PeCC: Prediction-error Correcting Cache

Adit Bhardwaj
Akamai Technologies
San Francisco
adbhardw@akamai.com

Vaishnav Janardhan
Akamai Technologies
San Francisco
vjanardh@akamai.com

Abstract

We propose using machine learning for predicting the future, request arrival sequence for similarly sized objects in cache, to simulate Belady's optimal caching scheme for small, but critical, in-memory caches. The proposed caching scheme, PeCC, is a low-cost and reliable scheme that gracefully accounts for errors in prediction to maximize the hit-rates. We show PeCC is able to bridge the wide gap in performance between classical online algorithms and Belady's for small caches.

1 Introduction

Large Internet platform companies and Content Delivery Networks (CDN) like Akamai rely on having many distributed points of presence (PoP). A large distributed caching platform achieves low-latency and high throughput for serving web and video content over the public Internet. With expanding content footprint sizes, machines with larger, but slower drives are deployed close to the end-users to maximize cache-hit rates for the long-tail. Edge-servers (ES) get deployed with two levels of cache, an application managed, in-memory cache called Hot Object Cache (HOC) Berger et al. [2017], and buffer-cache. Both caches offload the on-disk cache, whose read latency could often be more than the RTT to the end-users, making a hit in the HOC extremely valuable. The performance of in-memory caches is very crucial for the effectiveness of ES, as they not only help in improving the response times for objects in cache, but also improve the overall throughput of ES by reducing a number of expensive, small-object reads to disk Neglia et al. [2017].

Effectively caching objects in HOC is challenging, since the size of the content footprint is typically very large compared to the size of in-memory cache, making the caching decision by any online algorithm like LRU to be often wrong. As you can see in the Fig 1 there is wide gap between the cache-hit rates achievable for the same traffic if we compare an online algorithm like LRU with Belady's, Belady [1966] the theoretically optimal algorithm for maximizing cache-hit rates.The large performance difference between LRU and Belady's arises because LRU uses past inter-arrival time as the future inter-arrival time gap, which may or may not be true and has no such information about the objects it hasn't previously seen. When compared to Belady's, which uses the perfect knowledge of the future, at small cache-sizes, any Caching Decision Error (CDE) can lead to a cache-miss. CDE is defined as a caching decision that led to a lowered hitrate because of the decision to cache an object over others. The CDE's will tend to increase when we observe objects with no history like "one-hit-wonders", Maggs and Sitaraman [2015]. One-hit wonders, often tend to occupy around three quarters of the cache space Maggs and Sitaraman [2015], making them vulnerable to CDE's. Different classical caching algorithms like LRU, LFU, S4LRU Huang et al. [2013] have been proposed to work well over specific traffic profiles, but no single algorithm works well across all traffic profiles Busari and Williamson [2002], except for Belady's.

With smaller caches, the problem is more pronounced as we tend to operate in the steeper parts of footprint descriptor (FD) Sundarrajan et al. [2017] or miss-ratio curves, making them more vulnerable to CDE's as compared to when operating in the flatter parts of FD curves. Classical online algorithms assume each individual object's future inter-arrival will repeat the previously observed pattern, which is not always true Traverso et al. [2013a] and ignore any group dynamics of how related objects tend to behave similar, over their lifetimes. Such group behavior can be exploited for caching through predictive schemes. In spite of their known shortcomings, classical caching algorithms and its variants are quite popular, because of their low overhead and ease of use. Any new algorithm needs to bridge the gap with Belady's while outperforming classical algorithms at comparable cost overheads.

