SNN-Cache: A Practical Machine Learning-based Caching System Utilizing the Inter-relationships of Requests

Youngbin Im*, Prasanth Prahladan*, Tae Hwan Kim†, Yong Geun Hong†, Sangtae Ha*
*Department of Computer Science, University of Colorado Boulder
Boulder, CO, 80309-0430, USA
Email: {youngbin.im, prasanth.prahladan, sangtae.ha}@colorado.edu
†Electronics and Telecommunications Research Institute
Email: {thkimetri, yghong}@etri.re.kr

***Abstract*—An efficient caching algorithm needs to exploit the inter-relationships among requests. We introduce SNN, a practical machine learning-based relation analysis system, which can be used in different areas that require the analysis of relationships among sequenced data such as market basket analysis and online recommendation systems. In this paper, we present SNN-Cache that leverages SNN to utilize the inter-relationships among sequenced requests in caching decision. We evaluate SNN-Cache using an Information Centric Network (ICN) simulator, and show that it decreases the load of content servers significantly compared to a recent size-aware cache replacement algorithm (up to 30.7%) as well as the traditional cache replacement algorithms.**

I. Introduction

Caching has been widely used in various computing systems to enhance the performance of the system. Modern CPUs have a multi-level cache hierarchy to reduce the cost (in terms of time or energy) of accessing data in the main memory. A web cache is a technology for the temporary storage of web documents close to the users to reduce the latency of retrieving them from the remote server. In the datacenter, the results of common database queries are stored in the caching servers for faster access of subsequent identical queries (e.g., Memcached [12] and Redis [13]). Information Centric Network (ICN) is also designed to utilize the network caches in order to reduce network traffic for the duplicate content requests.

An efficient caching algorithm needs to utilize the inter-relationships between sequenced requests as these requests are typically correlated. For example, web users tend to request web contents related with previously requested contents. Requests also have temporal locality; data in specific memory regions are often accessed repeatedly. However, precisely understanding the relations among data is difficult and time-consuming due to a huge amount of data requests in general caching systems. Therefore, simple Least Recently Used (LRU) or Least Frequently Used (LFU) based algorithms have been widely used in practice.

These days, machine learning (ML) is widely used to solve problems in many different areas of computer systems. However, no prior work has utilized this ML approach to analyze the request patterns for obtaining the inter-relationships between sequenced requests and apply them to make an efficient caching system. Association rule learning is an representative technology for the inter-relationship analysis of a large dataset, which is based on a rule-based machine learning approach that discovers interesting relations between variables, using support, confidence, lift, interest-support, cross support as measures of significance and interest for the association rule [25]. Sequential pattern mining is a method which finds statistically relevant patterns among sequentially delivered data [21]. These approaches are mostly post-processing methods which are performed after the whole data are collected, and not suitable for applications that require real-time processing such as caching systems.

We present Stimulable Neural Network (SNN), a practical machine learning-based relation analysis system that analyzes the inter-relationships among sequenced requests in real-time. SNN considers a trade-off between the high accuracy and low computational overhead in the analysis. To show the efficacy of SNN, we present SNN-Cache, an example application which utilizes SNN to find the inter-relationships among data requests to make an efficient caching decision for ICN. The contributions of this paper are as follows.

1) New approach to the caching system. Unlike previous caching approaches based on recency, frequency, and cost, SNN-Cache analyzes the relationships among data to make caching decisions. This new approach can utilize the previously unknown information on time-varying relations of data, and incorporate different factors to the caching decision such as content size and data retrieval cost.

2) Evaluation using realistic data. We evaluate SNN-Cache using an ICN simulator and a large content name dataset, and show that it significantly decreases the load on content servers in comparison to (i) a recent size-aware cache replacement algorithm (by up to 30.7%), (ii) traditional cache replacement algorithms and (iii) ICN caching decision algorithms.

978-1-5386-0579-0/18/$31.00 ©2018 IEEE

The rest of this paper is organized as follows. Section II overviews the related work in caching systems. Section III presents the architecture and operations of SNN, and its applications including SNN-Cache. Section IV evaluates the performance of SNN-Cache by comparing it with various other algorithms. Section V concludes the paper.

II. Related Work

The landscape of caching work can be understood by categorizing algorithms based on their choice of measures for caching decisions. For example, one can categorize the caching work into recency-based approaches ([11], [24], [33]), frequency-based approaches, hybrid-approaches ([19], [23]), cost and function-based approaches ([6], [30], [32]). All of these algorithms rely on imposing a partial or complete ordering of the elements stored in the cache, by the use of appropriate data-structures.

