DSCA: A Data Stream Caching Algorithm

Antonio A. Rocha¹, Mostafa Dehghan²
Theodoros Salonidis³, Ting He³, and Don Towsley²

¹Fluminense Federal University, Niteroi, Brazil

²University of Massachusetts, Amherst, MA, USA

³IBM T.J. Watson Research Center, Yorktown, NY, USA

arocha@ic.uff.br, {mdehghan, towsley}@cs.umass.edu
{tsaloni,the}@us.ibm.com

ABSTRACT

The deployment of caches in the Internet has grown significantly in the last decade, thus enabling the vision of Content-Centric Networks (CCNs). The caching policy employed at these routers has significant impact on the potential gains in network performance. Policies that adapt to changes in content popularities are of special interest. In this paper, we propose a novel caching policy called Data Stream Caching Algorithm (DSCA) with the goal of maximizing cache hit rate of CCN routers by incorporating content popularity in caching decisions. In contrast to existing popularity-based caching policies, DSCA copes with dynamics in content popularity while operating under the memory and high processing rate constraints of CCN network routers. DSCA achieves the above objectives using a data streaming algorithm that identifies the most popular contents adapted to work in a windowed manner. We analyze the performance and robustness of the proposed caching policy through simulations. Evaluations on synthetic data and real-world traces show that DSCA outperforms LRU and other caching policies evaluated in this work.

CCS Concepts

•**Networks** → *Network performance evaluation*;

Keywords

Content-Centric Network Routers, Caching Policy, Data Stream, Least Recently/Frequently Used.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCDWN '15 Heidelberg, Germany

© 2015 ACM. ISBN 978-1-4503-4054-0/15/12...\$15.00

DOI: <http://dx.doi.org/10.1145/2836183.2836191>

1. INTRODUCTION AND BACKGROUND

The deployment of caches in the network has grown significantly in the Internet in the last decade, thus enabling the vision of content-centric networks (CCNs) [8]. These caches rely on a caching policy to decide whether a piece of content should be stored and what content should be discarded. Caching policies as key contributors to the performance of today's networks have received significant attention in the research community that has lead to the definition of new caching strategies as well as assessment of some well-established caching policies [12,13].

Typically caches need to make caching decisions for a large number of contents at high speeds. This is because the cache resides at a router fed at network line rate. In practice, such memory and speed constraints often dictate the use of low complexity caching policies such as Least Recently Used (LRU) [10]. LRU uses the recency of content requests to define the content replacement strategy. One problem with LRU is that it does not consider content popularity and thus, depending on the order of requests, a popular content may be replaced by a recently requested less popular content.

Another well-known caching policy is the Least Frequently Used (LFU) [10]. It keeps track of content popularities and stores the most popular contents in the cache to achieve a high cache-hit ratio. Despite overcoming the limitation of LRU by reducing the probability of replacing a popular content by a recently requested less popular content, the LFU policy suffers in other aspects. It does not naturally handle time varying popularity; contents that were highly popular in the past but are no longer requested may be kept for a long time, hence wasting the cache space.

One way to cope with this issue is to use a windowed model (sliding or jumping). A windowed approach 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 contents (e.g., contents in the last N requests) are considered and the rest are discarded.

Between the extremes of LRU and LFU, there exist a spectrum of cache replacement policies, called Least Recently/Frequently Used (LRFU) [9], that explores the recency and frequency trade-off. LRFU combines benefits of 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 *Data Stream Caching Algorithm (DSCA)* with the goal of maximizing cache hit ratio under content popularity changes by incorporating content frequency and recency in cache replacement decisions. DSCA differs from LRFU in the following aspects: (i) it explicitly discards old requests by only considering requests in a sliding window in estimating content popularities; and, (ii) it explicitly combines LRU and LFU by partitioning the cache into LRU-managed and LFU-managed portions. DSCA copes with dynamics in content popularities using a streaming algorithm that estimates the top- k popular contents in a jumping window. Our policy places the top- k contents in cache for a dynamically determined value of k based on bounds given by the streaming algorithm.

