RevOPT: An LSTM-based Efficient Caching Strategy for CDN

Hamza Ben-Ammar, Yacine Ghamri-Doudane

RevOPT: An LSTM-based Efficient Caching Strategy for CDN

Hamza Ben-Ammar and Yacine Ghamri-Doudane
La Rochelle University, L3i Lab., France.
Emails: {hamza.ben_ammar, yacine.ghamri}@univ-lr.fr

***Abstract*—In order to face the rise in data consumption and network congestion, caching structures like Content Delivery Networks (CDNs) are being more and more used and integrated into the network infrastructure. Knowing that the capacities of caching resources are most often limited due to their large operational cost, it has become very important that these entities are managed efficiently. Especially, at the caching operations level, the question that arises is what content should be cached or evicted from the cache when it becomes full. Having these in mind, we introduce a lightweight Artificial Intelligence-based caching scheme called Reversed OPT (RevOPT). In our proposal, we use a Long Short-Term Memory (LSTM) encoder-decoder model to learn future requests patterns from the past and exploit its outcome with a Counting Bloom Filter (CBF) structure to manage efficiently the caching decisions and to keep in the cache only contents expected to be reused in the near future. The conducted simulations show promising results of RevOPT in terms of the cache hit ratio compared to existing caching algorithms.**

***Index Terms*—Caching Scheme, Deep Learning, LSTM, CDN, Bloom Filter.**

I. Introduction

The last few years witnessed a shift on the Internet usage that switched from a host-centric model to a content-centric approach, especially when dealing with content retrieval and data dissemination [1]. This evolution is mostly driven by the rapid growth of media-enriched services, e.g., Peer-to Peer file sharing, Video on Demand, video/audio streaming and social networks. These had significantly changed the way that people experience the Internet, making media traffic one of the main drivers of the Internet economy. According to the Cisco Annual Internet Report [2], mobile Machine to machine (M2M) connections will grow from 1.2 billion in 2018 to 4.4 billion by 2023, which represents a four-fold growth. Across the globe, the overall IP traffic is expected to grow to 396 Exabytes (EB) per month by 2022, up from 122 EB per month in 2017.

The rapid growth of data consumption has led to the continuous raise during the last two decades of Content Delivery Networks (CDNs) [3], which use the mechanisms of caching to optimize the content delivery performance. The term *cache* [4] was first employed in computing systems to describe a data storing technique that provides the ability to access data or files at a higher speed. Caches can be implemented both in hardware and software. They are used to serve as an intermediary component between the primary storage appliance and the recipient hardware or software device to reduce the latency in data access. Caching is considered one of the most simple and effective techniques in computer science for improving a system's performance. The concept of caching was later adopted by the Internet [5]. Web caching solutions can help enhancing the content access time significantly by storing popular contents from previous requests. CDNs have become an important layer in the Internet ecosystem and are used to enhance the delivery of contents by replicating commonly the most requested ones across a globally distributed set of caching servers. The aim is to reduce the load on the origin servers and to improve the experience of the clients by delivering a local copy of the content from a nearby cache located at the edge of the network.

Due to the limited capacities of caches compared to the requested data in networks and the operational cost of large CDNs and their networks of caches, deciding which object must be stored in caches becomes critical. Even though extensive studies have been conducted over the years to propose efficient caching schemes, more room for improvement is still visible compared to the theoretical optimum (OPT) efficiency that can be achieved. The OPT replacement policy consists basically of evicting from the cache the item that will be requested the furthest in time. However, OPT can only be achieved if we have access to all future access requests, which is impractical in reality. Even if we have exactly the order of the future requests for contents, which have various sizes, computing OPT is NP-hard [6].

