NS Considering the Network Slicing case first, two remarks emerge from Fig. 7: first, for low parallelism (i.e., when the topology is split over $P \in [2, 3]$ parts, assigned to independent

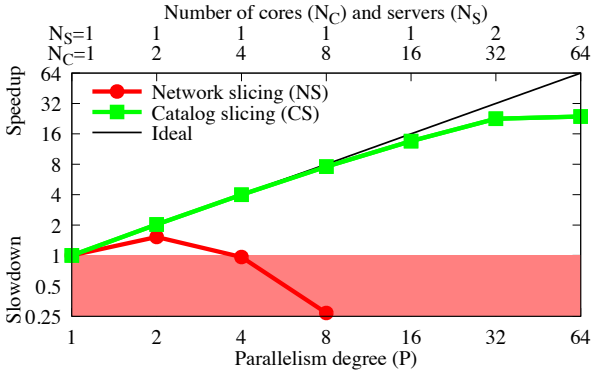


Fig. 7. Comparison of Network Slicing (NS) vs Catalog Slicing (CS): speedup (or slowdown) with respect to the single-core case, as a function of the degree of parallelism P (i.e., the number of slave threads). The number of servers N_S and cores N_C is reported on the top x-axis. NS incur a *significant slowdown* due to MPI overhead, already in single-server settings, when $P > 4$. Instead, CS *speedup is ideal* even with multiple servers (i.e., $P = 32$ and $N_S = 2$), and flattens afterwards ($P = 64$ and $N_S = 3$).

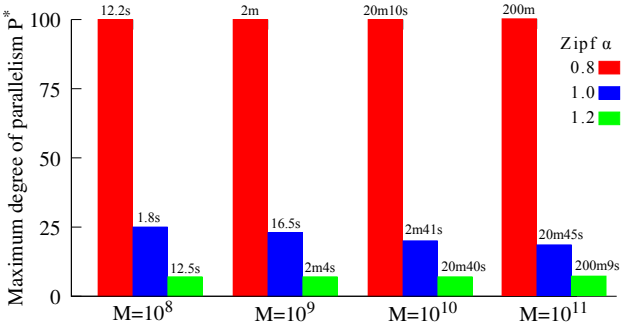


Fig. 8. CS-PRE-MT: maximum parallelism degree P^* for different sizes of the catalog M and Zipf's exponents. Execution time of the greedy solver are also annotated.

parallel threads), NS yield to a noticeable speedup, that however quickly degrades and vanishes already for $P = 4$. Indeed, while for simple topologies (e.g., a tree), slicing can achieve some benefit for low parallelism (e.g., right-half and left-half sub-trees), however more complex topologies and arbitrary routing protocols makes it hard to slice the network in practice. Furthermore, for higher parallelism, i.e., $P > 4$, network slicing entails a *significant slowdown*, already in single-server settings: notice, indeed, that when $P = 8$ simulations are $4\times$ slower than their single-threaded counterpart, and thus $32\times$ slower than in the case of an ideal speedup (we do not carry out experiments for $P > 8$, which would further share the load over multiple machines). Slowdown happens due to the correlation between neighboring caches: we also carried out experiments varying the MPI *laziness*[5], that correlates with the frequency at which synchronization messages are exchanged between different logical processes, with no observable benefit. As it seems that the overhead of using MPI quickly offsets any benefits tied to parallel execution, this makes NS of limited practical appeal.

CS Consider next the Catalog Slicing case, in its CS-POST-MT configuration (i.e., with a single generator whose requests are randomly split over time). Again, two observations can be gathered from Fig. 7: remarkably, an *ideal speedup* is not only achieved in single-server setting ($P = 16$), but it also persists even with multiple-servers ($N_S = 2$ or $P = 32$). Second, while in the range $P \in [32, 64]$ the speedup deviates from the ideal reference, it is important to notice that the speedup still grows, albeit slowly, in that range (which is hard to appreciate in the log-log scale). Several factors possibly contribute to the deviation from ideal speedup, notably: (i) external workload, that is more likely to be encountered when the number of servers increases; (ii) synchronous mode of operation at the master; (iii) growing impact of transient, statistic collection, output procedures for large P. We will come back on (iii) in Sec.VI-C, and dwelve (i),(ii) in Sec. VII-A

[5]Specifically, the laziness parameter in Omnet++ MPI is a synchronization rate $\lambda \in [0, 1]$, with maximum (minimum) synchronization rate achieved for λ equal to 1 (0).

C. Catalog Slicing: PRE vs POST binning and mapping

As early illustrated, two strategies allow to downscale the original catalog, and map it to a multi-thread scenario: an a priori binning strategy (PRE) does so by a spatial splitting of the objects that each thread simulates, whereas an a posteriori binning strategy (POST) allow to temporally split the request process over multiple threads.

