Caching Salon: From Classical to Learning-Based Approaches

Yangchao Zhao[1], Xu Zhang[1], Kai Yang[1], Qilin Fan[2], Dongchao Guo[3] Yongqiang Lyu[3], Hao Yin[3], and Zhan Ma[1]
[1]Nanjing University, [2]Chongqing University, [3]Tsinghua University

***Abstract*—As a key technology in content delivery, content caching can not only reduce the load on the origin server to save the traffic costs for content providers, but also can enhance users' Quality-of-Experience (QoE) by placing popular objects in close proximity. In reality, Content Delivery Network (CDN) providers always try to design their caching strategies carefully to improve the hit ratio of requests. With the development of learning-based methods and their great successes in many fields, researchers are also trying to explore learning-based caching approaches in recent years. In this paper, we first review the classical caching approaches, most of which are based on recency, frequency and content size. Then we summarize the recent learning-based methods in content caching field. Finally, we discuss the potential content caching strategies in the future, and point out some open issues. Through the studies in this article, we wish it can be used as a reference for interested parties.**

***Index Terms*—content caching, content delivery networks, learning-based methodology**

I. Introduction

With the development of 4G/5G technologies and the prevalence of smart phones in recent years, more and more people consume online videos at anytime anywhere, which results in the great increase of multimedia services. According to the forecast from Cisco [1], video will take up 82 percent of all IP traffic by 2022, up from 75 percent in 2017. Note that the traffic are mainly distributed by Content Delivery Networks (CDNs), which have cached contents in close proximity to end users [2]. The content caching can not only reduce the load on the origin server to save the traffic costs for content providers, but also can enhance users' Quality-of-Experience (QoE) by fetching contents from nearby edge servers. However, traditional caching methods do not perform well because of the various access patterns and the influence of users' behaviors. It is necessary to design either a more robust caching algorithm or a specific cache strategy for different caching goals. With the development of learning-based methods and their great successes in many fields, researchers are trying to study learning-based caching approaches in recent years.

In this paper, we have investigated the development of caching approaches. We first elaborate on some basic questions about content caching, including motivation and focus.

Why we need caching? For instance, assume that the content provider just deploys one origin server to deliver videos to users across wide geographic areas, and it's likely that multi-users watch the hot videos simultaneously. On one hand, with the number of concurrent users increases, the origin server would suffer from heavy access load, and even could cause the server to crash. On the other hand, users from faraway regions would experience long network delay due to the long propagation distance. It is worth noting that, even a few seconds of delay increase may lead users to abandon video service [3]. Higher abandon ratio would induce huge losses to content providers. To alleviate the load of origin server and improve the user experience, content providers leverage the CDN to deliver videos to end users. When a user access the videos, the nearest edge server is used to ensure the access delay and reduce the burden on the origin server if there are copies of the demanded videos. If there is no copy of the requested video in the edge server, the edge server will request it from the origin server acting as a proxy and then send the contend fetched from the origin server to the user. In this case, an additional traffic cost from the source server to edge server will be taken into account, and the overall delay increases significantly. Although the CDNs take advantage of the distributed framework and the caching technology to reduce the traffic costs, the efficiency depends mainly on the performance of the caching algorithms. The content caching strategy is important to keep the hot objects which are most likely to be accessed in the cache.

Normally, **the purpose of content caching** is to maximize the hit ratio of requests not only within recent time but also over a long period of duration. Moreover, new caching methods are designed to minimize the downloading delay, energy consumption and network congestions, or maximize user's QoE [4]. It is also important to make sure that caching algorithms have low overhead and are easy to be implemented, especially in the case facing a huge amount of data.

What to cache? Generally, we cache static contents such as videos and images which can be reusable in the edge servers to guarantee the performance. Note that caching in CDNs may be ineffective in the dynamic contents transmission scenarios, such as cloud gaming which focuses on the interactive contents between end users and game servers.