The rest of this paper is organized as follows. In Section 2, we briefly describe the most relevant related works. In Section 3, we present the Data Stream Caching Algorithm (DSCA). Simulation results on several workloads comparing the proposal to other replacement policies are presented in Section 4. Finally, in Section 5, we present our conclusions and future directions.

2. RELATED WORK

Caches are used in various applications such as databases, operating systems and computer networks, and hence there has been an ongoing effort in developing and improving caching algorithms. In computer networking, caching algorithms find applications in problems related to Web servers [11, 16], Multimedia streams [18], Peer-to-peer architectures [17, 19], CDNs [20], and more recently in CCNs [4, 10].

The LRU policy is widely used as the de-facto standard caching algorithm mostly due to its simplicity and reasonable performance. LRU replaces the content that has not been requested for the longest time in favor of a request for a new content. This simple policy suffers from disregarding content popularities, and hence performs poorly when there is little temporal locality. Hence, there has been a huge effort to improve the LRU performance; see for example LRFU [9], Clock [6], ARC [13], CAR [2], and references therein.

The LFU policy replaces the least frequently used element and, in very specific scenarios (e.g., fixed set

of elements and request pattern), it is the optimal policy [1]. However, LFU suffers from disregarding recency by keeping once popular contents with large request counts that are requested again.

A pure LRU or LFU policy is rarely used due to the above mentioned drawbacks. To cope with the limitations of LRU and LFU policies, researchers have tried to combine recency and frequency. LRU- k [15] is one of the first algorithms of this type. LRU- k records the last k times each content was requested and evicts the one with least recent k^{th} access. With $k = 2$, for instance, LRU-2 replaces the element with the least recent penultimate access. LRU- k improves LRU introducing the notion of frequency, while eliminating the lack of adaptability inherent to LRU.

A more recent caching policy with similarities to LRU- k is the k -LRU [12] algorithm. In this policy, content elements are stored in the cache only after passing through $k - 1$ virtual caches. Upon a request, if the element is in virtual cache i , then it is promoted to virtual cache $i + 1$; otherwise, if the element is not found in any of the k caches, it is stored in the first one. Whenever an element reaches the k^{th} cache, the content is physically stored. And, every cache (virtual or physical) manages items following the LRU policy. According to Martina et al. [12], $k = 2$ is enough to achieve significant improvements over LRU. It is justified by the fact that 2-LRU is able to filter out unpopular contents and cope with changes in popularity.

In between the extremes of LRU and LFU, there exist many other cache replacement policies. LRFU [9], for instance, combines benefits of LRU and LFU by specifying a weight for each request that decays exponentially over time. By tuning the rate of the exponential decay (λ), LRFU achieves a spectrum of recency/frequency trade-off: if $\lambda \rightarrow 1$, the policy emphasizes recency; if $\lambda \rightarrow 0$, the policy tends to frequency. The problem with LRFU is that its performance depends on the λ parameter that must be given *a priori*.

The necessity to pre-set the parameter λ in LRFU has motivated the development of self-tuning algorithms capable of reaching good performance independent of changes in the workload. Adaptive Replacement Cache (ARC) [13] is such a policy that maintains a history of both frequently and recently used elements, and keeps track of those recently evicted. The recently evicted list is used to automatically adjust the preference to recent or frequent elements.

Belady's algorithm [3] is an optimal policy for replacing the content elements in cache. It evicts the element that has the greatest distance to its next occurrence. The Belady algorithm can not be implemented in practice since it requires predicting the time of future content requests. However, it is useful as an upper bound in evaluating the performance of other algorithms. We compared Belady's algorithm results to those obtained from DSCA, LRU and 2-LRU.

3. DSCA DESIGN

DSCA maintains counters for every request. At the time of a request, DSCA (i) increases the counter for that specific content, (ii) checks if the requested content is in cache or not; (iii) if not, recovers it and forwards it to consumer; and, (iv) possibly rearranges the top- k most popular elements in cache.

The important question to ask is: *is there an efficient way (computationally feasible) to estimate the top- k popular elements?* According to Hui Chen [5], 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 a data request can be examined only once, and generally in a computationally limited environment (e.g., with memory space restriction), but with the expectation of fast and accurate results. DSCA gives a positive answer to the above question by modifying a top- k estimation data streaming algorithm [14] to work efficiently in a windowed manner.