Another categorization of caching algorithms could be based on the data structure that is used to organize the cached elements. While most algorithms are based on queues, lists or trees, there exist a few algorithms like Hyperbolic Caching [7] that uses a set representation of the cached items. These algorithms have an implicit ordering of the elements obtained by the computation of a priority function of each item. The choice of the data-structure thus impacts the flexibility and the extensibility of the caching strategy.

Based on the nature of caching policy, the caching strategies can be classified as follows: 1) static policies: the caching-decision is based on the same criteria for all requests and for all time, 2) dynamic policies: the caching-decision is based on criteria that varies over time, but lies within a finite set of pre-defined values, 3) learning policies: the caching-decision is based on a criteria that varies over time, and assumes values that are learned based on past-experiences. The parameters that define the policy are not constrained to a finite set of pre-defined values. We shall now proceed to illustrate this classification with associated previous works.

Most of the earliest cache replacement algorithms were static policies. These were primarily of the type that measured temporal locality of the incoming requests based on ”recency”, ”frequency” or a hybrid combination of the two. This measure was used to create a partial or complete ordering of the cached elements. Most of these algorithms were then modified by the introduction of measures of cost or the definition of a priority-function. Some algorithms like GreedyDual [33] uses priority queues for size-aware caching in web proxies and GDWheel [20] uses a more efficient wheel data structure. Other algorithms like RIPQ [31] use the size of content items as a measure of cost, while some algorithms proposed the use of expiration-time [1], [5] and freshness [30].

Dynamic caching algorithms are designed to dynamically select between a set of caching policies. Some of the early implementations used multiple-queue implementations, starting with the LRU-2 policy. Other works employing multiple queues include LRU-K [24], 2Q [16], MQ [34], LIRS [15] and ARC [23]. Several of these algorithms such as [23] incorporate ghost caches, which track information about items which are no longer in the cache.

Learning policies refer to a class of algorithms that started off with the publication of the Adaptive-LRFU [18], which helped realize a caching strategy that could implement any policy lying between the two extremes of LRU and LFU. In essence, this class of policies generally comprise of a tuning-parameter, that needs to be set either off-line or on-line, which then determines the policy that shall be implemented on the incoming workload. The caching policy is informed by the characteristics of the workload rather than being defined by a heuristic. Some of the recent learning policies use ghost-queues to learn the parameters that define the policy in an on-line manner. For example, ARC automatically tunes the queue sizes of an LRU-2-like configuration. Other algorithms like Hawkeye [14] use memory and a simulation of an optimal algorithm, to guide the learning policy.

III. SNN: Stimulable Neural Network

SNN is a machine learning approach that analyses the inter-relationships among sequenced data in real time and with low computational complexity. SNN can be used in various applications that require real-time relation analysis including ICN caching, market-basket analysis and on-line recommendation systems. In this section, we provide an overview of SNN based caching strategy - the architecture and constituent operations. This shall be followed by a description of a few illustrative applications of this technology.

A. Overview of SNN

SNN is inspired by the biological neural network in which a large number of connections are made among neurons to conduct complex computations. SNN captures the inter-relationships between the data items by using a set of real-valued matrix filters. As new data items enter into the cache, the data-names are used to identify areas within the filters that shall be updated. The values in the filters get updated with each incoming-request. In this way, these filters help account for the structural similarity between the data items and the temporal relationship in the occurrence of different data items.

B. Architecture and Operations

SNN consists of several modules: Data Reception Module, Data Preprocessing Module, Correlation Filter Module, and the Impulse Calculation Module.

1) Data Reception Module: Data Reception Module extracts the metadata which can be used for the analysis from the received data and conveys it to Data Preprocessor module. For a system that analyses the relationships among the network packets, the packet header information is extracted. The metadata extracted varies according to the application in which SNN is used.