How to cache? According to the two kind of decisions made by caching strategies, caching policies are divided into two classes. One is the admission policy, which decides whether put the new coming objects into cache. Another is the eviction policy, which decides which object should be evicted out of the cache when there is no capacity for the coming objects. While most approaches focus on the eviction policy and put the objects into cache when they are accessed at the first time. This may lead "cache pollution" caused by cache space occupancy by objects which is request only once or a few times. To this end, some works [5], [6] focus on admission policy while improving the efficiency of eviction policy.

Over the past two decades, lots of approaches are proposed to improve the performance of classical methods or make dedicated network architectures effective and flexible in terms of caching. Especially, some works [4], [7]–[11] use machine learning or data-driven methods to get outstanding results. Considering the historical development of content caching and the new ways it may benefit from, we survey the existing caching strategies and discuss the problems when designing a potential caching strategy in the future.

The remainder of this paper is organized as follows. We first review the classical caching methods in Section II. We then survey the latest caching methods which is based on learning methods in Section III. Section IV discusses the design problems about datasets and implementation. We conclude this work in Section V

II. Classical Caching Approaches

In this section, we mainly review the classical caching approaches. In the earlier days, caching algorithms are originally derived from page replacement algorithms in operating systems, such as FIFO (First In First Out), LRU (Least Recently Used) and LFU (Least Frequently Used). After LRU and LFU are proposed, lots of approaches based on these two methods emerged, and some works take the content size into account, which are unfolded as follows.

A. LRU-based approaches

LRU replaces the least recently requested content when the cache is full, which can be implemented by an ordered queue. The most recently accessed content is always at the top of the queue. In other words, when the requested content is in the cache, it will be move to the top of the ordered queue. If the request content is not in the cache and there is no capacity for new arrivals, the content in the queue's bottom, which is the least recently accessed, will be evicted out of the cache.

The classical LRU method is easy to implement and has a low overhead. It works well in the random access occasion. However, it takes very few information about the access history of objects into account, and keeps considerable amount of objects with low popularity in the cache. This wastes the cache that might be used to accommodate more popular objects. To this end, LRU-K [12] is proposed to include the historical insights for LRU. LRU-K evicts the object which has the maximum Backward K-distance. The Backward K-distance $b_t\{p, K\}$ of object p at time t is defined as:

$$b_t\{p, K\} = \begin{cases} x \text{ , if case1} \\ \infty \text{ , if case2} \end{cases} \tag{1}$$

where case1 is: p is accessed at time $t - x$ and is accessed $K - 1$ times from time $t - x$ to time t, and case2 is: p is accessed less than K times until time t. When $K = 1$, LRU-1 is same as LRU. What's more, LRU-2 can achieve most of the advantages across the different K values. LRU-K fills the gap left by the LRU's neglect of access history, quickly removing the objects with very few request times.

However, the LRU-K method has a higher overhead to implement. Then 2Q [13] is proposed to achieve the similar hit ratio to LRU-K while exhibiting a low computational complexity. The simplified 2Q consists of two queues. One is a FIFO queue $A1$ which is a collaboration queue; the other one called Am, is used for cache list which is managed following the LRU rule. The new coming objects are first placed in the queue $A1$, when an object is accessed and it is still in $A1$, then put the target object into the cache list Am. 2Q focuses on both the admission policy and the eviction policy, and it can be implemented with constant time overhead.

LRU-K and 2Q require selecting tunable parameters offline, and the performance depends on the selected parameters. To this end, an online self-tuning method called adaptive replacement cache [14] (ARC) are developed, which achieves a low overhead by maintaining two LRU lists. The first LRU list $L1$ contains contents that have been requested only once recently. The second LRU list $L2$ contains contents that have been requested at least twice recently. Briefly, the main idea of ARC is to keep m top contents from list $L1$ and $c - m$ top contents from list $L2$ in the cache, where c is the cache capacity, $0 \leqslant m \leqslant c$. ARC tries to adjust m adaptively according to the request characteristics of the contents which were recently evicted from the cache.