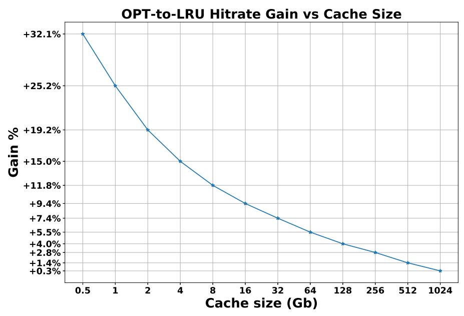

Figure 1: The gains of OPT over LRU decreases with cache size

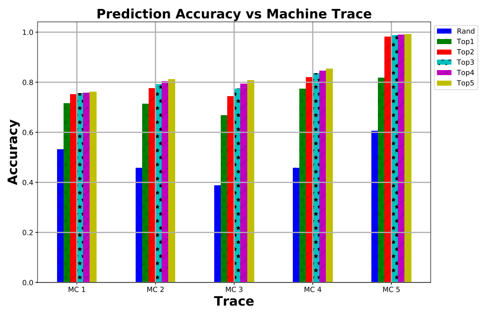

Figure 2: Prediction accuracy of top-k predictions in the proposed ML scheme compared with a random, class-distribution aware predictor. Figure shows top k = [1,2,3,4,5] predictions results. Large improvement in accuracy, from random predictor, suggesting predictability of request interarrivals.

2 Previous work

There have been many proposals in the past to effectively cache objects in RAM using online algorithms in Berger et al. [2017] and Neglia et al. [2017] but they are only able to improve the hitrates marginally as compared to the wide gap between classical schemes and Belady's. Until recently caching schemes using future arrival times were relegated to theoretical exercises, as predicting future was found to be nearly impossible for general caching, even though using future arrival time was explored for cpu caches by Jaleel et al. [2010] and Jain and Lin [2016]. Other work that tries to use any ML techniques for caching Lykouris and Vassilvitskii [2018], Hasslinger et al. [2018] tries to use the future popularity and advise on what objects to keep, without attempting to achieve hitrates close to the optimal. Again schemes like Vietri et al. [2018] tries to reduce regrettable evictions without aiming for the optimal hitrates if the prediction are accurate. Each one of the proposed solutions could perform well for certain traffic profiles. However, Akamai's ES network tends to have a dynamic mix of well known traffic patterns, observable only at the level of individual customers. The combined traffic pattern at an ES is more complex than any individual pattern and is constantly changing, rendering any know classical algorithm insufficient for our needs. Belady's on the other hand is optimal for all traffic mixes.

Belady's is not know to be optimal for variable object size caching, but for HOC, capping the max file size allowed to enter Berger et al. [2017], makes Belady's a good approximation for optimal. We exploit a few properties of Belady's to greatly simplify the prediction scheme; in Belady's we don't need the future popularity, but only need to know 1) the relative, future sequence of requests for objects, 2) only needs precise sequencing in the time series, where the evictions are happening and 3) the relative sequencing is only on objects that are currently in cache and not for any new object in the future. We use these simplified requirements for Belady's and exploit the similarity in behaviour among objects in traffic classes to predict the relative sequencing in our caching scheme called Prediction-error Correcting Cache (PeCC).
Even a well trained ML systems is bounded by the Bayes error rate. The rate of overall errors drops significantly when we consider top k predictions in a multi-class classification problem Lapin et al. [2015]. PeCC will need use the top k predictions to cache for long enough, but not more, to reduce CDE's resulting from prediction errors.

3 Prediction

Popularity of individual content on the Internet is well studied Szabo and Huberman [2010], Figueiredo et al. [2011], Szabo and Huberman [2010] and is known to be fairly predictable. In addition to per-customer level similarity, content exhibits similar behaviour across a traffic class Sundarrajan et al. [2017], which is used as a unit of load-balancing at Akamai. The ES tends have a dynamic mix of well studied traffic patterns like Zipf or lognormal. But a complex overlapping of these traffic patterns makes caching hard. However, well designed machine learning models can easily learn individual patterns in the mix, and make correct prediction of request sequence. The traffic behavior of an object, among other things, is dictated by its content class, request's position in the object's life-cycle and the traffic on the machine. The similarity in inter-arrival gap among objects can be due to a combination of different object features, which can be learned by using per-object features.