3.1 DSCA Operation

Let C denote the cache size in number of contents. Let L_{lru} and L_{stk} be two distinct lists maintained by DSCA containing references to recently requested elements and to frequently requested elements, respectively.

L_{lru} maintains content identifiers according to the LRU policy, ordered from the most to the least recently requested. New content requests are added to the head of L_{lru} . If the element is already in the list, it is removed from the current position, otherwise the element at the tail of L_{lru} must be discarded.

L_{stk} maintains a set of element identifiers and their respective frequency counters using the *Space Saver* algorithm [14]. In the list, M distinct elements are monitored. Associated with each element is a triple that includes the element identifier, a frequency counter, and an error estimate of the frequency counter. For file i for example, L_{stk} would have the entry (i, f_i, e_i) . Elements in L_{stk} are ordered by their frequency counters from the most to the least frequent items. Every new content request is stored in the list. After storing the first M distinct element triples, for each requested content the counter is incremented if it is a monitored element; otherwise, the triple with the smallest counter in the list has its id replaced by the newly requested element id , the counter is incremented, and the error takes the counter value of the evicted element, i.e. $f_{i'} \leftarrow f_i + 1$ and $e_{i'} \leftarrow f_i$.

L_{stk} makes it possible to estimate the top- k frequent content elements. The top- k estimation algorithm [14] computes k (where $0 \leq k \leq C$) and returns the first k ranked elements from L_{stk} that are guaranteed to be part of the ground truth requested files distribution. So, k is determined by the *Space Saver* algorithm [14] and is equal to the number of “top- k guaranteed” items in L_{stk} .

Content i is “top- k guaranteed” if its frequency counter is greater or equal to the counter of the $(C+1)^{th}$ ranked element in L_{stk} (i.e., $f_i - e_i \geq f_{(c+1)}$, where $(c+1)$ is the id of the $(C+1)^{th}$ ranked element).

To cope with possible changes in content popularity, we have modified the top- k data streaming algorithm to work with a jumping window of size N . For every sequence of N requests, we use L_{stk} to compute k , the set of top- k guaranteed contents that will be placed in the cache. For every N requests L_{stk} is recomputed from scratch.

DSCA works in equally sized phases of N requests. As a bootstrap in the first phase, k is set to zero, the cache operates as a regular LRU policy and it is fully filled with all the elements of L_{lru} . At the end of any phase and before the beginning of a new one, DSCA defines the new value of k and places the top- k guaranteed contents in cache based on the estimates from previous (learning) phase. The remaining cache space is filled with $C - k$ most recently requested elements from L_{lru} that were not already placed as one of the top- k guaranteed elements. After this point, the $C - k$ portion of the cache space is maintained as an LRU cache based on the request arrivals.

4. PERFORMANCE EVALUATION

In this section, we evaluate the performance of DSCA through synthetic and trace-driven simulations. The simulations consist of a single cache receiving content requests according to various workloads. We compare the hit probabilities obtained by the DSCA with those of LRU and 2-LRU. We also compare DSCA to Belady’s offline optimal replacement algorithm.

Here, we assume that all contents have the same size, and define a cache with storage capacity $C = 100$. For DSCA, we set $M = 500$ in the algorithm. We also evaluate DSCA for different window sizes with $N \in \{1500, 3000, 6000, 9000, 12000, 15000\}$ requests.

4.1 Simulation Workloads

We use both synthetic and real-world traces as workloads in our simulator. The traces consist of Web content requests collected from the IBM T.J. Watson Research Center [22] and YouTube video requests obtained from a campus network gateway [23]. In order to perform controlled experiments, we have also used two different synthetic data models as workload in the simulations, both characterizing changes in content popularities.