B. LFU-based approaches

Instead of evicting contents according to the recency, LFU evicts the least frequently accessed content out of cache when the cache is full. LFU is widely used when the object access distribution conforms to a power law [15], because it stores contents with high popularity in the cache. LFU usually performs well over a fixed object access distribution and a fixed requested object set [16], [17]. However, it is not adaptable to the traces whose request characteristics and request object set vary significantly over time. For example, assume that the requested objects set is $\{A, B, C, D\}$ at time t, where each letter stands for a requested object. While at time $t + \Delta t$, the set changes to $\{E, F, G, H\}$. In this case, LFU may work even worse than LRU. Further more, LFU is difficult to implement with low overhead as it needs the statistical information about the access history of all the objects requested [18]. This is because LFU needs to maintain large and complex meta-data and requires a logarithmic implementation complexity in cache size, making LFU take a long time to make caching decisions with a large cache capacity.

To this end, In-Memory LFU [19] and WLFU [18] are proposed to improve the performance of LFU in various cases. In-Memory LFU takes only the request history of the content in cache into consideration, alleviating the high cost of maintaining the request history of all contents. On the other hand, WLFU adds a time window to the LFU methods, making LFU to focus on the request history of the last n requests, where n is the length of the window.

In addition, LRFU [20] is proposed to leverage the CRF (Combined Recency and Frequency) value to take both recency and frequency into consideration when evicting an object out of cache. The CRF value of content p which has been accessed k times at time t is the weighted sum of the time spans, noted as $CRF(p,t)$:

$$CRF(p,t) = \sum_{i=1}^{k} W(t - t_i) \tag{2}$$

Where t_i is the time when content p is accessed the i th time, W is the weighted function. Generally, in order to give more weight to recent requests, the weighted function is a monotonically nonincreasing function. LRFU can choose the proper weight function to make a trade-off between recency and frequency.

C. *Size-aware approaches*

Most of the methods aforementioned assume that all contents have the same size and the cache capacity is a constant. While in reality, contents have different sizes and the number of objects that the cache can store also changes as the sizes of contents change. To this end, LRU-S [21] is used to move the requested file based on LRU queue with a probability related to the file size. Adaptsize [5] is proposed to set a dynamic threshold of file size to decide which file is to be cached using a Markov chain model combined with a LRU queue. This method admits the objects with probability $e^{-size/c}$, and adjusts the parameter c online based on the Markov model. Adaptsize is in favor of admitting small size objects to increase hit ratio under a size-limited cache. If there are more popular big-size contents, it may not contribute noticeably to traffic savings.

III. Learning-based approaches

In this section, we briefly investigate the emerging learning-based approaches over the last two years, as shown in Table I. Most of them focus the caching strategies applied in CDN edge servers and the base stations (BSs) in wireless networks. Until 2014, there is no particular caching strategies for CDNs, most of which were still using commonly methods like LRU and LFU, or their variants [22]. But the traditional methods face big challenges with the dramatic growth of data through CDNs and the fast-changing access characteristics. Recently, machine learning is promising due to the rapid development of computer hardwares. Especially, deep learning had made a lot of significant achievements in various fields. which motivated the researchers try to apply the learning methods to resolve network problems [23]. Most of these researches use machine learning to solve congestion control and routing tasks. At the same time, there emerged some learning-based caching strategies [4], [7]–[11].

According to the problem formulation of the works, we briefly divide them into two classes. One is the *Popularity Prediction*, where the main idea of these approaches is using learning-based methods to estimate the popularity of contents in the future and guide the caching decision of classical caching strategies such as LRU. Another is the *Policy Learning*, which uses learning-based algorithms make caching decisions, such as the content sharing between nodes and whether to cache the requested content.

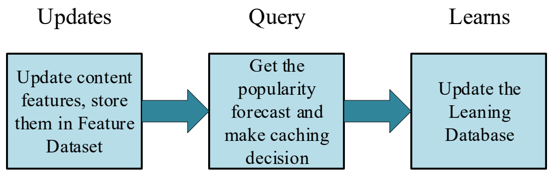
Updates
Query
Learns
Update content features, store them in Feature Dataset
Get the popularity forecast and make caching decision
Update the Leaning Database

Fig. 1. Workflow of the PopCaching system.