Using AI-based approaches, many studies tried to mimic OPT operations or predict what items will be requested in the future in order to achieve better outcomes. Nevertheless, these proposals generally induce significant temporal complexity and their efficiency is impacted by the requests access patterns and the tuning parameters. Our goal in this paper is to use AI tools to build an efficient caching scheme with a low-overhead. In this context, we propose RevOPT, which resembles the mechanisms of OPT and can be seen as a reversed version of it, where a Recurrent Neural Network (RNN) is used for the prediction of future requests to be generated. Unlike OPT where the items that will be requested the furthest in the future will be evicted from the cache, RevOPT will admit in the store the ones that will get at least one hit before the time where they are supposed to be evicted. This is basically achieved by looking each time a new request is received at the N next future distinct objects to be requested, N being the cache size in terms of the number of items. By doing so, RevOPT allows

achieving a higher cache hit ratio than other caching schemes from the literature.

The reminder of this paper is structured as follows. Section II reviews the main related work. This is followed in Section III by a detailed description of our proposal RevOPT. In Section IV, we provide the performance evaluation of our proposal to finally conclude this paper and present some future research directions in section V.

II. Related Work

The study of caching has started a long time ago [7], where it was mainly used in operating systems before it becomes widely adopted to retrieve data from the Internet. In [8] and later on in [9], Belady's MIN and, respectively, Mattson's OPT were presented and both provided an optimal replacement policy. Due to the many similarities in their operations, there seems to be a long-lived confusion between these two proposals where authors referred to Belady's MIN, but in reality, they described Mattson's OPT [10]. These optimal algorithms can be achieved only when future data accesses are known in advance, which can not be implemented in practice where caching schemes can operate only using past information. Besides, calculating OPT when the items have different sizes become an NP-hard problem [6]. Still, many proposed implementable caching algorithms were inspired by OPT. Since then, there has been a very large and considerable amount of work aiming to design efficient caching algorithms. With the proliferation over the last years of CDNs and the rapid deployment of 5G networks along with the advancements of AI and its adoption in the field of networks and distributed systems, many studies have started to propose AI-based caching schemes in order to outperform the limitations attained by classical approaches [11], [12] and reduce the gap with OPT.

In [13], the author proposed LFO (Learning From OPT), where a gradient boosting decision tree learning algorithm is used to learn from OPT and the requests' features and thus making the caching decisions. The authors in [14] presented a Machine Learning (ML) approach where the goal is to approximate Belady's MIN algorithm (oracle) by finding items to eject from the cache based on the past access information. To avoid the prohibitive cost of using the naive version of Belady's MIN, a relaxed version of it was used instead. In [15], a framework for content caching called DeepCache was proposed. By leveraging an LSTM-based model to predict the popularity of content objects, DeepCache allows to proactively cache the items that are mostly to be demanded in the future. The authors in [16] introduce a Federated learning-based Proactive Content Caching (FPCC) scheme. Their proposal aims to provide an efficient caching scheme and at the same time to preserve the users' privacy by avoiding the centralization of their data for training purposes thanks to the use of the Federated Learning (FL) approach. In [17], Xu et al. present an incremental learning-based framework for data caching in edge networks and CDNs. In their proposal and instead of rebuilding a new caching model each time the system dynamics are changed, the valuable knowledge gathered from the previous models are preserved and the same model is incrementally improved to adapt faster and efficiently to the dynamic workloads.