PRE The idea of the request splitting approach is to distribute the request process among the several threads running in parallel, in such a way that each one would generate an equal amount of total requests (i.e., R^p, with $p \in [1, P]$) for non-overlapping meta-contents. This requires the execution of an optimization problem, with the goal of finding the optimum partition of the downscaled catalog that maximizes the number of *efficient* parallel threads P^*, i.e., the maximum parallelism P^* such that splitting the simulation among $P > P^*$ threads would not bring any additional benefit, which is due to the skew in object popularity. Intuitively, under Zipf popularity skew (and $\alpha = 1$ for simplicity) in the limit case $P \to \infty$ where each content could be split into a different bin, then request process associated to the bin comprising the most popular object (having rank 1) would generate two (three, etc.) times more request than the bin with rank 2 (three, etc.) object. In other words, due to skew in the object popularity, a skew in the per-thread workload persists, and the whole simulation execution is bottlenecked by the slowest thread. More in details, in the PRE binning strategy, meta-contents follow a Zipf distribution with decreasing popularity (i.e., $\lambda_i > \lambda_j$, for $i < j$): the first meta-content represents an aggregate of Δ objects, whose request rate λ'_1 is considerably higher than the one associated with the last meta-content M', corresponding to the last batch of $[M - \Delta, M]$ unpopular objects. Hence, since requests for the first meta-content cannot be split further and assigned to different threads, λ'_1 caps the maximum number of efficient parallel threads P^*, in the sense that any additional core with $\lambda'_x < \lambda'_1$ would have a

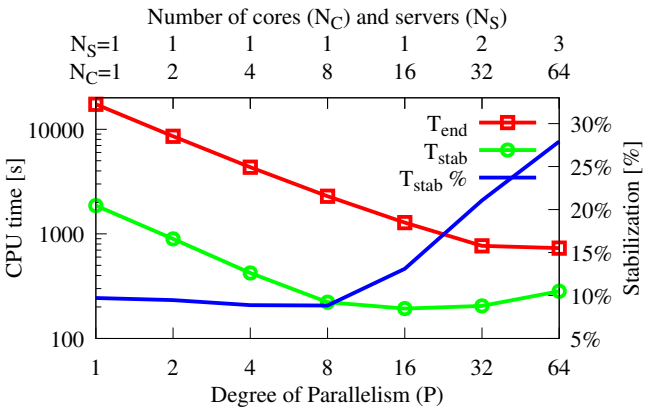


Fig. 9. CS-POST-MT: T_{end} total execution time, T_{stab} stabilization time, and T_{stab}/T_{end} relative percentage, for different degrees of parallelism P, N_C, and N_S configurations. CDN-like scenario with $M = 10^{11}$.

shorter individual execution time, without however lowering the global execution time dominated by the first meta-content.

To understand the maximum parallelism under CS-PRE-MT, we solve the optimization problem with a greedy heuristic approach. Specifically, Fig.8 reports the maximum number of useful threads $P^\star$, for different catalog cardinalities M and popularity skews α, as output of the optimization problem, when requesting $P = 100$ parallel threads as input (which is limited by the number of cores at our disposal). As expected, when content popularity skew is low (i.e., $\alpha = 0.8$), the corresponding per-thread workload is more uniform and the maximum number of efficient parallel threads P^* is not capped by λ_1' (since $P^* = P$). However, using popularity skews that are typically found in the CDN/ICN literature, the maximum number of useful threads P^* is possibly limited (to about $P^\star \approx 20$ threads when $\alpha = 1$) or even dramatically reduced ($P^\star = 7$ when $\alpha = 1.2$). Notice, also, the decreasing trend of $P^\star$ (e.g., from $P^\star = 25$ at $M = 10^8$, to $P^\star = 20$ at $M = 10^{11}$) as the original content catalog M grows; this results from the fact that the head of the catalog becomes more important as M grows (recall Sec.II). Notice, finally, that the choice of P^* and the mapping process itself has a sizeable cost: e.g, the running time of the greedy solver for $M = 10^{11}$ can grow up to 200min, and we interrupted the solution for $M = 10^{12}$ after one *day* of CPU time, as this could end up dominating the parallel simulation duration.

Overall, it seems that while the PRE binning strategy plays well in a single-threaded environment (CS-PRE-ST), and was indeed used in ModelGraft [40], however its extension to a multi-threaded environment faces a number of unexpected complications. Notably, the expected speedup widely varies with the popularity skew (7×-100×), and slightly diminishing for increasing catalog sizes The allocation of workload to the different threads possibly requires significant computation time, especially for large catalogs. It seems thus that a parallelization of the CS-PRE-MT technique can be scenario-dependent and yield to fragile gains, which thus limits its appeal from a practical viewpoint.

POST The POST binning strategy slices request over time, mapping them with a random temporal splitting among threads. This achieves two immediate advantages over PRE: first, it is not scenario dependent, and removes the need to solve a preliminary optimization process to instantiate the most efficient configuration to be used for the parallel evaluation of a cache network; second, random temporal splitting makes mapping very simple and also tends to equalize the load among multiple cores. In particular, since all the meta-contents can be requested by any thread during each independent realization of the same simulation, load distribution is just done by sharing the total number of requests $R^P = \lceil R/P \rceil$ (and so the approximated end time) according to the number of parallel threads P.

While this does not impose an a priori capping on P, as in the PRE case, however an ideal speedup is not always guaranteed as P grows high, even for POST, as Fig.7 from Sec. VI-B illustrates. Here, we report complementary details of the same evaluation presented in Fig.7; in particular, Fig.9 depicts the total execution time of the simulation (T_{end}, red squares) as well as the stabilization time (T_{stab}, green circles), i.e., the time it takes the master process to state the end of the transient phase. Additionally, the picture reports the relative importance of the stabilization time over the whole simulation duration (T_{stab}/T_{end}, blue line). It can be seen that both T_{end} and T_{stab} decrease linearly in the number of threads (notice the log-log scale) up to $P = 8$; already for $P = 16$, there is a noticeable change of slope in the T_{stab} curve, whereas the linear decrease continues for T_{end} up to $P = 32$: specifically, T_{stab} still decreases, albeit slowly, up to $P = 16$, where there is a change in the trend and it starts to *increases* for $P > 16$.