2) Data Preprocessing Module: The metadata received from Data Reception Module is further processed so that it can be used by the Correlation Filter Modules. The Correlation Filter Modules can maintain multiple filters corresponding to

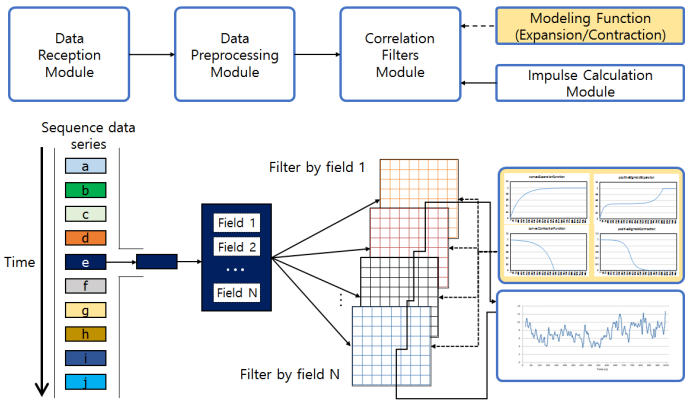

Fig. 1: Architecture of SNN.

each field of the metadata. In such case, the Data Preprocessing Module separates the metadata into required fields. In ICN caching, the name segments of the content name are extracted from the metadata. Additional information related to the incipient content can also be used to inform the update process within the filters. For example, within the ICN domain, the size of the content and the hop-count from the data-source are used as alternative measures of cost.

3) Correlation Filters Module: Correlation Filters Module stores the inter-relationships among data in encoded matrices. The number of filters depend on the number of fields extracted within the Data Preprocessing Module. If the metadata is separated into 3 fields, $f1$, $f2$, and $f3$, three Main Relation Filters for the self-relations of each field, $f1-f1$, $f2-f2$, and $f3-f3$, are created. In addition, three Cross-field Relation Filters that represent the relations between two fields, $f1-f2$, $f1-f3$, and $f2-f3$, are created. Finally, one additional filter, Data Identification Filter, for identifying the data itself is created. The definition of the filters and the assigned meaning to the indices are customizable to the domain in which SNN is utilized. When a new data is delivered, the corresponding values at filters are expanded by using the Expansion Function. The expansion is to reflect the increased incidence of the relationships between identified objects. The Contraction Function is used to contract all the values in all the filters at a periodic interval. The contraction is applied to reflect the decrease in importance of past patterns in relationships as time increases. These two functions should be monotonically increasing (Expansion Function) and monotonically decreasing (Contraction Function), respectively, with a maximum value of 1 and minimum value of 0. These two functions can be chosen according to the characteristics of the system domain.

When a new data arrives, the index of the matrix for the data (idx) is calculated by applying a hashing function (H) to the value of the corresponding field (f) and a modular operation using the filter size (n).

$$idx(f) = H(f) \ mod \ n \tag{1}$$

If the calculated index is c, the values at locations (x, c) and (c, y) where $1 \leq x \leq n$, $1 \leq y \leq n$ are updated by using the Expansion function. The Expansion Function can take into account the cost of the data. For example, the cost can be a content size in Web proxy or Content Delivery Network(CDN) node. For a data with the same priority, a larger data is more valuable than smaller one because it takes more resources to retrieve the larger data from the network or from a lower level in the cache hierarchy such as a disk. The cost can also be indicated by a hop count in an ICN network: the hop count reflects the traffic cost to retrieve the content from the Content Router or the Origin server. The Contraction Function can be applied periodically over a fixed time period (e.g., 30 seconds) or data-count period (e.g., 30 data items).

4) Impulse calculation Module: The relations of a datum with other data in a specific time point are calculated in this module. The relations of a datum with all other data are calculated by summing up all the values in the row or column that corresponds to the index of the datum for each filter and then adding up the values obtained from each filter. To reflect the relative importance of different filters, each filter is set with a constant positive weight (> 0). It is also possible to calculate the relation of a datum with a specific datum or a set of data. In this case, only the columns or rows that correspond to the considered data are used in the calculation.

C. Applications

1) SNN-Cache: SNN-based ICN content router: SNN can be applied to ICN content routers (CRs) as shown in Figure 2. CRs can learn the content request/delivery patterns and use them in the cache replacement policy. SNN-Cache can enhance not only the hit-ratio of a single content router, but also the hit-distance and traffic load over the network, by efficiently using the limited cache space through the elimination of redundant caching among content routers.

SNN-Cache can be implemented by extending the basic functions of SNN modules. The operations of the caching algorithm differ according to the type of the packet: Interest or Content. Interest packets are used for learning the content request pattern, while Content packets are used to make actual decisions on caching.