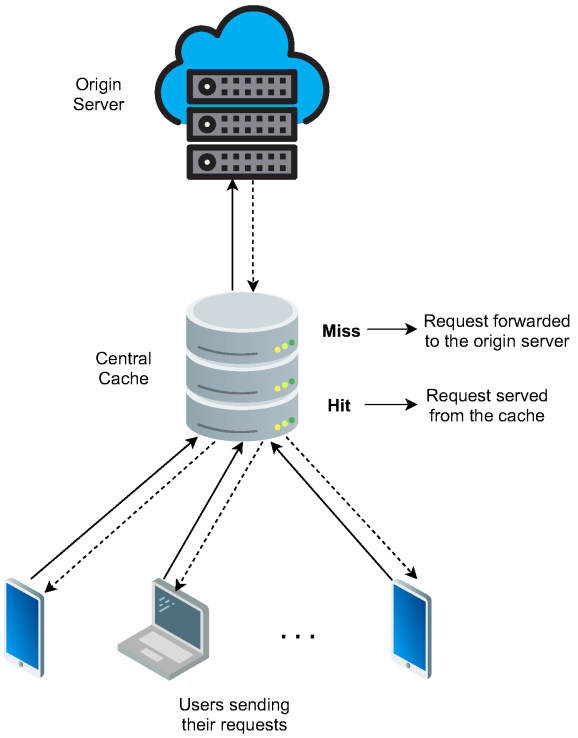

Figure 1. System model.

All these previous approaches come with a higher cost as they increase the temporal complexity. With this in mind, we aim to use AI to design an efficient caching scheme with a low-overhead.

III. LSTM-based caching

A. System model

The system model of our proposal is depicted in Figure 1. We consider a catalog of contents available to users from the Internet and hosted initially at an origin server. We have a caching entity (operated for example by a CDN) located between the users and Wide Area Network (WAN), where it can store only a portion of all the available contents. When a specific data is requested by a user, it is checked first whether or not it exists in the cache. If a copy is found (which represents what we call a cache hit), the requested item is sent to the client. If not, we say that a cache miss has occurred and the requested data is to be served by the origin server. One key metric to measure cache efficiency is thus the cache hit ratio, which is the ratio between the cache hits occurrences and the total number of requests sent by the users. Since the cache size is limited, it is essential through a caching manager to choose wisely how contents are managed within the cache. More specifically, each time a specific content passes by the cache, it has to decide whether a copy should be saved or not

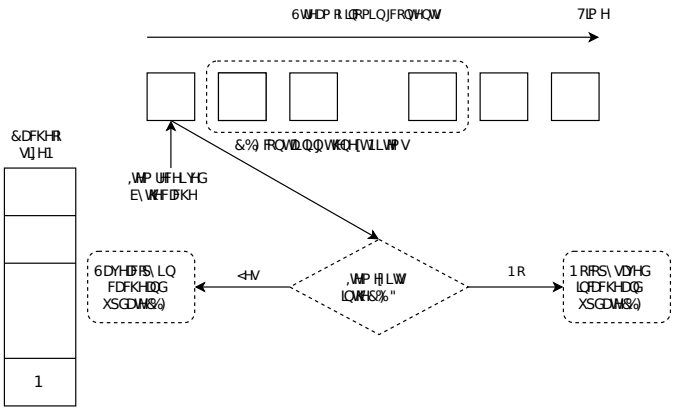

Figure 2. RevOPT overview.

(this is called the caching decision). Once the cache is full and whenever a new object is going to be stored, an item has to be chosen to be ejected from the cache (which is managed through a cache eviction policy). Multiple cache replacement algorithms exist. Among them, one can cite First In First Out (FIFO), Least Recently Used (LRU), and so on and so forth.