A. *Popularity Prediction*

To estimate the future popularity of contents is the most common idea when design a caching strategy. In this way, the caching performance is limited to the popularity prediction accuracy. Classical approaches always apply the simple statistic information of historical requests to estimate popularity, which suffers from the poor performance because of the various request patterns and dynamic users' behaviors. To this end, some works [4], [7], [10], [11] use learning based methods to get a better prediction. PopCaching [7] is proposed to leverage the online popularity prediction to make a better caching replacement decision. It is a caching system which has two more modules than the conventional cache node: the Feature Updater and the Learning Interface, and two more databases: Feature Database and Learning Database. Fig. 1 shows the workflow of PopCaching. It contains three steps. The first step is Updates. When there is a new request, the features of the requested object are updated and stored into the Feature Database. Then, PopCaching requests the popularity prediction from the Learning Database and decides whether to cache the target object according to the prediction. Finally, when the real popularity of the target object is revealed, the *content-popularity* pair will be store in the Learning Database. This is a online learning process and it dose not need the training phase. PopCaching uses a simple learning method based on the history of the requested objects and the partition of the feature space. The features it collects could be the user's profile and the property of contents. Note that PopCaching does not directly learn the popularity of each object but the relationship between the objects which have similar access patterns. PopCaching is a complete caching system, however it is difficult to deploy based on the subsistent cache node. And its running speed is faster than LFU while much slower than LRU.

In [4], an adaptive scheme is proposed for the YouTube content caching in Cellular Networks. The main idea of this approach is using extreme learning machine [25] (ELM) to predict the popularity of contents, and use mixed-integer linear programming (MILP) to decide where to place the content. Moreover, this work compare the performance of ELM to other popular machine learning methods in popularity estimation. The ELM model with optimal neurons decided by stochastic

TABLE I
THE OVERVIEW OF LEARNING-BASED CACHING STRATEGIES IN OUR INVESTIGATION

Name	Year	Application Scenario	Main Method	DataSet	Comparison Methods	Information Exchange Between Nodes
†Li *et al.* [7]: PopCaching	2016	CDN	Popularity forecast	moive.douban.com 38 million requests	FIFO LRU LFU LFUDA MIN [24]	No
†Tanzil *et al.* [4]	2017	Cellular Network	ELM [25] MILP SPSA [26]	NS-3 simulator Youtube dataset: 12500 videos	FemtoCaching [27] Bastug *et al.* [28] David *et al.* [29]	No
‡Song *et al.* [8]	2017	Wireless Network	MAB ADMM [30]	synthetic data sets: 20 unique objects	LFU LRU Reference algorithm [31]	Yes
‡Zhong *et al.* [9]	2017	Wireless Network	Wolpertinger Policy [32] DDPG [33]	synthetic data set: 5000 unique objects	LRU LFU FIFO DQN-based [34]	No
†Narayanan *et al.* [10]: DeepCache	2018	CDN	LSTM [35] Seq2Seq [36]	2 synthetic data sets: 50 unique objects 1425 unique objects	LRU k-LRU [37]	No
†Li *et al.* [11]: MFLRU KnnDyn	2018	CDN	Recommendation algorithms: Matrix Factorization [38] Collaborative Filtering	VoD sevice requests	LRU FiF [24]	Yes

†Popularity Prediction, ‡Policy Learning

perturbation simultaneous approximation [26] (SPSA) shows relatively high accuracy with a low running time.

In order to make full use of the access history to predict the next popularity distribution, DeepCache [10] is proposed using the Seq2Seq [36] model to predict the content popularity, which leverages the temporal characteristics of the requests. The main idea of DeepCache is to use the deep learning networks to make more accurate prediction for traditional caching strategies like LRU. DeepCache contains an object characteristic predictor which is composed of an Encoder-Decoder Model. The input of the predictor is the time sequence of probability vectors of all unique contents. In each vector, the request probability of every object is computed across a sliding window. Moreover, the probability is calculated in multi time scales. The output of the predictor is the request probability of each object. After predicting the popularity, *i.e.*, the request probability, DeepCache uses it to guide the caching decision of LRU. In order to interoperate with LRU without making caching system structure changes, the authors make "fake requests" according to the output of predictor as the input request trace of LRU.