As highlighted in Sec.VI-C, this is due to multiple factors, like (i) a higher probability of sampling busy machines when increasing P (i.e., part of the CPU can be consumed by other tasks), or (ii) the de-synchronization among the P threads that send hit and miss sample in different time instants, thus progressively delaying the centralized computation of the steady-state done by the master. Hence, the relative importance of the stabilization (which accounts for roughly 10% of the whole T_{end} up to $P = 8$), remarkably grows (to account for 30% of the simulation duration for $P = 64$), thus limiting the global speedup with respect to the single-threaded version of the same scenario.

D. Comparison at a glance

We finally report results comparing all techniques at a glance in Fig. 10. We now include ① Event-Driven (ED) simulation strategy, the ② single-threaded CS-PRE-ST technique originally introduced by ModelGraft [40] and ③ the best performing among the different parallelization techniques explored in this work, namely CS-POST-MT.

We stress that not all techniques can scale to the same scenario size. Particularly, Event-Driven (ED) simulation faces a RAM memory bottleneck: ED techniques can scale only up to $M \leq 10^8$ when rejection inversion sampling is not used (ccnSim-v0.3), and up to $M \leq 10^9$ when rejection inversion sampling is used (ccnSim-v0.4). It follows that $M = 10^9$ is

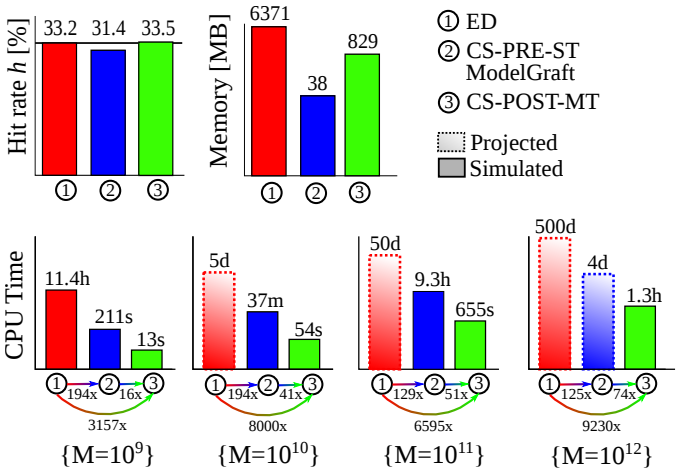


Fig. 10. Results at a glance: Accuracy and memory requirements for the CDN scenario with $M = 10^9$ (top) and simulation duration for $M \in [10^9, 10^{12}]$ (bottom) for Event Driven (ED), the original ModelGraft [40] (CS-PRE-ST) and the best performing multi-threaded alternative (CS-POST-MT). Gains, annotated over the arrows in the bottom part of the figure, increase with increasing scenario size.

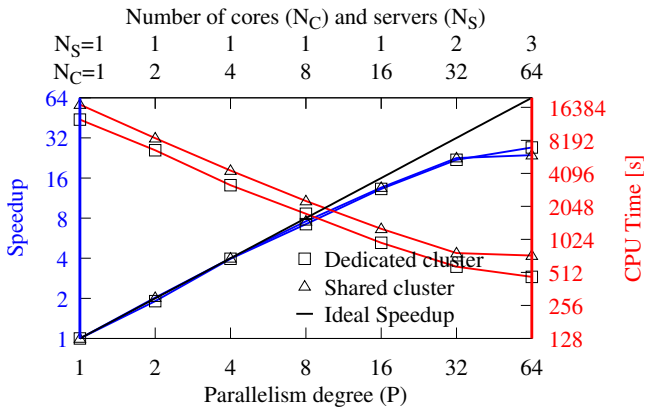


Fig. 11. Shared vs Dedicated Cluster: relative speedup (blu lines) and absolute execution time (red lines) for different degree of parallelism (P). CDN-like scenario with $M = 10^{11}$.

the largest scenario where we can directly contrast all relevant key performance indicators on the execution of the very same scenario with different techniques. In particular, results related to both *average hit rate* and *memory usage* in the CDN-like scenario for $M = 10^9$ are reported in the top part of Fig. 10. The total *CPU time* is shown in the leftmost plot of the bottom part of Fig. 10 for $M \geq 10^9$.

Scenario $M = 10^9$ is instrumental to appreciate hit rate accuracy and memory performance. Conversely, such a "small"-scale scenario does limit the achievable gain in CPU time[6]. We remark that hit rates measured in the downscaled CS-PRE-ST and in the parallel CS-POST-MT evaluation are close to statistics gathered with ED. We will analyze accuracy at a greater detail in Sec. VII-C and Sec. VII-B. In terms of memory requirements, while ED requires nearly 7GB of RAM, CS-PRE-ST just needs 32MB (i.e., 1/167 of ①) which is due to both downscaling and inversion rejection sampling. RAM usage increases in CS-POST-MT to 829MB due to the replication of the downscaled catalog for each thread. However, this is not of a concern given that (i) this is still about one order of magnitude less than ED, (ii) it is not expected to increase for increasing catalogs, since the downscaling factor Δ increases as well (indeed, memory occupancy in the $M = 10^{12}$ case tops to nearly 2GB, that is furthermore shared over multiple servers).