B. RevOPT overview

In our proposal and by leveraging state-of-the-art machine learning techniques, the basic idea is to learn future characteristics of the system. Then, we feed the learned features to a caching algorithm capable of using them to achieve improved performance, while at the same time keeping the overhead as low as possible. The caching scheme that we designed, called RevOPT, is described in Figure 2 and works as follows. First, we use a recency-based cache replacement policy like LRU. Now, once an item has been saved in the store (of size N), we know that it will be evicted only if N other distinct objects are saved in the cache. The idea then is to operate at the cache admission level and allow items to be cached only if we are sure that they will get at least one hit before getting evicted. To do so and whenever a new item is received by the cache, we look at the N next future distinct requests (in case of items having different sizes, we look at the next distinct future requests whose total size is equal to N). If the received item exists within them, a copy is saved. Otherwise, it is not cached. So, in order for RevOPT to work properly, we need to access the future requests that are to be received. For now, let's suppose that we have this information and we will explain later how it can be accessed using ML. More precisely, each time a new item is received, we need to access a table of size N and do a lookout operation. Since we aim to keep the overhead as low as possible, during the lookout operation we need only to check the presence of an element without accessing its exact position. To do so, we use a Bloom Filter (BF) [18] containing the future sequences of requested objects. A Bloom Filter is a space-efficient probabilistic data structure used generally to test whether an element is a member of a set or not, which matches perfectly the lookout operation in RevOPT. Now every time we move forward in the sequence of requests to decide whether to cache the current item or not, we need to add at the same time new elements to the table and remove from it the next element to be checked. In a classical BF, elements can be added to the set, but not removed. To avoid this issue, we use a structural variant of BF called Counting Bloom Filter (CBF) [19]. Unlike classical BF, a CBF allows updating the filter without recreating it afresh. A CBF can suffer from a high false-positive rate when the designed capacity of the table is exceeded due to its incapacity to expand the filter. Since in our case, we know in advance the maximum size of the filter, this is no longer an issue.

To sum up, the idea behind RevOPT is to offer a good trade-off between temporal complexity and cache efficiency by forbidding from the cache the items that will be saved then evicted before getting any cache hits. We have a complexity of $O(1)$ for the replacement policy since LRU is used and a complexity of $O(1)$ for the caching admission thanks to the CBF structure used in RevOPT. Of course, RevOPT is not expected to achieve the same performance as OPT. It only shares with it somehow the same conceptual similarities and can be seen as a "Reversed" version of OPT where the items that will be requested the furthest in the future are evicted from the cache. Even when having the exact information of the future requests and in the case where the items have the same size, OPT is unpractical to apply as each eviction decision has a time complexity of $O(N)$ and OPT becomes NP-hard in case of items with variable sizes.

C. LSTM encoder-decoder model

As we have seen in the previous part and to apply RevOPT, we need to have in advance the sequence of requests to be generated in the future by the users and received by the cache entity. One way to tackle this problem is to extract from a given sequence of past requests the evolution of its probability distribution in time in order to predict the future distributions. More specifically, the input sequence of items is transformed, using time intervals or based on the number of items, into a sequence of probability distributions to predict the future ones. The aim is then to use these predicted distributions to generate a sequence of items representing the future objects to be requested by the users and feed it to RevOPT. This problem falls under the umbrella of the time series prediction family. One approach to such type of prediction problems that has proven to be very effective is to use a special type of RNNs called LSTM [20].

An LSTM is basically an artificial Recurrent Neural Network architecture used in the field of Deep Learning where hidden layers are treated as memory units. It is capable of detecting patterns in sequential information by learning the relationship between present inputs and previous ones in order to predict future outputs. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs and the resulting "short memory" behavior when the most recent inputs in time are prioritized at the expense of forgetting old inputs.

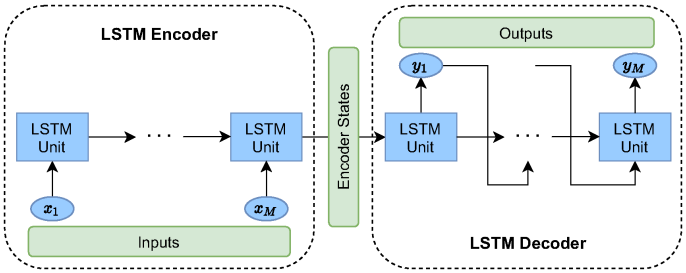

Figure 3. A basic architecture of a one-layer LSTM encoder-decoder model.

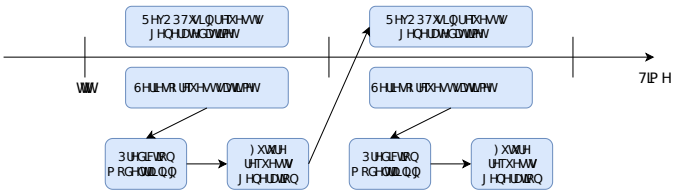

Figure 4. Applying LSTM encoder-decoder with RevOPT.