Unlike DeepCache, Li *et al.* [11] proposes two data-driven approaches, using recommendation algorithms to predict the probability of contents in the next time slot. In this work, each edge server is considered as a super user, *i.e.*, a group of the users served by this edge server. According to the access log of the super user to all the contents, MFLRU (Matrix Enhanced LRU Caching Strategy) uses Matrix Factorization [38] (MF) to infer the future preference for the contents that have not been accessed by this super user. MF methods is computationally expensive which makes it difficult to make prediction in real time. MFLRU combines MF and LRU together to balance the speed of LRU and the high performance of the MF. KnnDyn is then proposed in this paper to overcome the high cost of MF. It use the K-Nearest-Neighbor (KNN) collaborative filtering to make predictions in order to capture the smaller timescale changes in the request characteristics.

B. Policy Learning

Unlike *Popularity Prediction*, *Policy Learning* mainly assumes the future popularity of contents is unknown, making decisions based on the requests observed and the caching performance at the current time. For wireless networks, Song *et al.* [8] propose to estimate the popularity distributions of contents according to the historical requests, and then use multi-armed bandit (MAB) learning to decide the content caching and sharing between BSs. The workflow of content request in wireless networks is similar to that in CDNs. BSs are connected by links with each other. If a user requests a content and the BS stores the targeted content, it will send the content to the user directly. If there is no copy in the local BS, the BS retrieves the requested content from other BSs or CDN server. Then the caching problem is formulated as to maximize the caching reward at each time-slot, where the reward is given by the difference between the revenue of content caching and the costs of content sharing. This is a classical reinforcement learning problem. Two methods are proposed, *i.e.*: the centralized algorithm named single player MAB (SPMAB), and the decentralized algorithm named multi-player MAB (MPMAB). The latter one is designed to reduce the computational complexity of SPMAB. In order to achieve rapid convergence, MPMAB limits the BSs to exchange local information only with their neighbor BSs.

Considering that the optimal caching usually depends on the future content access characteristics, there is no deterministic solution at current moment. The execution process of the caching algorithm is, in a sense, very similar to the interaction between the agent and the external environment in deep reinforcement learning (DRL). Therefore, Zhong *et al.* [9] proposed to use DRL to make the caching decisions. This method takes the total access number of each content in the cache at three historical time scales in each time slot as the states. The action is defined as replacing one of the content in the cache with current requested content or not cache the content. So if the cache capacity is C, there are total $C + 1$ candidate actions. The reward consists of a short-term and a long-term reward, both of which are in proportion to the access frequency of the current requested content in future. Considering the running time and the adaptation to the variety of the request patterns, the Wolpertinger architecture [32] is employed. Compared to DQN-based (Deep Q-Network) method [34], the proposed approach achieves competitive performance and has a lower runtime. This DRL-based method is novel for content caching, however it assumes that all contents have the same size and the cache capacity is constant, which does not match the actual situation of real caching systems.

IV. Remaining issues about content caching

In this section, we discuss some problems that might be encountered when design a flexible caching system.

A. Dataset

Nowadays, content caching plays a very important role in various scenarios. In order to guarantee the generalization performance of caching algorithms, it is important to choose a reasonable dataset for evaluation. Previously, researchers focused on synthesizing datasets by specifying request distributions. In order to apply the caching strategies to the existing system, especially in the case of the caching performance is obviously affected by user behaviors, more and more works focused on synthesizing datasets based on real requests or directly using real request records to evaluate caching algorithms. The most common way to synthesize a dataset is to generate corresponding content requests based on the Zipf-like distribution [15], or other probability models.

Synthesizing realistic request data is usually based on the accurate workload characterization. Tang *et al.* [39] proposed a publicly available streaming media workload generator according to the analysis on two long-term traces of streaming media. Summers *et al.* [40] discussed the methodologies for generating HTTP streaming video workloads. Curiel *et al.* [41] discussed the workload generation approaches in details. There were many researches focusing on the users' behaviors and content request characteristics [42]–[44].