The bottom portion of Fig. 10 reports simulation duration for growing catalog sizes. While for CS-PRE-ST ② memory is no longer a bottleneck, CPU time can still grow arbitrarily large due to single-threading: without loss of generality, we thus set a threshold of 1 day of CPU time per instance: i.e., single-threaded executions on scenarios whose CPU times are expected to be longer than 1 day are only projected[7]: projected durations are represented by a bar dashed margins and gradient fill in Fig. 10. For CS-POST-MT ③, results are obtained using only minimal server resources[8]: as expected, the relative speedup brought by multi-threading grows with the size of the scenario, topping when $M = 10^{12}$ to a gain of nearly *2 orders of magnitude* over ② and *4 orders of magnitude* over event driven simulation ①. Hence, with current common-off-the-shelf hardware, simulating cache networks at a scale $M = 10^{12}$ is attainable only with multi-threading, and in particular only with the CS-POST-MT technique as the others fail (NS or even CS-PRE-MT). To put it otherwise, given a memory limit and time budget, techniques such as CS-POST-MT enables the study of scenarios that are orders of magnitude larger than those attainable by any known technique. This allows to study realistic scenarios, getting rid of the over- and under-estimation errors due to naïve downscaling shown early in Sec.II.

VII. Sensitivity Analysis

In this section we assess the robustness of the introduced technique, by assessing performance variability across across different *hardware infrastructures* (Sec. VII-A), verifying that the *self-stabilization* property holds on multi-threaded simulation (Sec. VII-B) and finally evaluating the speedup and accuracy under more *complex scenarios*, including non-LRU caches (Sec. VII-C). Further scenarios corresponding to 340 parameters combinations, for cumulated 31 days of CPU time are available in an extended technical report [38].

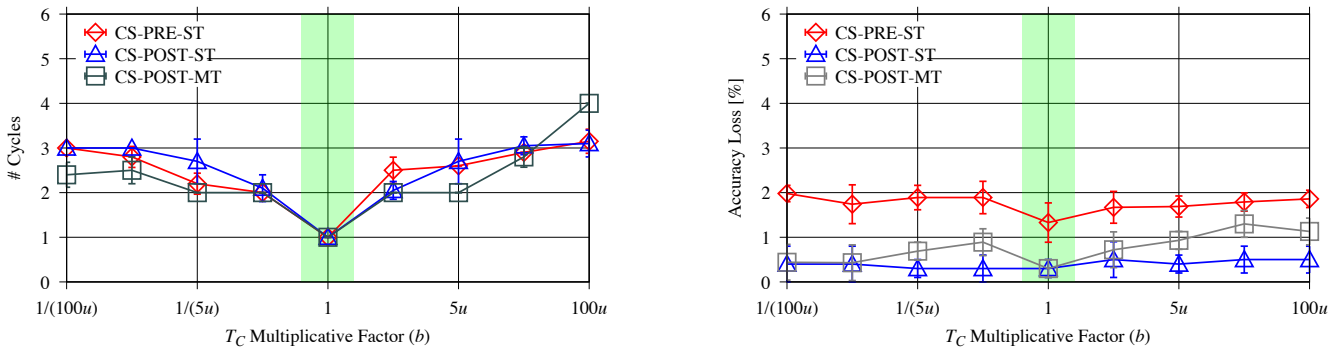
A. Shared vs Dedicated Hardware Infrastructure

While all the results presented so far have been collected from simulations carried out in a *shared* cluster, in this section

[6]Note that CS-POST-MT completes *in just 13 sec* with only $P = 16$ threads (instead of 11.4 hours as in ED), a sizeable part of which is spent in executable loading, bootstrapping and IO (logs and statistics)

[7]Notice that, given that the number of requests in the steady state is known (user-defined), and the event processing rate per scenario is known (by empirical observation), the expected simulation time of ED and ModelGraft can be accurately estimated [41].

[8]Specifically, only a single server $N_S = 1$ with $P = 16$ for $M = 10^9$, two servers $N_S = 2$ with $P = 32$ for $M = 10^{10}$, and all the three servers in the shared cluster with $P = 64$ for $M \geq 10^{11}$.



(a) Number of Cycles

(b) Accuracy Loss

Fig. 12. T_C Sensitivity: comparison of POST vs PRE binning in both ST and MT configurations.

we add a *dedicated* cluster as further term of comparison. These two infrastructures exhibit a clear tradeoff. On the one hand, computing resources are always available in the *shared* cluster, which simplifies the scheduling of experiments; however, as there is no resource isolation, exogenous workload can perturb the execution of our simulations. On the other hand, the *dedicated* cluster guarantees complete isolation of the allocated resources, but it introduces an additional burden tied to resource reservation, which could impact the whole simulation campaign agenda.

Hardware characteristics of the dedicated cluster have been thoroughly listed in Sec. VI-A, but it is interesting to point out that CPUs in the dedicated cluster runs at a higher frequency than in the shared one, and that are additionally equipped with a larger L1/L2/L3 cache memory. In reason of the higher-profile specs, and of the absence of exogenous workload, we thus expect the dedicated cluster to outperform the shared one.