Dataset and Feature Details Each request for an object has a large set of features associated with it. Out of all such features, we use a few hundred of them and categorize them into 3 main types: 1. Static object features like size, content-type, etc 2. Dynamic Features that include Age, inter-arrival history, time of request, etc and 3. Traffic Features like traffic mix rates seen on machine and features related to end-users. These features can be continuous, ordinal, categorical, or time-series. We design a neural network which can learn end-to-end for such a variety of input types. The output of the neural network is a quantized next-arrival time (τ_q) of a request. This makes the learning task a ordinal-category classification problem.

Neural Network Design The neural network architecture has three major initial layer stacks to processes the continuous, categorical and time-series features available in our feature set.
Embedding Layer Stack: Embedding layer architecture can map elements from a set to a vectors, popularly used in Natural Language Processing task Gal and Ghahramani [2016]. We use this function approximation idea to map our categorical features, such as owner, content-type, time of day, etc into a Euclidean space of 100 dimensions. The learning of this mapping happens with the supervised learning of the rest of the network, unlike in the language modeling tasks. *Time-Series Layer Stack:* The Time-Series layer stack consist of multiple Long short-term memory (LSTM) layers Graves et al. [2013] for each time-series feature. The internal states of the LSTM nodes are fed to the fully connected layer of the network. *Continuous-Feature Layer Stack:* This module of the network takes all the continuous features of our system as input. It consist of few dense ReLU Nair and Hinton [2010] followed by Dropout layers Srivastava et al. [2014]. The output of all 3 layer stacks is fed to the Fully-connected part of the network, which consists of combination of dense PReLU He et al. [2015] and batch-normalization layers Ioffe and Szegedy [2015].
Loss function and Output The output for our task is a ordinal variable with many categories. Hence, normal cross-entropy loss does not yield good results because of its intrinsic inability to force ordering among output classes. To address this problem, we design network with loss functions to penalize every pair of true class and predicted class differently, hence enforcing an ordering among the classes. To obtain the final output we use an ensemble Rokach [2010] of all the trained models.

4 PeCC Design

As discussed in the previous section, we can achieve close to optimal performance by predicting future arrival time using a Deep learning system and evicting the objects furthest in the future (FiF). Instead of fully sequencing the next request for objects in cache, we relax our ML based prediction system to produce request's next arrival time as quantized bins (τ_q). This converts the prediction problem from a regression to a multi-class classification. The quantization bin size and the max quantization level are chosen based on the traffic profile. The size of individual bin, decreases as the the traffic rate increases and vice-versa. The task of prediction engine is to predict the τ_q for each object in the cache. However the predictions can be wrong in three way: **Prediction error 1**: $\tau_q^{pred} >> \tau_q^{true}$ **Prediction error 2**:$\tau_q^{pred} << \tau_q^{true}$ **Prediction error 3**: $|\tau_q^{pred} - \tau_q^{true}| < \delta$, for some small δ. Since errors in web traffic predictions are an inescapable, we try to reduces the impact of prediction errors on caching performance by reducing the likelihood of them becoming CDE's. We achieve this by designing a NN with a caching scheme that produces and uses multiple unreliable predictions with decreasing levels of confidence to cache and evict objects. The prediction system provides two estimate of the expected next interarrival time τ_q^{Pred1} and τ_q^{Pred2}. However, both the predictions can be wrong in one of three prediction errors types as discussed before. This results in multiple error combination cases for the pair of τ_q^{Pred1} and τ_q^{Pred2}. To recover from such error cases, we define and categorize errors by their impact on the caching performance and setup sub-caches to recover from these errors. Given the traffic and cache size, let T be the max interarrival observed for the objects getting hits in the cache. The objects with $\tau_q < T$ will have high chances of getting hit in the cache, while objects with $\tau_q > T$ has lower chances of getting hit. Thus T gives us an estimate of whether an object of some future inter-arrival time, can get-in and get hit in the cache. To make ideal caching decisions, using FiF to evict, we keep objects with $\tau_q^{Pred1} < \tau_q^{FIF}$ in the cache while keep out the objects with $\tau_q^{Pred1} > \tau_q^{FIF}$. But since the τ_q^{Pred1} predictions might have errors, we keep objects with predicted $\tau_q^{Pred1} > \tau_q^{FIF}$ and $\tau_q^{Pred2} < \tau_q^{FIF}$ in a separate limited cache segment, called Popular error cache (PEC).The objects with $\tau_q^{Pred1} < \tau_q^{FIF}$, which don't get a hit in the cache for τ_q^{Pred1} are marked as expired objects. Such expired objects with $\tau_q^{Pred2} < \tau_q^{FIF}$ goes from the main cache to expired cache (EC) segment. The share of each cache segment, out total cache space, is estimated by observing percentage of prediction error 1 and prediction error 2 in the traffic. Fig 3 describes the overall flow of objects within different cache segments.