Considering user's privacy and other issues, it is difficult to gain a open dataset involving requests from real system. There are some works using real-world datasets from CDNs or websites, but they are not publicly available. Some other works give ideas for the acquisition of datasets. Tanzil *et al.* [4] used YouTube API to collect features to estimate the viewcount of a YouTube video. Chang *et al.* [45] mentioned an open dataset from the MOMENTUM projects. And PopCaching [7] used crawler to get dataset from websites.

B. Simulation Platform

We find that there are very few caching strategies are evaluated in a real system. Instead, most of them are evaluated by simulations with certain assumptions. It is worth noting that, Cliffhanger [46] is implemented on Memcached [47] and tested in Memcachier [48] applications. Adaptsize [5] is implemented on top of Varnish [49] which is a production caching system. It is also evaluated in a dedicated data center and compared with production caching system such as Varnish and Nginx. Thus, if we can implement our methods in real system and test it with real-world datasets, it will bring more reliable and trustworthy guidance to our researches. The evaluation of caching algorithms is an interesting issue that we can explore.

V. Conclusion

In this paper, we mainly review the classical caching methods and the latest learning-based caching approaches. In the past two decades, lots of methods are proposed to solve the caching problems in operating systems and content delivery tasks. Most of them are based on recency, frequency and content size. Recently, various machine learning methods have come back to people's view due to the advances of hardware capability, resulting in some learning-based caching strategies leveraging deep learning, reinforcement learning and recommendation algorithms. These methodologies use learning-based methods to predict content popularity in future or to learn caching policy. Besides, we also discusses the potential issues such as datasets and implementation. We hope this paper can be a reference for further researches.

Acknowledgment

This work was supported in part by the National Key Research and Development Program under Grant No. 2016YFB1000102, in part by the National Natural Science Foundation of China under Grant No. 61571215, 61672318, 61631013, and in part by the Fundamental Research Funds for the Central Universities under Grant No. 2018CDXYRJ0030. Corresponding authors: Xu Zhang and Zhan Ma.

References

[1] V. Cisco, "Cisco visual networking index: Forecast and methodology 2016–2021.(2017)," 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[2] H. Yin, X. Zhang, S. Zhao, Y. Luo, C. Tian, and V. Sekar, "Tradeoffs between cost and performance for cdn provisioning based on coordinate transformation," *IEEE Transactions on Multimedia*, vol. 19, no. 11, pp. 2583–2596, Nov 2017.

[3] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 6, pp. 2001–2014, 2013.

[4] S. S. Tanzil, W. Hoiles, and V. Krishnamurthy, “Adaptive scheme for caching youtube content in a cellular network: Machine learning approach,” *Ieee Access*, vol. 5, pp. 5870–5881, 2017.

[5] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, “Adaptsize: Orchestrating the hot object memory cache in a content delivery network.” in *NSDI*, 2017, pp. 483–498.

[6] B. M. Maggs and R. K. Sitaraman, “Algorithmic nuggets in content delivery,” *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 52–66, 2015.

[7] S. Li, J. Xu, M. Van Der Schaar, and W. Li, “Popularity-driven content caching,” in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[8] J. Song, M. Sheng, T. Q. Quek, C. Xu, and X. Wang, “Learning-based content caching and sharing for wireless networks,” *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4309–4324, 2017.

[9] C. Zhong, M. C. Gursoy, and S. Velipasalar, “A deep reinforcement learning-based framework for content caching,” in *Information Sciences and Systems (CISS), 2018 52nd Annual Conference on*. IEEE, 2018, pp. 1–6.

[10] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, “Deepcache: A deep learning based framework for content caching,” in *Proceedings of the 2018 Workshop on Network Meets AI & ML*. ACM, 2018, pp. 48–53.

[11] G. Li, Q. Shen, Y. Liu, H. Cao, Z. Han, F. Li, and J. Li, “Data-driven approaches to edge caching,” in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*. ACM, 2018, pp. 8–14.

[12] E. J. O’neil, P. E. O’neil, and G. Weikum, “The lru-k page replacement algorithm for database disk buffering,” *Acm Sigmod Record*, vol. 22, no. 2, pp. 297–306, 1993.

[13] D. Shasha and T. Johnson, “2q: A low overhead high performance buffer management replacement algoritm,” in *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago, Chile*, 1994, pp. 439–450.