Fig. 11 reports both the *relative speedup* (blue lines, left y-axis) with respect to the single-threaded execution of each scenario, i.e., CS-POST-MT/CS-POST-ST on either a shared or a dedicated cluster, and the *absolute execution time* of CS-POST-MT (red lines, right y axis). In particular, we simulate the CDN-like scenario described in Sec. VI-A with a catalog of $M = 10^{11}$ contents downscaled by $\Delta = 10^5$. In terms of relative speedup, Fig. 11 confirms for the dedicated cluster the trend early shown in Fig. 7: ideal speedup up to $P = 32$, and slower increase afterwards. As the trends is practically indistinguishable, this also confirms the growing relevance of the stabilization time in the total execution time. Especially, it identifies in the *synchronous mode* (as opposite to exogenous traffic, which is absent by design[9] in the dedicated cluster) the first performance bottleneck at high parallelism degrees $P \geq 64$.

Notice also that differences in the absolute execution times (red lines, right y-axis) are simply due to the different performance between the two hardware sets (by about a factor of 1.5×). It hence follows that *absolute simulation duration* values reported in this paper are rather conservative.

[9]We remark that since performance trends are very similar, it also follows that using a well designed script to probe the load on the shared servers and allocate slaves on the least loaded ones, helped to minimize possible performance degradations related to exogenous loads in the shared cluster.

B. Self-stabilization

Self-stabilization is a key property to let the methodology have a practical relevance: as the T_C values of a complex system are not easily guessed, the self-stabilization property makes the technique resilient against wrong T_C values provided as input. However, self-stabilization has been proven only for the single-threaded CS-PRE-ST configuration [40]. The aim of this section is to ensure this property to hold also for different configurations proposed in this paper. Specifically, we aim at assessing both the impact of the newly proposed binning and mapping strategy (i.e., POST instead of PRE) as well as the impact of distributed measurement at slaves (i.e., MT instead of ST) and their centralization into a *consistency check* at the master (point ⑤ of Sec. V-B).

In particular, we stress the self-stabilization property by purposely introducing controlled errors in the input T_C values: $T_C^0(i) = b(i)T_C^{sim}(i)$, where $T_C^0(i)$ is the initial characteristic time for node i, $T_C^{sim}(i)$ is the accurate value (gathered via event-driven simulation), and $b(i) \in [1/(Bu), Bu]$ is a multiplicative factor obtained by multiplying a controlled 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* (when $b(i) > 1$) and *underestimating* ($b(i) < 1$) the actual $T_C^{sim}(i)$ value. 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*, varying furthermore from node to node due to the uniform random variable u.

Fig. 12 summarizes, as a function of the input error magnitude (i.e., bias $b(i)$), results obtained from CS-PRE-ST (i.e., the original ModelGraft [40]) against CS-POST-ST (i.e., the new binning strategy proposed in this work, still applied in a single-threaded mode) and CS-POST-MT (i.e., full-flavored multi-threading evaluation). In particular, Fig. 12(a) reports the *average number of cycles* (with 95% confidence intervals computed over 25 repetitions) needed to converge to a consistent state and stop the simulation. Fig. 12(b) instead reports the *average accuracy loss* with respect to the event-driven results (again with 95% CI).

Observing the number of cycles in Fig. 12(a), one gathers that (i) as expected, the number of cycles equal to 1 when the input $T_C^0(i) \approx T_C^{sim}(i)$, which happens for $b \approx 1$. At the same time, notice that (ii) the number of cycles is generally very small (i.e., below 3) even for widely wrong input $T_C^0(i)$ values. This confirms the methodology to converge fast under all explored settings (i.e., CS-PRE-ST, CS-POST-ST, CS-POST-MT) with only a very limited impact across different binning and multi-threading options. Observing Fig. 12(b), one further gathers that (iii) not only the simulation converge fast, but also converges to an accurate value, as the error is bound below 2% across all CS-PRE-ST, CS-POST-ST, CS-POST-MT strategies. Finally, (iv) the newly introduced CS-POST-ST and CS-POST-MT strategies are even more accurate than CS-PRE-ST (i.e., accuracy improves with respect to ModelGraft [40]).

In general, one can conclude that the combination of the POST binning with the MT mapping preserves the self-stabilization property: regardless of the level of parallelism, the number of cycles always remain bounded to a small integer value, for an accuracy loss lower than 1% in expectation.

C. Accuracy under complex scenarios

We finally assess the accuracy of the newly proposed POST binning in more details. While POST is crucial to achieve ideal speedup in MT settings, we also remark that it additionally yields to a *higher accuracy* than the PRE binning. It is however interesting to assess properties of the POST binning independently of its MT application, and to particularly select other more complex strategies than simple LRU caching, to assert whether accuracy improvements still hold.

Specifically, we consider a network comprising caches operated with different decision policies. In particular, we consider (i) Leave a Copy Everywhere (LCE) where all new data is admitted in the cache forcing the eviction of an old copy, a simple (ii) Leave a Copy Probabilistically (LCP) strategy where new data is admitted in the cache with probability $p = 1/10$ and a significantly more complex (iii) a 2-LRU strategy where the admission process happens in two phases: only in case of a "name" hit in the first cache, the corresponding "data" is cached in the second cache. We remark that this choice is of particular relevance, since LCP is used in practice [5] and is the basis for a richer family of probabilistic approaches [4], and since 2-LRU, whose behavior has been proven to converge to LFU in the case of stationary catalog [21], is furthermore the best choice in non-stationary cases [16].