Since we are addressing multi-step time series forecasting, which is a type of sequence-to-sequence prediction problem, we use the Encoder-Decoder LSTM architecture, a very effective tool to deal with such types of problems. In this architecture (see Figure 3), we have an LSTM encoder model to read the input sequence step-by-step and encodes it into a hidden state vector representing an internal learned representation. The encoded sequence is then fed to the LSTM decoder model to generate the output sequence step-by-step using each time the actual output to train the encoder-decoder model. All of these operations are done using LSTM units [20], which are commonly composed of a cell and three types of gates (input, output and a forget gate). The cell's role is to remember values over time intervals and the gates are used to regulate the flow of information into and out of the cell.

As illustrated in Figure 4 and for each time window, we train our LSTM model on the series of requests received at that point in time to generate the future requests to be used by RevOPT in the next time window. In the meantime, we use RevOPT with the requests generated in the previous time window, except at the beginning where there is no past information. In that case, we use only LRU waiting to have enough requests sent by the users to start applying the LSTM model.

IV. Performance Evaluation and Results

A. Evaluation settings

In this section, we evaluate the performance of our proposal RevOPT in terms of prediction accuracy and cache hit efficiency. RevOPT is compared to the following caching algorithms: LRU, S4LRU [21], LRB (Learning Relaxed Belady) [14] and OPT. S4LRU is a variant from the classic LRU where the cache is quadruply-segmented and items move from one queue to another instead of considering the cache as a whole. LRU and S4LRU are widely used when studying cache performance [14]. LRB had been recently proposed and it is a machine-learning approach that tries to mimic a relaxed version of the Belady MIN algorithm. It has been compared in [14] to many other AI-based and classical caching proposals. We believe that it is interesting to use it for comparison against RevOPT.

The LSTM encoder-decoder model in our proposal was developed in Python using the Keras deep learning library. We considered a 4-layer depth model with 64 hidden units for middle layers and 128 hidden units for the others. This LSTM architecture had been chosen as we have seen that additional layers and hidden units did not show any efficiency improvement in the model's results in our case. We used the mean-squared-error and ReLU as the loss function and the activation function, respectively. The number of epochs was set to 50 after stability and we used the Adam algorithm as the optimizer with a learning rate of 0.001.

During the evaluations, we used a catalog of contents containing 10000 items whose popularity distribution follows the Zipf's law, where each object with a rank r has a probability to be requested that is equal to: $p_r = r^{-\alpha} / \sum_{i=1}^{R} i^{-\alpha}$, α being the skew of the distribution. The Zipf's law has been used in many fields, including the modeling of the requests popularity distribution in CDNs and Cloud/Edge Computing environments [22]. The synthetic datasets generated contain each 1 Million requests in total and the model training is launched each 100k requests. We considered the cases of either contents having the same sizes (cache size is then expressed in the number of maximum objects that can hold) or having different sizes (varying between 500 MB and 5000 MB) and multiple values of the Zipf's law parameter α and the cache size are tested.

B. Results and analysis

To measure the content popularity prediction's accuracy of RevOPT and its impact on the caching efficiency, we compare in Figure 5 the cache hit ratio as a function of α of our proposal with RevOPT having the exact information concerning the future sequence of requests to be generated. We can see that in general and for the results that are exposed in this figure, the LSTM encoder-decoder model achieves on average a good accuracy to predict the future requests' distribution. The accuracy is impacted by the skewness of the distribution (controlled by the parameter α of the Zipf's law in this case) as it can achieve 98% of accuracy for high values of α and vice-versa. This is actually normal as a high value of α will result in fewer items dominating the requests, which makes it easier to predict the overall popularity. In contrast, a low value of α means less skewness in the distribution and thus, more difficulty anticipating which objects will be requested in the future.