Data Reception Module sends the content name to Data Preprocessing Module along with the hop count information (from the content requester) additionally in the case of Interest pack-

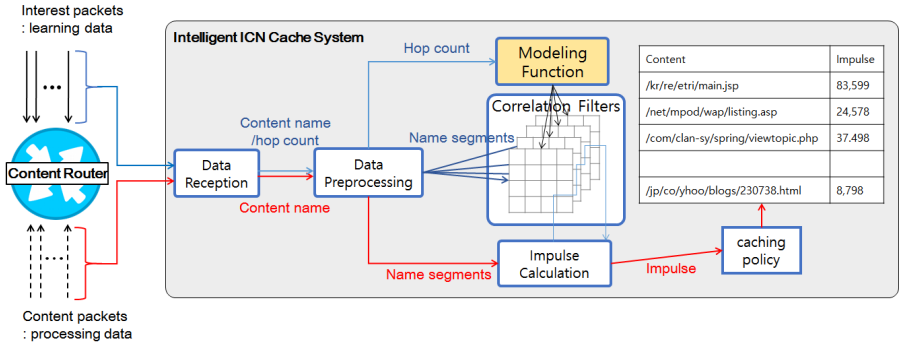

Fig. 2: Architecture of SNN-Cache.

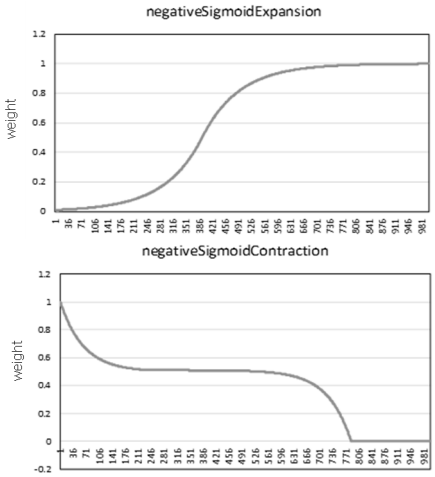

Fig. 3: The Expansion Function(top) and Contraction Function(bottom) for SNN-Cache.

ets. For an Interest packet, Data Preprocessor Module separates the content name into segments, and sends them to Correlation Filters Module with the hop count information. It divides the content name into 3 segments: Global routing name segment(GRNS), Application dependent name segment(ADNS), and Object dependent name segment(ODNS). For a content name /com/blogspot/cocinaconsilvia/almendras.html, the segments are divided as "/com/blogspot" (GRNS), "/cocinaconsilvia" (ADNS), and "/almendras.html" (ODNS). Three Main Relation Filters(MRF), three Cross-field Relation Filters(CRF), and one Data Identification Filter(DIF) are created for the three classes of name-segments. For a content packet, the name segments extracted are sent to the Impulse Calculator Module(ICM).

Correlation Filters Module updates its filters when it receives the hop-count and name segments for Interest packets according to the Expansion Function. The gradient of increment applied to the expansion function is inversely proportional to the hop count. This helps to increase the probability of caching content-items in the CRs close to the content requesters.

When Impulse calculation Module receives name segments for a content packet from Data Preprocessing Module, it calculates the impulse value for the content, which is sent to the Caching Policy Module. In the calculation of impulse value, we multiply the original impulse value by the content-size to reflect its load on the network traffic when serving the content. If the cache has an content with smaller impulse than the received impulse value, Caching Policy Module replaces that content with new content. We test various types of Expansion and Contraction Functions, and discovered that negative sigmoid function (shown in Figure 3) works best to reflect the variation of content request/delivery patterns.

2) Market basket analysis: SNN can be applied to the market basket analysis, an analysis of relations among products purchased in the market. In existing market basket analysis, the relations among sales of the products are derived by using statistical methods through data mining. Primary metrics are support $\left(P(X \cap Y)\right)$, confidence $\left(P(Y \mid X) = \frac{P(X \cap Y)}{P(X)}\right)$, and lift $\left(\frac{P(Y|X)}{P(Y)} = \frac{P(X \cap Y)}{P(X)P(Y)}\right)$. The support is the ratio of the sales that include the products X and Y among all sales records, while the confidence is the probability that the product Y is purchased when the product X is purchased. The lift is a metric that decides the independence of products X and Y. However, these methods are not suitable for the analysis of the time-varying relations in sales data. In addition, the complexity increases drastically when the number of products to observe is increased. SNN can be used for a real-time analysis of the relations among purchased products. When a purchase event occurs, the information is transmitted to the analysis system, and the relations among products stored at filters are updated. When a query for the relation of specific products is delivered, the analysis system returns the result by calculating the impulse value. By observing the traces of these relations, we can also analyse the temporal patterns of the sales data.