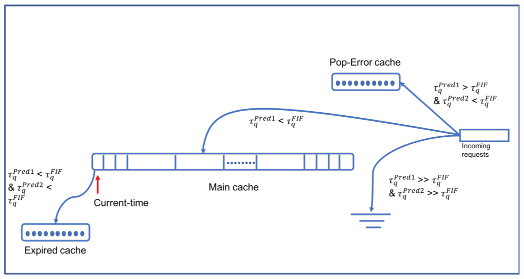
Pop-Error cache
$\tau_q^{Pred1} > \tau_q^{FIF}$ & $\tau_q^{Pred2} < \tau_q^{FIF}$
$\tau_q^{Pred1} < \tau_q^{FIF}$
Incoming requests
$\tau_q^{Pred1} < \tau_q^{FIF}$ & $\tau_q^{Pred2} < \tau_q^{FIF}$
Current-time
Main cache
$\tau_q^{Pred1} >> \tau_q^{FIF}$ & $\tau_q^{Pred2} >> \tau_q^{FIF}$
Expired cache

Figure 3: PeCC design

Trace	Gain % = $H_{pecc-lru}$/$H_{opt-lru}$
MC 1	48.26%
MC 2	52.62%

Table 1: Gain % in the hits by PeCC from theoretically max possible.

5 Results

In this section, we present the prediction result obtained from our proposed network architecture. We then discuss the improvements in caching performance observed through content aware predictive caching. For this study, we collect request logs from a set of randomly selected representative machines on the Akamai network for over a week. We train on 3-4 day's of data and test the prediction and the caching performance for the last few days.

The quantized next request inter-arrival time prediction task is a challenging ordinal multi-class classification problem because the output classes can be a few thousands in number. The prediction accuracy results from five ES traces are shown in Fig 2. The accuracy numbers are compared to a random, class-distribution aware, predictor. The results clearly corroborate that the traffic is not totally random and models can be trained to predict traffic request sequences. However, some traffic traces are more complex than other and hence the prediction accuracy that can be achieved varies from machine to machine. The final caching decision task still benefits from the next inter-arrival prediction.

In order to evaluate the improvements in cache performance we consider the cache hits obtained by LRU (current online caching systems), Belady's (offline optimal caching), and proposed PeCC. Let H_{lru}, H_{opt}, H_{pecc} be the hits in cache obtained by the LRU, Belady's and PeCC respectively. The theoretical possible gain in hits from LRU from OPT is

$$H_{opt-lru} = H_{opt} - H_{lru} \tag{1}$$

Similarly, the gain in hits from LRU to PeCC is

$$H_{pecc-lru} = H_{pecc} - H_{lru} \tag{2}$$

From our experiments, we observe a substantial increase in hits by PeCC from LRU. Table 1 show the percentage of hit recovered by PeCC as a percentage of theoretically possible recoverable hits by OPT from LRU. The results show large gains in the total hit by PeCC which LRU fails to acquire.

6 Conclusion

In this work, we successfully demonstrate that modern machine learning systems can be built for content aware caching which can significantly improve the caching performance obtained by classical online caching algorithms and other heuristic based variants.