In Figure 6, we display the cache hit performance as a function of α of RevOPT in comparison to the theoretical optimum values and other caching strategies. This is realized for the case of catalog's items having the same size. We can see that in general and for multiple configurations of the cache size

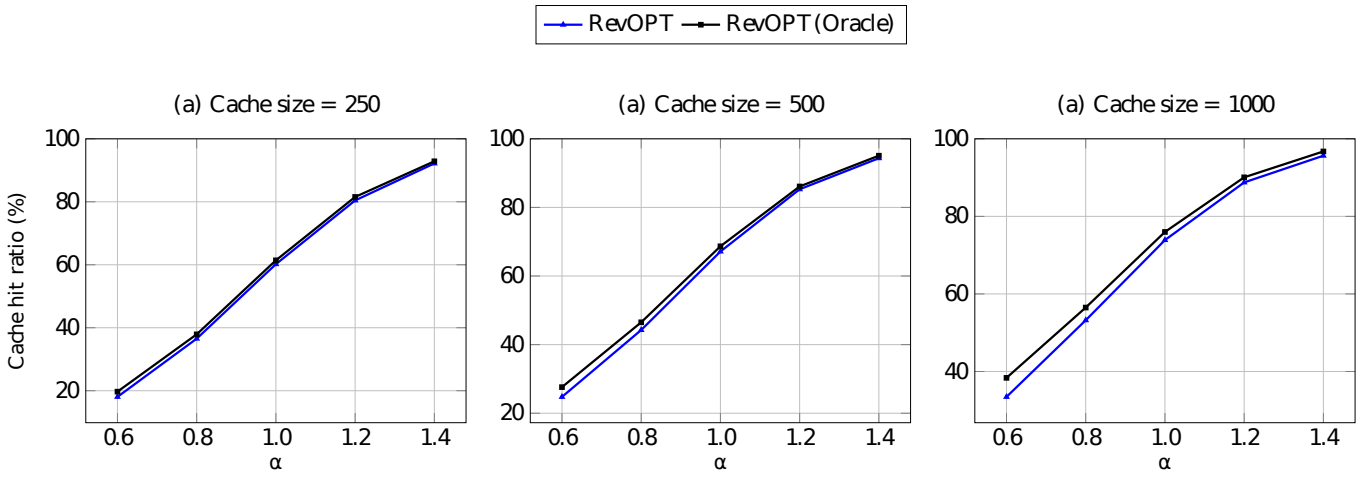

Figure 5. Popularity prediction accuracy of RevOPT.