IV. EVALUATION

We evaluate the performance of SNN-Cache in this section.

A. Simulation settings

For the evaluation of the performances of different caching algorithms, we use Icarus, the caching simulator for ICN [28]. We implement the basic algorithm of SNN in C++ module, and connect the Icarus simulator with the SNN module by using

Setuptools [2] and SWIG [3]. We also implement the Hyperbolic Caching in Icarus' cache replacement policies modules. We calculate the priority of the content i by $\left(p_i = \frac{n_i \cdot s_i}{t_i}\right)$ where n_i is the request count for i since its entrance to the cache, t_i is the time from the entrance to the cache, s_i is the content size. We also add a performance metric for calculating per-node and network traffic on the Performance metrics loggers module in Icarus.

We test two scenarios: a single router scenario and network topology scenario. The first scenario is to look into the performance of our algorithm without the effect of the network topology, and also corresponds to a single caching system such as web cache and data center cache servers. For the single router scenario, we use a content request trace of 1,000 contents, 100 clients, and 20,000 content requests with the Zipf content size distribution of $\alpha = 1.0$ according to [29], and Zipf content popularity distribution of $\alpha = 0.6, 0.8, 1.0$. Measurements of [8] and [22] provide estimates for alpha parameter between 0.64 and 0.83. More recent results in [9] present a Zipf popularity with $\alpha \approx 0.88$. Based on the content names from [4], our request generator creates content requests for a set of given parameters such as number of contents, requests, α for Zipf content size distribution, α for Zipf content popularity distribution, etc. We compare the performance with a up-to-date cache replacement algorithm, Hyperbolic Caching [7] and traditional algorithms such as LFU, LRU, segmented LRU. We also compare with baseline algorithms such as random and FIFO. For the cache decision algorithm, we use LCE (Leave Copy Everywhere).

For the network topology scenario, we use the WIDE topology provided by Icarus. The WIDE topology includes 11 content servers, 6 clients, and 13 content routers. We also use the same settings for the content requests as the single router scenario with one exception of 6 clients. We compare the performance with different ICN cache decision algorithms, such as Symmetric Hash Routing, Asymmetric Hash Routing, Multicast Hash Routing, Asymmetric-Multicast Hybrid Hash Routing, Symmetric-Multicast Hybrid Hash Routing [27], Cache Less for More [10], ProbCache [26], LCD [17] and random algorithm which caches the content on a randomly selected node on the delivery path. We use LFU as the cache replacement algorithm for each of the alternative ICN cache decision algorithms mentioned above. While executing the SNN-Cache algorithm, we use LCE as the caching-decision algorithm.

B. Analysis

The result of single router scenario is shown in Figure 4. The content size-agnostic algorithms such as LFU, LRU, SLRU, RANDOM, FIFO increase the server traffic significantly compared to SNN-Cache. The Hyperbolic Caching algorithm reduces the server traffic considerably, but still it requires 6.6 to 30.7 % higher server traffic than SNN-Cache. We assert that the SNN-Cache algorithm is thus more effective in reducing the server traffic than the simple frequency- and size-based algorithm like Hyperbolic Caching. The α parameter affects

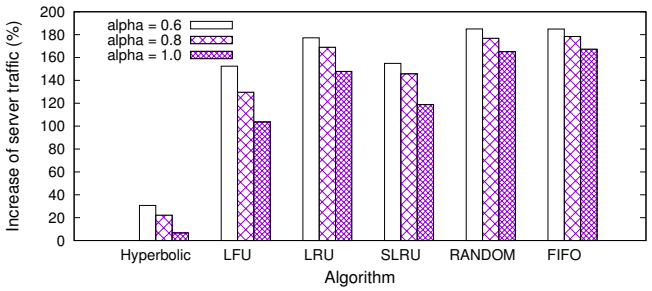

Fig. 4: Performance comparison in the single router scenario. The relative increases of server traffic compared to SNN-Cache are shown.

the performance of comparison algorithms. While SNN-Cache shows a similar ratio of the traffic amount by cache hit and the total traffic amount for different α parameters, it decreases in all other comparison algorithms as α increases. We claim that this is a good feature of SNN-Cache, because SNN-Cache achieves an efficient cache utilization regardless of the content popularity distribution.