Our first synthetic workload model assumes that element requests follow a Zipf distribution with skewness parameter $\alpha = 0.8$ and that the popularity of each content changes over time. Thus, we consider a fixed set of $L = 1500$ contents and assume that with rate 0.1 file i changes the rank with file $(i \bmod L) + 1$, for $1 \leq i \leq L$. This model captures the Zipf-like popularity law and accounts for changes on specific content element popu-

(a) Synthetic model 1

(b) Synthetic model 2

Figure 1: Evaluation results through simulations with request from synthetic data

larities.

The second synthetic workload model is an extension of the first model where the skewness parameter α and the number of contents L also change. The model considers two different values for $\alpha \in \{0.5, 0.8\}$ and, with rate $1/2000$, changes the skewness parameter. With the same rate $1/2000$, the model also changes the number of elements, which may take the values $L \in \{1000, 1500\}$. As in the first synthetic model, this one captures the Zipf-like popularity law and changes on element popularities, but it also considers changes on popularity distribution and on the number of content elements possibly requested.

Our synthetic data models follow the same idea of Wolman et al. [21] and Elayoubi et al. [7] in which content elements have dynamic popularities and finite lifetimes. However, we have added to account that the number of elements and the popularity distribution skewness may also change over time.

4.2 Simulation Results

Results for the synthetic data models are found in Figure 1(a) for the workload model that considers only changes in the content rank, for a fixed number of elements and skewness distribution parameter (i.e., $L = 1500$ and $\alpha = 0.8$); and Figure 1(b), the workload model that considers changes in the rank of contents and also in the number of elements ($L \in \{1000, 1500\}$) and skewness parameter ($\alpha \in \{0.5, 0.8\}$).

Figure 1(a) shows the hit probabilities computed as a function of time for the three schemes (DSCA, LRU and 2-LRU). For DSCA, the estimates are plotted for all values of N . We observe from those results that DSCA per-

forms better than LRU and at least as good as 2-LRU, independent of the jumping window sizes adopted in the algorithm. We also observe that all DSCA schemes and 2-LRU provide hit probabilities approximately 15% below that of Belady's algorithm.

Figure 1(b) shows the average hit probability (with confidence interval) obtained from 10 simulation runs. The first two points are respectively the LRU and 2-LRU policies, and the last three points are DSCA results for $N \in \{6000, 9000, 12000\}$. (Because of space limitations, we do not show the other values of N , but they do not differ much from those shown in the graph.) We observe that for these three values of N , DSCA has a higher average hit probability than LRU and 2-LRU. The graph also shows that the DSCA average hit probabilities are 18% below that of Belady's algorithm.

Figures 2(a) and 2(b) show the results obtained from trace-driven simulation. Both graphs illustrate the hit probabilities for the three schemes (DSCA, LRU and 2-LRU) inside intervals of 3000 requests during all the simulation time. Figures 2(a) plots the results for the YouTube trace. We observe that the performance computed for all schemes are very similar. We also note that the hit probabilities for all the schemes are close to those obtained with the Belady algorithm. Figures 2(b) shows the results for IBM Web content request trace. In this graph, we observe an interesting result. For all values of N , DSCA performs as well as LRU, and, DSCA and LRU outperform 2-LRU. DSCA and LRU have performed approximately 5% lower than Belady. The total hit probabilities of both trace-driven simulations for all schemes (LRU, 2-LRU, Belady and different window sizes of DSCA) are summarized in Table 1.

(a) YouTube accesses trace

(b) Web access trace

Figure 2: Evaluation results through trace-driven simulations

Table 1: Total hit probabilities for trace-driven simulations

Policy	YouTube	IBM Web Access
LRU	0.0759	0.5230
2-LRU	0.0309	0.5563
Belady	0.1402	0.5894
DSCA (N=1500)	0.0756	0.5450
DSCA (N=3000)	0.0766	0.5456
DSCA (N=6000)	0.0769	0.5434
DSCA (N=9000)	0.0774	0.5428
DSCA (N=12000)	0.0773	0.5410
DSCA (N=15000)	0.0772	0.5429

5. CONCLUSIONS AND FUTURE WORK

In this work, we presented DSCA, a new caching policy that incorporates content popularity in caching decisions. DSCA uses a jumping window data streaming algorithm to dynamically estimate the most popular contents while operating under memory and high processing rate constraints. We believe that our algorithm forms a new approach to capture frequency/recency trade-offs in caching for edge content-centric networks. We used synthetic data and real-world traces to evaluate through simulations the effectiveness of our policy. The simulation results show that DSCA adapts to the dynamics in different workloads, and outperforms the LRU and 2-LRU policies.