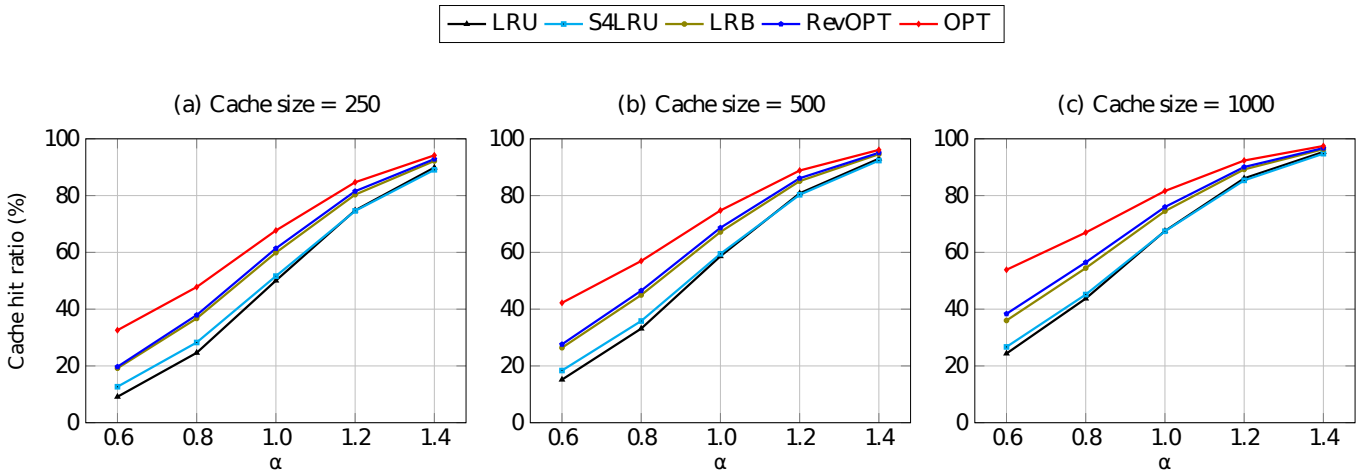

Figure 6. Cache performance in terms of cache hit with catalog's items having the same size.

and the parameter α, that our proposal outperforms classical approaches like LRU and S4LRU and is slightly better than LRB (or at least equal to it). One should note that LRB has been shown in [14] that it outperforms many efficient caching algorithms from the state-of-the-art. One important aspect to point out from Figure 6 is that as the cache size is increased, especially with the rise of α, the performance of all the tested caching strategies become similar and very close to OPT (for the same reasons explained earlier). This means that in some cases, using a caching policy other than, for example, LRU (which has very low complexity) can be useless.

In Figure 7, we have the cache hit performance as a function of the cache size in the case of the catalog's items having different sizes. Compared to OPT and other caching strategies, we get approximately the same results in terms of the cache hit performance as the catalog's objects having different sizes did not have a negative impact on RevOPT performance. The results displayed in this section represent preliminary and promising outcomes on the efficiency of RevOPT, which uses a lightweight AI approach with a low-overhead caching policy compared to other AI-based caching strategies.

V. Conclusion and Future Work

In this paper, we studied the content caching problem and proposed a new caching scheme called RevOPT. Our proposal was designed using practical state-of-the-art machine learning techniques in order to outperform classical approaches. To do so, we used an LSTM-based model to predict future requests to be generated by users by learning from their habits, which are depicted by their past data. The information provided by the LSTM model was then fed to a caching algorithm that was designed to keep the cache miss ratio as low as possible by allowing in the cache, only the items that are the most likely to be requested at least once after they are saved in the content store. To keep a very low-overhead, a Counting Bloom Filter structure was used in RevOPT to manage the necessary data that are needed to make the caching decisions. We compared our proposal to existing caching algorithms and

