sLRFU: A Data Streaming Based Least Recently Frequently Used Caching Policy

Antonio A. Rocha
Fluminense Federal University, Brazil
arocha@ic.uff.br

Theodoros Salonidis, Ting He
IBM T.J. Watson Research Center
{tsaloni, the}@us.ibm.com

Don Towsley
Univ. of Massachusetts, Amherst
towsley@cs.umass.edu

Abstract—In this paper, we propose a novel caching policy called stream Least Recently/Frequently Used (sLRFU) that aims to maximize cache hit rate of CCN routers by incorporating content popularities in the caching decisions. In contrast to existing popularity-based caching policies, sLRFU copes with dynamics in content popularity while operating under the memory and high processing rate constraints of CCN network routers.

I. INTRODUCTION

The deployment of network routers with caching capabilities has grown significantly in the Internet in the last decade, thus enabling the vision of content-centric networks (CCNs). Those caching routers (CR) must decide according to a caching policy, based only on limited and local information, whether a data should be stored and what should discarded. Such caching policies play a key role in the potential gains in network performance obtained with the CR. Thus, research in this area has received a significant attention, which includes not only the definition of new caching strategies, but also the review of some well-established techniques of caching policies.

Typically CRs need to make caching decisions for a large number of data elements with relatively small cache sizes at very high speeds (network line rate). In practice, such memory and speed constraints often dictate caching policies of low complexity such as Least Recently Used (LRU) [1]. LRU uses the recency of content requests to define the content replacement strategy. One problem of LRU is that the content popularity is not explicitly considered and thus, depending on order of requests, a popular element may be replaced by a recently requested less popular element.

Another well-known caching policy is the Least Frequently Used (LFU) [1]. It keeps track of content request history to estimate content popularity and stores in cache those elements with the highest popularity to achieve a high cache-hit ratio. Despite overcoming the limitation of LRU by reducing the probability of replacing a popular element by a recently requested less popular element, the LFU policy suffers in other aspects. Firstly, elements that used to be highly popular in the past but are no longer requested may be kept for a long time, wasting cache space. This is because LFU ignores recency of requests and thus may perform poorly in case of content popularity changes. Secondly, for high speed caching routers, the expected amount of requests is large, and the solution of keeping a counter for each distinct element can be costly in terms of both memory/storage usage and update time.

One way to cope with the first issue is to use a window model (sliding or jumping). However, the window introduces a trade-off between recency and frequency. While a small window better captures changes in popularities, a larger window allows more precise popularity estimates. Furthermore, in a windowed model, only recent elements (e.g., elements in the last N requests) should be considered and the rest should be discarded. Thus, while in the original LFU only one counter is needed for each distinct element, in the windowed case it is necessary to track the order (or timestamp) of every request.

Between the extremes of LRU and LFU, there exist a spectrum of cache replacement policies, called Least Recently/Frequently Used (LRFU) [2], that explores the recency and frequency trade-off. LRFU aims to combine benefits of both LRU and LFU by specifying a weight for each request that decays exponentially over time. By tuning the rate of the exponential decay, LRFU achieves a spectrum of recency/frequency trade-off.

In this work we propose a caching policy called streaming Least Recently/Frequently Used (sLRFU) that aims to maximize cache hit rate of CR under popularity changes by incorporating content frequency and recency in cache replacement decisions. sLRFU differs from LRFU in the following aspects: (i) it explicitly discards old requests by only considering requests in a sliding window in estimating content popularity; and, (ii) it explicitly combines LFU and LRU by partitioning the cache into an LFU-managed portion (based on a sliding window) and an LRU-managed portion. Operating under the memory and high processing rate constraints of CR, sLRFU copes with dynamics in content popularity using a streaming algorithm that estimates the top-k popular elements in a sliding window. The policy places the top-k elements in cache for a dynamically determined value of k based on bounds given by the streaming algorithm. We analyze the hit ratio and the adaptivity of sLRFU through simulations. Evaluations on synthetic and real-world traces show that sLRFU outperforms both LRU and a policy that caches most popular elements in the sliding window (i.e., windowed variation of LFU).