We instantiate the large-scale scenario of [40], i.e., 4-level binary tree, with $M = 10^9$ contents, $C = 10^6$ cache size, downscaling $\Delta = 10^5$, popularity skew $\alpha = 1$. By simulating this scenario, we can not only compare the performance of both CS-PRE-ST (i.e., the original ModelGraft [40]) and CS-POST-ST, but also include performance gathered with a classic Event-Driven (ED) simulation approach.

Results are summarized in Tab. II: it clearly appears that, despite the already satisfactory accuracy of the CS-PRE-ST strategy (i.e., losses smaller under 2%), the new CS-POST-ST technique is (i) significantly *faster* (i.e., almost 2×), and (ii) remarkably *more accurate* (i.e., accuracy loss significantly reduce and tops to 0.6% in the worst case) than CS-PRE-ST.

TABLE II
EVENT-DRIVEN SIMULATION VS CS-PRE-ST VS CS-POST-ST: ACCURACY LOSS, CPU AND MEMORY GAIN

Cache Decision Policy	Technique	p_{hit}	Loss	CPU time	Gain	Mem [MB]	Gain
LCE	CS-PRE-ST	**31.4%**	**1.8%**	**211s**	**194x**	**38**	**168x**
	Event-driven	33.2%		11.4h		6371	
	CS-POST-ST	**33.1%**	**0.1%**	**143s**	**286x**	**23**	**277x**
LCP(1/10)	CS-PRE-ST	**34.0%**	**1.4%**	**291**	**90x**	**38**	**168x**
	Event-driven	35.4%		7.3h		6404	
	CS-POST-ST	**35.3%**	**0.1%**	**171s**	**153x**	**23**	**277x**
2-LRU	CS-PRE-ST	**36.1%**	**0.9%**	**402s**	**97x**	**38**	**234x**
	Event-driven	37.0%		10.8h		8894	
	CS-POST-ST	**37.6%**	**0.6%**	**180s**	**216x**	**24**	**370x**

We believe that, while not the main focus of this work, this result represent a contribution on its own. Notice indeed that, albeit a 2× improvement may seem quite small compared to an ideal speedup of $P\times$, nevertheless this improvement holds for single-threaded execution as well – and a factor of 2× improvement is an engineering accomplishment on its own. Additionally, the speedup comes with a significant increase in the accuracy – a not-so-common win-win situation.

VIII. RELATED WORK

Discrete-Event Simulation (DES) is an important methodology for the study of complex systems such as networked protocols: however, classic DES approaches face massive computational and time requirements, which makes their usage prohibitive in large-scale scenarios. Valuable work aiming at overcoming these limits, have either attempted at *parallelizing* execution of packet-level simulation or using *hybrid modeling* techniques to alter the granularity of the simulation. Clearly, the area of *cache networks modeling* is also relevant.

A. Parallel discrete-event simulation

A first approach, known as Parallel Discrete-Event Simulation (PDES) [15], focuses on *parallelization* as a method to speedup execution time: the simulation model is divided into sub models, called Logical Processes (LPs), which can be distributed among processors/cores. The whole instruction set is split among LPs, which execute their own event-list following their own simulation clock; however, they are limited by the so called *causality constraint* (i.e., instructions need to be executed following a timestamp order), which requires the exchange of explicit synchronization messages among LPs, thus often offsetting benefits coming from parallelization. This is true not only for cache networks (where the correlation among neighboring caches makes the overhead of message passing offsetting any benefit in the NS strategy), but for every

scenario that presents challenging features (e.g., reaching a 50% parallel efficiency for a very large-scale scenario with thousands of high-speed links would be possible only if using supercomputers with million processors [22]).

In this context, several work have tackled the study of caches, although from the perspective of a computer architecture: numerous existing techniques are well covered in [44]. In this space, our work shares similarity with seminal work such as [17, 20]: more specifically, [17] introduces parallel cache simulation via time partitioning, and [20] further adds horizontal slicing along the orthogonal content dimension. From a high-level viewpoint, the PRE and POST techniques introduced in this work can be traced back respectively to [20] and [17]. However, there are several important differences between [17, 20] and this work: first, computer architecture work limitedly focus on single LRU cache, or a very small cache hierarchy (e.g., L1 and L2 caches). Our work is fit for complex network of caches with arbitrary topologies, routing algorithms and meta-caching algorithms (recall Tab.I). Second, the workload are extremely different: furthermore, by exploiting content dynamics decoupling induced by Che's approximation, our work introduces innovative downscaling techniques, that –relatively simple in the hindsight– were not previously introduced to the best of our knowledge.

B. Hybrid modeling techniques

A complementary approach leverages *analytical models* [31, 24, 32, 23, 40] to abstract the fine-grained packed level simulation dynamics into a coarse grained (and possibly downscaled) *hybrid network models* with improved performances with respect to classic DES. For instance, fluid models for TCP/IP networks [31] are used to either feed a suitable scaled version of the system with a sample of the input traffic [32], or to represent background traffic as fluid flows [24, 23].

Fewer work combines multi-resolution and parallelization techniques: for instance, [28] implements the model of [31] in a *parallel hybrid network simulator* offloading computationally intensive background fluid-based traffic calculations to Graphic Processing Units (GPUs), which offer better performance (i.e., $10\times$) and parallelization with respect to normal CPUs. Despite what we propose in this paper shares the spirit of [28] in integrating hybrid simulation with parallelization, it differs in the (i) target (i.e., cache networks), (ii) approach (i.e., downscaling), and (iii) parallelization technique (i.e., catalog slicing), which jointly enable the design of a lean and efficient simulator for networks of caches.