In spite of the known shortcomings, classical caching algorithms and its variants are quite popular, because they have very low overhead and generally work well when the traffic satisfies the algorithm's underlying assumptions. However, with the advancements in practical ML systems, need for better cache-hit rates, and increasing long-tail content footprint, a novel approach of content aware caching using ML is ideally suited to improve current caching systems.

For the next step, we strive to improve the performance of PeCC by building a more complex model that accounts for complex cache hierarchies in the network and is robust over drastic traffic changes. At the same time, the learning cost involved in training models can be drastically reduced by exploiting the transfer learning opportunities we have observed across several machine in a region.

References

L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

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

M. Busari and C. Williamson. Prowgen: A synthetic workload generation tool for simulation evaluation of web proxy caches. *Comput. Netw.*, 38(6):779–794, Apr. 2002. ISSN 1389-1286. doi: 10.1016/S1389-1286(01)00285-7. URL `http://dx.doi.org/10.1016/S1389-1286(01)00285-7`.

F. Figueiredo, F. Benevenuto, and J. M. Almeida. The tube over time: Characterizing popularity growth of youtube videos. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 745–754, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0493-1. doi: 10.1145/1935826.1935925. URL `http://doi.acm.org/10.1145/1935826.1935925`.

Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1027–1035, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL `http://dl.acm.org/citation.cfm?id=3157096.3157211`.

A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013. URL `http://arxiv.org/abs/1303.5778`.

G. Hasslinger, J. Heikkinen, K. Ntougias, F. Hasslinger, and O. Hohlfeld. Optimum caching versus lru and lfu: Comparison and combined limited look-ahead strategies. In *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2018 16th International Symposium on*, pages 1–6. IEEE, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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*, SOSP '13, pages 167–181, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2388-8. doi: 10.1145/2517349.2522722. URL `http://doi.acm.org/10.1145/2517349.2522722`.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

A. Jain and C. Lin. Back to the future: Leveraging belady's algorithm for improved cache replacement. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 78–89, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-4673-8947-1. doi: 10.1109/ISCA.2016.17. URL `https://doi.org/10.1109/ISCA.2016.17`.

A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (rrip). *SIGARCH Comput. Archit. News*, 38(3):60–71, June 2010. ISSN 0163-5964. doi: 10.1145/1816038.1815971. URL `http://doi.acm.org/10.1145/1816038.1815971`.

M. Lapin, M. Hein, and B. Schiele. Top-k multiclass svm. In *Advances in Neural Information Processing Systems*, pages 325–333, 2015.

T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. *CoRR*, abs/1802.05399, 2018. URL `http://arxiv.org/abs/1802.05399`.

B. M. Maggs and R. K. Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

G. Neglia, D. Carra, M. Feng, V. Janardhan, P. Michiardi, and D. Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4):21, 2017.

L. Rokach. Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1-2):1–39, Feb. 2010. ISSN 0269-2821. doi: 10.1007/s10462-009-9124-7. URL `http://dx.doi.org/10.1007/s10462-009-9124-7`.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

A. Sundarrajan, M. Feng, M. Kasbekar, and R. K. Sitaraman. Footprint descriptors: Theory and practice of cache provisioning in a global cdn. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 55–67, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5422-6. doi: 10.1145/3143361.3143368. URL `http://doi.acm.org/10.1145/3143361.3143368`.

G. Szabo and B. A. Huberman. Predicting the popularity of online content. *Commun. ACM*, 53(8): 80–88, Aug. 2010. ISSN 0001-0782. doi: 10.1145/1787234.1787254. URL `http://doi.acm.org/10.1145/1787234.1787254`.

S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today's content caching: Why it matters and how to model it. *SIGCOMM Comput. Commun. Rev.*, 43(5):5–12, Nov. 2013a. ISSN 0146-4833. doi: 10.1145/2541468.2541470. URL `http://doi.acm.org/10.1145/2541468.2541470`.

S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, 2013b.

G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan. Driving cache replacement with ml-based lecar. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, Boston, MA, 2018. USENIX Association. URL `https://www.usenix.org/conference/hotstorage18/presentation/vietri`.