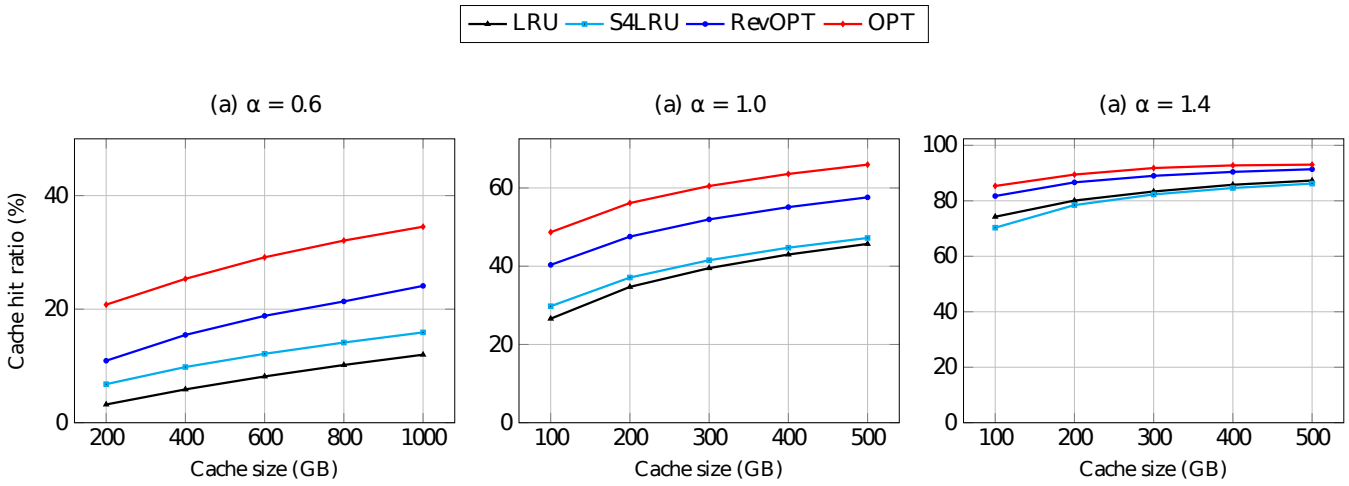

Figure 7. Cache performance in terms of cache hit with catalog's items having different sizes.

OPT. The conducted simulations have shown promising results for our proposal in multiple configurations. For now, we tested our proposal in the case of a centralized cache, with a relatively small catalog size and a content popularity following the Zipf's law. We aim in the future to adapt our proposal and extend it to the case of networks with multi-cache settings and evaluate its efficiency on real dataset traces.

References

[1] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *IEEE Communications Magazine*, vol. 49, pp. 26–36, Jul. 2011.

[2] Cisco, "Cisco annual internet report (2018-2023)," White paper, Mar. 2020.

[3] T. Hau, D. Burghardt, and W. Brenner, "Multihoming, content delivery networks, and the market for internet connectivity," *Elsevier Telecommunications Policy*, vol. 35, pp. 532–542, Jul. 2011.

[4] A. J. Smith, "Cache memories," *ACM Computing Surveys*, vol. 14, pp. 473–530, Sep. 1982.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *IEEE INFOCOM Proceedings*, Mar. 1999, pp. 126–134.

[6] D. S. Berger, N. Beckmann, and M. Harchol-Balter, "Practical bounds on optimal caching with variable object sizes," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, Jun. 2018.

[7] M. V. Wilkes, "Slave memories and dynamic storage allocation," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 270–271, Apr. 1965.

[8] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, pp. 78–101, 1966.

[9] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, pp. 78–117, 1970.

[10] P. Michaud, "Some mathematical facts about optimal cache replacement," vol. 13, pp. 1–19, Dec. 2016.

[13] D. S. Berger, "Towards lightweight and robust machine learning for cdn caching," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, Nov. 2018, p. 134–140.

[11] R. Fares, B. Romoser, Z. Zong, M. Nijim, and X. Qin, "Performance evaluation of traditional caching policies on a large system with petabytes of data," in *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*, Sep. 2012, pp. 227–234.

[12] H. S. Goian, O. Y. Al-Jarrah, S. Muhaidat, Y. Al-Hammadi, P. Yoo, and M. Dianati, "Popularity-based video caching techniques for cache-enabled networks: A survey," *IEEE Access*, vol. 7, pp. 27 699–27 719, Mar. 2019.

[14] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed belady for content distribution network caching," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Feb. 2020, pp. 529–544.

[15] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Making content caching policies 'smart' using the deepcache framework," *SIGCOMM Comput. Commun. Rev.*, vol. 48, p. 64–69, Jan. 2019.

[16] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–6.

[17] G. Xu, B. Tang, L. Yuan, Y. Xue, Z. Gao, S. Mostafa, and C. W. Sung, "An incremental learning based edge caching system: From modeling to evaluation," *IEEE Access*, vol. 8, pp. 12 499–12 509, Jan. 2020.

[18] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, p. 422–426, Jul. 1970.

[19] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 281–293, Jun. 2000.

[20] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[21] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, Nov. 2013, p. 167–181.

[22] A. Mahanti, N. Carlsson, A. Mahanti, M. Arlitt, and C. Williamson, "A tale of the tails: Power-laws in internet measurements," *IEEE Network*, vol. 27, pp. 59–64, Jan. 2013.