C. Cache networks models

The above hybrid techniques however targets very different application domain than network of caches: with this respect, close work worth referencing is [8, 19, 33, 12, 37, 26].

Particularly, approximate [8] and exact [19] models for cache asymptotics are relevant. Interestingly, we stress that Fagin [10] already published in 1977 the results (on "computer" caches) independently found by Che [8] in 2002 (on "network" caches) that this work build upon and extends to general cache networks.

Simple models of general cache networks are also given in [33] and [37] for Shortest Path and Nearest Replica Routing respectively. However, as we show in [37], the accuracy of these models degrades significantly as we move to topological regions faraways from the leaf nodes –as they resort on independence assumptions among the states of caches, and rough characterizations of the request processes at non-leaves caches– this does not happen with the new technique we propose, which is able to correctly represent the request processes at every cache while capturing existing correlations. Moreover we would like to remark that differently from previous models, which are limited to the analysis of steady state regime, our new technique also allow to study transient behavior, make links fall down, change the routing, add and remove nodes.

Finally, TTL caches are studied in depth by [12, 26], which this work also heavily leverages. However, as their size is not fixed a priori (unlike LRU caches), simulating TTL caches may become impractical (as their simulation requires more memory [41]). Thus, downscaling techniques such as the one proposed on this (CS-POST) our own previous work (CS-PRE, referred to ModelGraft in [40]) are highly relevant.

Finally, worth mentioning is the space of cache network simulators: we point out that a direct comparison with existing simulators have been presented in [39], from which it emerges that ccnSim is among the fastest and the most scalable ones. Given the generality of the technique proposed in this work, we hope that releasing its description and code can simplify its porting to other tools such as those compared in [39].

IX. Conclusion

In this work, we question whether it is possible to efficiently and accurately perform parallel simulation of general cache networks. We find that while the question has a positive answer, however a design that yields to ideal speedup is not easy to find. This paper carries on a broad exploration of the design space, and reports on detailed performance evaluation from implementation of different strategies in this space.

Our first and foremost contribution is the introduction of a Catalog slicing (CS) technique, which is essential to circumvent the (huge) MPI overhead suffered by classic Network slicing (NS) approaches, which is due to correlation among neighboring caches and that render NS useless from practical purposes. A second contribution is the exploration of different combinations of binning strategies for the catalog downsizing (PRE vs POST), that are either based on an a priori partitioning of the catalog along the spatial dimension, or on a posteriori splitting of the request process along the time dimension. These binning strategies naturally yield to different mapping strategies for the multi-threading case. We observe that MT mapping under PRE binning yields to an optimization problem, which ends up being the dominating running time cost and thus render CS-PRE-MT of little practical relevance. Conversely, MT mapping remains simple under CS-POST-MT, which enables an ideal speedup for a large range of parallelism degree. Particularly, the POST binning strategy represents another contribution in itself, as it (i) enables an ideal speedup of CS-POST-MT up to $P < 64$, (ii) is twice

as fast as PRE when $P = 1$ and (iii) is additionally more accurate under both MT and ST settings.

While our methodology is general, we additionally make the knowledge gained in this research readily available to the community as an simulation engine (i.e., directly usable on standard scenarios of event-driven simulation) of the ccnSim open source simulator at [1].

ACKNOWLEDGMENTS

This work was carried out at LINCS (https://lincs.fr) and benefited from support by NewNet@Paris, Cisco's Chair "NETWORKS FOR THE FUTURE" at Telecom ParisTech (http://newnet.telecom-paristech.fr).