II. sLRFU PROPOSAL

Consider a high speed CR that receives 1 billion content requests per second. At every request, sLRFU policy implementation should: (i) increase the counter for that specific content and decrease for the one expired; (ii) check if the...
requested content is in cache or not; (iii) if not, recover it and forward to consumer; and, (iv) based on the last $N$ requests, possibly rearrange the top-$k$ most popular elements in cache (adding and/or removing content from the list), and complementing the available spaces in cache with the $C - k$ most recently requested elements that are not already in cache, where $C$ is the cache size.

For the example described, an important question is: is there an efficient way (computationally feasible) to estimate the top-$k$ popular elements from the last $N$ requests? According to Hui Chen [3], a data stream may be seen as a continuous, unbounded, and time-ordered sequence of data elements, arriving at a high rate. The challenge is that generally it can be examined only once, in a limited computational capability environment (e.g., with memory space restriction), but with expectation of fast and accurate results. Now, lets consider requests arriving in CRs as a data stream, sLRFU answers positively the posed question and achieves the above objectives using a top-$k$ estimation data stream algorithm adapted to work efficiently in a windowed manner [4].

III. sLRFU Evaluation

We have implemented sLRFU in a simulator and derived some initial results to show the potential of our proposal. For the simulations, we assume all elements have the same size. We have defined a cache with limited content storage size ($C = 100$) and a memory space restriction for the streaming algorithm in terms of the number of counters ($M = 500$). For this preliminary evaluation, we have implemented the top-$k$ data streaming algorithm in [4] to work in jumping windows. We use a segment of size $N = 3000$ requests as learning phase in which we compute the file popularities and determine the top-$k$ estimates. The algorithm returns $k$ elements (where $0 \leq k \leq C$) that are guaranteed to be part of the ground truth requested files distribution. The learning phase is used to determine the set of files that will be placed in the cache because of frequency and to define the fraction of space must be filled with top-$k$ (consequently, the space left for recent requests). Every $N$ requests the learning phase is recomputed.

We evaluated sLRFU on synthetic data and real-world traces. The latter consists Web content requests collected from an industrial research lab and YouTube video requests obtained from a campus network gateway. The synthetic data model assumes that files requests follow a Zipf distribution. And, to characterize changes on popularities, we considered with some rate changes on skewness parameter (0.5 or 1.5), number of requested files (1000 or 1500) and rank of files.

All results are plotted in Figure 1. The graph (a) shows the mean hit rate (with confidence interval) obtained from 10 simulation runs, considering requests following the synthetic data. The first two points are the proposed policy and the simple LRU policy. We observe that in those results the sLRFU (hit rate 0.71) outperforms a baseline LRU (hit rate 0.65). The graph also shows the same kind of results for the case in which a Real Top-$k$ (RTk) estimate has been used to fully populate the cache and for six other base-line schemas (STk) in which we fixed the fraction of frequent content placed in cache to a set of values ($k = 10 30 50 70 90 100$). Results show that sLRFU has performed better than RTk and at least as good as all the schemas with fixed $k$. However, it is worth noting that the schemas with fixed $k$ may suffer under different request patterns. Figure 1(b) and (c) show the results obtained from trace-driven simulation. Both graphs illustrate the evaluation of the three schemas (sLRFU, LRU and RTk) inside every each window during all the simulation time. While the graph (b) shows that the performance computed for all schemas are very similar with slightly better result for sLRFU and RTk, the graph (c) shows the case in which the RTk performs much worse than the other two policies. The total hit rates for both trace-driven simulations are summarized on the top of each graphs.

IV. Future Directions

We foresee many directions for future research in this work. Now, we are incorporating sliding window functionality to sLRFU. The streaming algorithm [4] has been adapted to work in such a manner that considers a larger window controlling adds and removals of smaller sub-windows, while keeping all guarantees of top-$k$ estimates. Afterward, we will optimize/evaluate impact of $N$. We will also evaluate the performance of sLRFU against other cache policies, such as k-LRU, LRFU and Beledy’s algorithm. Later, we will investigate the performance in a caching network, where CRs implement sLRFU as its own caching policy.

REFERENCES