[14] N. Megiddo and D. S. Modha, “Arc: A self-tuning, low overhead replacement cache.” in *FAST*, vol. 3, no. 2003, 2003, pp. 115–130.

[15] A. Clauset, C. R. Shalizi, and M. E. Newman, “Power-law distributions in empirical data,” *SIAM review*, vol. 51, no. 4, pp. 661–703, 2009.

[16] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” in *INFOCOM’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.

[17] D. N. Serpanos and W. H. Wolf, “Caching web objects using zipf’s law,” in *Multimedia Storage and Archiving Systems III*, vol. 3527. International Society for Optics and Photonics, 1998, pp. 320–327.

[18] G. Karakostas and D. N. Serpanos, “Exploitation of different types of locality for web caches,” in *Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on*. IEEE, 2002, pp. 207–212.

[19] S. Podlipnig and L. Böszörmenyi, “A survey of web cache replacement strategies,” *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[20] 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 transactions on Computers*, no. 12, pp. 1352–1361, 2001.

[21] D. Starobinski and D. Tse, “Probabilistic methods for web caching,” *Performance evaluation*, vol. 46, no. 2-3, pp. 125–137, 2001.

[22] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, “Revisiting caching in content delivery networks,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1. ACM, 2014, pp. 567–568.

[23] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, “Machine learning for networking: Workflow, advances and opportunities,” *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2018.

[24] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[25] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32–48, 2015.

[26] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[27] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “Femtocaching: Wireless content delivery through distributed caching helpers,” *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[28] E. Baştuğ, M. Bennis, and M. Debbah, “Proactive caching in 5g small cell networks,” *Towards 5G: Applications, Requirements and Candidate Technologies*, pp. 78–98, 2016.

[29] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, “Optimal content placement for a large-scale vod system,” in *Proceedings of the 6th International COnference*. ACM, 2010, p. 4.

[30] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[31] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, “Learning distributed caching strategies in small cell networks,” in *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*. IEEE, 2014, pp. 917–921.

[32] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, “Deep reinforcement learning in large discrete action spaces,” *arXiv preprint arXiv:1512.07679*, 2015.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[35] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[37] M. Garetto, E. Leonardi, and V. Martina, “A unified approach to the performance analysis of caching systems,” *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 1, no. 3, p. 12, 2016.

[38] M. Jamali and M. Ester, “A matrix factorization technique with trust propagation for recommendation in social networks,” in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 135–142.

[39] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, “Medisyn: A synthetic streaming media service workload generator,” in *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. ACM, 2003, pp. 12–21.

[40] J. Summers, T. Brecht, D. Eager, and B. Wong, “Methodologies for generating http streaming video workloads to evaluate web server performance,” in *Proceedings of the 5th Annual International Systems and Storage Conference*. ACM, 2012, p. 2.

[41] M. Curiel and A. Pont, “Workload generators for web-based systems: Characteristics, current status, and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1526–1546, 2018.

[42] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, “Understanding user behavior in large-scale video-on-demand systems,” in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4. ACM, 2006, pp. 333–344.

[43] T. Qiu, Z. Ge, S. Lee, J. Wang, Q. Zhao, and J. Xu, “Modeling channel popularity dynamics in a large iptv system,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 275–286.

[44] Z. Han, J.-A. Ma, and H. Zhao, “An access pattern model analysis for online vod multimedia flow for bigdata,” in *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*. IEEE, 2018, pp. 34–38.

[45] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, “Learn to cache: Machine learning for network edge caching in the big data era,” *IEEE Wireless Communications*, vol. 25, no. 3, pp. 28–35, 2018.

[46] A. Cidon, A. Eisenman, M. Alizadeh, and S. Katti, “Cliffhanger: Scaling performance cliffs in web memory caches.” in *NSDI*, 2016, pp. 379–392.

[47] “Memcached,” https://github.com/memcached/memcached/wiki/DevelopmentRepos.

[48] “Memcachier,” https://www.memcachier.com/.

[49] P. Graziano, “Speed up your web site with varnish,” *Linux Journal*, vol. 2013, no. 227, p. 4, 2013.