As a future research, we intend to incorporate a slid-

ing window functionality to DSCA. The streaming algorithm [14] can be modified to employ a larger window that controls addition and removal of smaller sub-windows, while maintaining accuracy of the top- k estimates. Another interesting avenue of future work would be an investigation of analytical model for optimizing the window size N . Finally, we plan to investigate the performance of DSCA in a setting with network of caches.

Acknowledgment

This work was supported in part by CNPq and FAPERJ, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001 and the National Science Foundation under Grant No. CNS-1413998. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsors. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

6. REFERENCES

- [1] A. V. Aho, P. J. Denning, and J. D. Ullman. Principles of optimal page replacement. *J. ACM*, 18(1):80–93, Jan. 1971.
- [2] S. Bansal and D. S. Modha. Car: Clock with adaptive replacement. In *USENIX/FAST*, pages 187–200, 2004.
- [3] L. A. Belady. A study of replacement algorithms

- for virtual storage computers. *IBM Systems Journal*, 5(2):78–101, 1966.
- [4] W. Chai, D. He, I. Psaras, and G. Pavlou. Cache “less for more” in information-centric networks. In *NETWORKING 2012*, volume 7289 of *Lecture Notes in Computer Science*, pages 27–40. Springer Berlin Heidelberg, 2012.
- [5] H. Chen. Mining top-k frequent patterns over data streams sliding window. *Journal of Intelligent Information Systems*, 42(1):111–131, 2014.
- [6] F. Corbato. A paging experiment with the multics system. *Honor of P.M. Morse*, pages 217–228, 1969.
- [7] S. E. Elayoubi and J. Roberts. Performance and cost effectiveness of caching in the mobile access network. In *ACM/ICN*, 2015.
- [8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *ACM/CoNEXT*, pages 1–12, 2009.
- [9] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.*, 50(12):1352–1361, Dec. 2001.
- [10] Z. Li, G. Simon, and A. Gravey. Caching policies for in-network caching. In *IEEE/ICCCN*, pages 1–7, July 2012.
- [11] Z. Liu, P. Nain, N. Niclausse, and D. Towsley. Static caching of web servers. In *SPIE/Multimedia Computing and Networking*, pages 179–190, 1998.
- [12] V. Martina, M. Garetto, and E. Leonardi. A unified approach to the performance analysis of caching systems. In *IEEE/INFOCOM*, pages 2040–2048, April 2014.
- [13] N. Megiddo and D. Modha. Arc: A self-tuning, low overhead replacement cache. In *USENIX/FAST*, pages 115–130, 2003.
- [14] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.
- [15] E. J. O’Neil, P. E. O’Neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *ACM/SIGMOD*, pages 297–306, 1993.
- [16] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.
- [17] O. Saleh and M. Hefeeda. Modeling and caching of peer-to-peer traffic. In *IEEE/ICNP*, pages 249–258, 2006.
- [18] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *IEEE/INFOCOM*, pages 1310–1319, 1999.
- [19] W. Shi and Y. Mao. Performance evaluation of peer-to-peer web caching systems. *Journal of Systems and Software*, 79(5):714 – 726, 2006.
- [20] A. Vakali and G. Pallis. Content delivery networks: status and trends. *Internet Computing, IEEE*, 7(6):68–74, Nov 2003.
- [21] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *ACM/SIGMOD*, pages 297–306, 1993.
- [22] P. Zerfos, M. Srivatsa, H. Yu, D. Dennerline, H. Franke, and D. Agrawal. Platform and applications for massive-scale streaming network analytics. *IBM Journal for Research and Development: Special Edition on Massive Scale Analytics*, 57:1–11, May 2013.
- [23] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of youtube network traffic at a campus network - measurements, models, and implications. *Comput. Netw.*, 53(4):501–514, Mar. 2009.