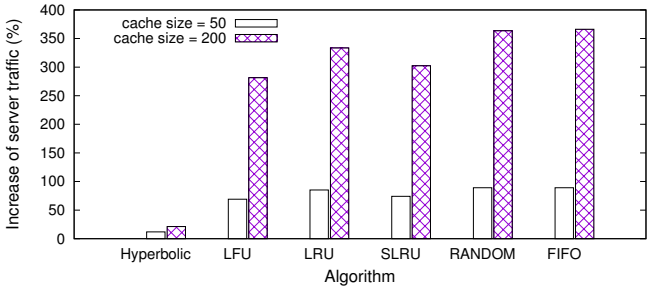

Fig. 5: Performance comparison with different cache sizes in the single router scenario. The relative increases of server traffic compared to SNN-Cache are shown.

We observe the performances of different cache replacement algorithms by varying the size of the cache entries in Figure 5. We run the experiments with the cache size of 50 and 200 in addition to the original size 100. The amount of the server traffic increase depends on the cache size. As the cache size becomes larger, SNN-Cache achieves a higher performance enhancement to the other cache replacement algorithms.

The result of WIDE network topology scenario is shown in Figure 6. All the tested cache decision algorithms show higher mean node traffic than SNN-Cache. While Symmetric Hash Routing, Multicast Hash Routing, Symmetric-Multicast Hybrid Hash Routing produce lower server traffic than SNN-Cache, they induce more traffic on the whole network for the content diversity on the cache nodes.

V. Conclusion

In this paper, we present a real-time, machine learning based approach, called SNN, for the analysis of inter-relationships among sequenced data. As an example application of SNN, we present SNN-Cache that exploits the inter-relationships among sequenced requests in caching decision. We evaluate

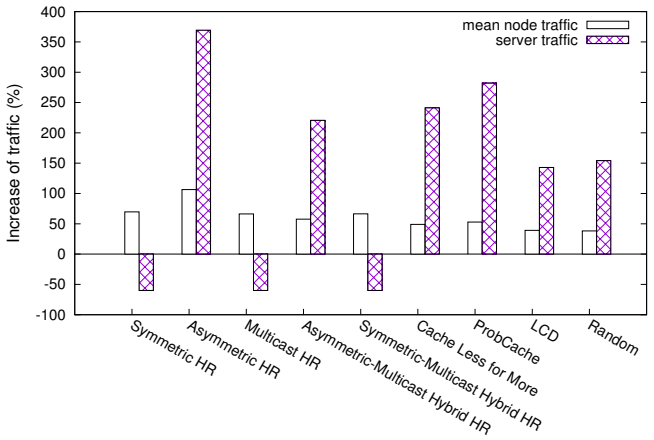

Fig. 6: Performance comparison in the WIDE network topology scenario. The relative increases of server traffic and mean node traffic compared to SNN-Cache are shown.

SNN-Cache using an ICN simulator, and show that it reduces the load of content servers significantly compared to not only the traditional cache replacement algorithms and ICN caching decision algorithms, but also the up-to-date size-aware cache replacement algorithm. As a future work, we plan to implement and evaluate other SNN-based systems such as market basket analysis and online recommendation systems.

ACKNOWLEDGMENT

This work was supported by the NSF under Grant CNS-1525435 and the National Research Council of Science & Technology (NST) grant by the Korea government (MSIP) (No. CRC-15-05-ETRI).

REFERENCES

[1] Multiworld Testing Decision Service. http://aka.ms/mwt.

[2] Official project repository for the Setuptools build system. https://github.com/pypa/setuptools.

[3] Simplified Wrapper and Interface Generator. http://www.swig.org/.

[4] The Content Name Collection. http://www.icn-names.net/.

[5] AGARWAL, A., BIRD, S., COZOWICZ, M., HOANG, L., LANGFORD, J., LEE, S., LI, J., MELAMED, D., OSHRI, G., RIBAS, O., ET AL. A multiworld testing decision service. *arXiv preprint arXiv:1606.03966* (2016).

[6] AGGARWAL, C., WOLF, J. L., AND YU, P. S. Caching on the world wide web. *IEEE Transactions on Knowledge and data Engineering 11*, 1 (1999), 94–107.

[7] BLANKSTEIN, A., SEN, S., AND FREEDMAN, M. J. Hyperbolic caching: Flexible caching for web applications. In *2017 USENIX Annual Technical Conference* (2017), USENIX Association, pp. 499–511.