REFERENCES

[1] http://perso.telecom-paristech.fr/drossi/code/ccnSim.
[2] ttps://github.com/TeamRossi/ccnSim-0.4-Parallel.
[3] Akaroa Project Website. https://akaroa.canterbury.ac.nz/akaroa/.
[4] A. Araldo, D. Rossi, et al. Cost-aware caching: Caching more (costly items) for less (isps operational expenditures). *IEEE Transactions on Parallel and Distributed Systems,*, 2015.
[5] S. Arianfar and P. Nikander. Packet-level Caching for Information-centric Networking. In *ACM SIGCOMM, ReArch Workshop*. 2010.
[6] ATIS 5G Americas. Understanding information centric networking and mobile edge computing. https://goo.gl/QMEbKV, 2016.
[7] G. Carofiglio, L. Mekinda, et al. Analysis of latency-aware caching strategies in information-centric networking. In *Proc. of ACM CoNEXT, CCDWN Workshop*. 2016.
[8] H. Che, Y. Tung, et al. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE JSAC*, 20(7):1305, 2002.
[9] G. Ewing, K. Pawlikowski, et al. Akaroa-2: Exploiting network computing by distributing stochastic simulation. *SCSI Press*, 1999.
[10] R. Fagin. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences*, 14(2):222, 1977.
[11] S. K. Fayazbakhsh, Y. Lin, et al. Less Pain, Most of the Gain: Incrementally Deployable ICN. *ACM SIGCOMM*, 2013.
[12] N. Fofack, P. Nain, et al. Analysis of TTL-based cache networks. In *Proc. of VALUETOOLS*. 2012.
[13] N. Fofack, P. Nain, et al. Performance evaluation of hierarchical TTL-based cache networks. *Elsevier Computer Networks*, 65, 2014.
[14] C. Fricker, P. Robert, et al. A versatile and accurate approximation for LRU cache performance. In *Proc. of ITC24*. 2012.
[15] R. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30, 1990.
[16] M. Garetto, E. Leonardi, et al. Efficient analysis of caching strategies under dynamic content popularity. In *Proc. of IEEE INFOCOM*. 2015.
[17] P. Heidelberger and H. S. Stone. Parallel trace-driven cache simulation by time partitioning. In *IEEE Winter Simulation Conference*. 1990.
[18] J. Jaeyeon, A. W. Berger, et al. Modeling TTL-based Internet caches. In *Proc. of IEEE INFOCOM*. 2003.
[19] P. Jelenkovic and A. Radovanovic. Asymptotic insensitivity of least-recently-used caching to statistical dependency. In *In Prof of. IEEE INFOCOM*. 2003.
[20] R. E. Kessler, M. D. Hill, et al. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664, 1994.
[21] E. Leonardi and G. Torrisi. Least Recently Used caches under the Shot Noise Model. In *Proc. of IEEE INFOCOM*. 2015.
[22] J. Liu. Parallel simulation of hybrid network traffic models. In *Proc. of IEEE PADS Workshop*. 2007.
[23] J. Liu and Y. Li. On the performance of a hybrid network traffic model. *Simulation Modelling Practice and Theory*, 16(6):656 , 2008.
[24] 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.
[25] V. Martina, M. Garetto, et al. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*. 2014.
[26] D. Berger et al. Exact Analysis of TTL Cache Networks: The Case of Caching Policies Driven by Stopping Times. In *Proc. of ACM SIGMETRICS*, pages 595–596. 2014.
[27] N. Fofack et al. On the performance of general cache networks. In *Proc. of VALUETOOLS Conference*, pages 106–113. 2014.
[28] J. Liu, Y. Liu, et al. GPU-assisted Hybrid Network Traffic Model. In *Proc. of ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'14)*. 2014.
[29] K. Pentikousis et al. Information-centric networking: Evaluation methodology. Internet Draft, https://datatracker.ietf.org/doc/draft-irtf-icnrg-evaluation-methodology/, 2016.
[30] G. Xylomenos et al. A survey of information-centric networking research. *IEEE Commun. Surveys & Tutorials,*, 16(2):1024, 2014.
[31] V. Misra, W. Gong, et al. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *ACM SIGCOMM Comput. Commun. Rev.*, 30(4):151, 2000.
[32] 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.
[33] E. J. Rosensweig, J. Kurose, et al. Approximate Models for General Cache Networks. *Proc. of IEEE INFOCOM*, 2010.
[34] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *IEEE INFOCOM, NOMEN Worshop,*. Orlando, FL, 2012.
[35] G. Rossini and D. Rossi. ccnSim: a highly scalable CCN simulator. In *Proc. of IEEE ICC*. 2013.
[36] G. Rossini and D. Rossi. Evaluating CCN Multi-path Interest Forwarding Strategies. *Comput. Commun.*, 36(7):771, 2013.
[37] G. Rossini and D. Rossi. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proc. of ACM ICN*. 2014.
[38] M. Tortelli, D. Rossi, et al. Accurate, scalable and flexible analysis of general cache networks (extended version). In *Tech. Rep.* 2016.
[39] M. Tortelli, D. Rossi, et al. Icn software tools: survey and cross-comparison. *Elsevier Simulation Modelling Practice and Theory*, 63:23, 2016.
[40] M. Tortelli, D. Rossi, et al. Modelgraft: Accurate, scalable, and flexible performance evaluation of general cache networks. In *ITC28*. 2016.
[41] M. Tortelli, D. Rossi, et al. A hybrid methodology for the performance evaluation of internet-scale cache networks. *Elsevier Computer Networks,*, 2017.
[42] S. Traverso, M. Ahmed, et al. Temporal locality in today's content caching: Why it matters and how to model it. *ACM SIGCOMM Comput. Commun. Rev.*, 43(5):5, 2013.
[43] S. Traverso, M. Ahmed, et al. Unravelling the impact of temporal and geographical locality in content caching systems. *IEEE Transactions on Multimedia*, 17(10):1839, 2015.
[44] J. J. Yi and D. J. Lilja. Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations. *IEEE Transactions on computers*, 55(3):268, 2006.

Michele Tortelli received the MSc and PhD degrees from Politecnico di Bari, Italy, in 2011 and 2015, respectively. He has lead the Telecom ParisTech team in the EIT Digital European project "Information-aware data plane for programmable networks", and he participated in the program committee of IEEE and ACM ICN conferences. He is currently a Research Assistant at Telecom ParisTech, working on the design, modeling, and performance evaluation of massive-scale information-oriented networks.

Dario Rossi received his MSc and PhD degrees from Politecnico di Torino in 2001 and 2005 respectively, was a visiting researcher at University of California, Berkeley during 2003-2004, and is currently Professor at Telecom ParisTech and Ecole Polytechnique. He has coauthored over 150 conference and journal papers, received several best paper awards, a Google Faculty Research Award (2015), and an IRTF Applied Network Research Prize (2016). He is a Senior Member of IEEE and ACM.

Emilio Leonardi received the Dr. Ing. degree in electronics engineering and Ph.D. degree in telecommunications engineering from the Politecnico di Torino, Turin, Italy, in 1991 and 1995, respectively. He is currently a Professor with the Department of Electronics at Politecnico di Torino, Turin, Italy. His research interests include performance evaluation of computer networks and distributed systems, dynamics over social networks, and human centric computation.