[8] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., AND SHENKER, S. 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* (1999), vol. 1, IEEE, pp. 126–134.

[9] CARLINET, Y., KAUFFMANN, B., OLIVIER, P., AND SIMONIAN, A. Trace-based analysis for caching multimedia services. *Orange Labs Technical Report* (2011).

[10] CHAI, W. K., HE, D., PSARAS, I., AND PAVLOU, G. Cache less for more in information-centric networks. In *International Conference on Research in Networking* (2012), Springer, pp. 27–40.

[11] CORBATO, F. J. A paging experiment with the multics system. Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1968.

[12] FITZPATRICK, B. Distributed caching with memcached. *Linux journal 2004*, 124 (2004), 5.

[13] HTTPS://REDIS.IO/. Redis key value store database.

[14] JAIN, A., AND LIN, C. Hawkeye: Leveraging beladys algorithm for improved cache replacement.

[15] JIANG, S., AND ZHANG, X. Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review 30*, 1 (2002), 31–42.

[16] JOHNSON, T. 2q: A low overhead high performance buffer management replacement algorithm. *Proc. the 20th VLDB, 1994* (1994), 439–450.

[17] LAOUTARIS, N., CHE, H., AND STAVRAKAKIS, I. The lcd interconnection of lru caches and its analysis. *Performance Evaluation 63*, 7 (2006), 609–634.

[18] LEE, D., CHOI, J., KIM, J.-H., NOH, S. H., MIN, S. L., CHO, Y., AND KIM, C. S. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *ACM SIGMETRICS Performance Evaluation Review* (1999), vol. 27, ACM, pp. 134–143.

[19] LEE, D., CHOI, J., KIM, J.-H., NOH, S. H., MIN, S. L., CHO, Y., AND KIM, C. S. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE transactions on Computers 50*, 12 (2001), 1352–1361.

[20] LI, C., AND COX, A. L. Gd-wheel: a cost-aware replacement policy for key-value stores. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 5.

[21] MABROUKEH, N. R., AND EZEIFE, C. I. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR) 43*, 1 (2010), 3.

[22] MAHANTI, A., WILLIAMSON, C., AND EAGER, D. Traffic analysis of a web proxy caching hierarchy. *IEEE Network 14*, 3 (2000), 16–23.

[23] MEGIDDO, N., AND MODHA, D. S. Arc: A self-tuning, low overhead replacement cache. In *FAST* (2003), vol. 3, pp. 115–130.

[24] O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. The lru-k page replacement algorithm for database disk buffering. *ACM SIGMOD Record 22*, 2 (1993), 297–306.

[25] PIATETSKY-SHAPIRO, G. Discovery, analysis, and presentation of strong rules. In *Knowledge Discovery in Databases* (1991).

[26] PSARAS, I., CHAI, W. K., AND PAVLOU, G. In-network cache management and resource allocation for information-centric networks. *IEEE Transactions on Parallel and Distributed Systems 25*, 11 (2014), 2920–2931.

[27] SAINO, L., PSARAS, I., AND PAVLOU, G. Hash-routing schemes for information centric networking. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking* (2013), ACM, pp. 27–32.

[28] SAINO, L., PSARAS, I., AND PAVLOU, G. Icarus: a caching simulator for information centric networking (icn). In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques* (ICST, Brussels, Belgium, Belgium, 2014), SIMUTOOLS '14, ICST.

[29] SCHMEISER, S. The size distribution of websites. *Economics Letters 128* (2015), 62–68.

[30] SHIM, J., SCHEUERMANN, P., AND VINGRALEK, R. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering 11*, 4 (1999), 549–562.

[31] TANG, L., HUANG, Q., LLOYD, W., KUMAR, S., AND LI, K. Ripq: Advanced photo caching on flash for facebook. In *FAST* (2015), pp. 373–386.

[32] YANG, Q., ZHANG, H. H., AND ZHANG, H. Taylor series prediction: A cache replacement policy based on second-order trend analysis. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on* (2001), IEEE, pp. 7–pp.

[33] YOUNG, N. *Competitive paging and dual-guided on-line weighted caching and matching algorithms*. PhD thesis, Princeton University, 1991.

[34] ZHOU, Y., PHILBIN, J., AND LI, K. The multi-queue replacement algorithm for second level buffer caches. In *USENIX Annual Technical Conference, General Track* (2001), pp. 91–104.