Let Once-Request Data Go: An Online Learning Approach for ICN Caching

Yating Yang
School of Computer Science of Technology
Beijing Institute of Technology
yangyating@bit.edu.cn

Tian Song*
School of Computer Science of Technology
Beijing Institute of Technology
songtian@bit.edu.cn

ABSTRACT

In-network caching significantly improves the efficiency of data transmission in ICN by replicating requested data for future re-access. In this work, we shift our focus on *once-request* data, which cannot be re-used and would lead to under-utilization of in-network caching. We present a name feature-based online learning approach to recognizing and filtering once-request data when making caching decision. It can dynamically update its parameters through online observation on previous recognition. Evaluation results show that our learning approach can recognize once-request data with more than 80% accuracy. By filtering those data, 76% cache replacement operations are saved and cache hit ratio is increased by 151%.

CCS CONCEPTS

• **Networks** → *In-network processing*; *Storage area networks.*

ACM Reference Format:
Yating Yang and Tian Song. 2019. Let Once-Request Data Go: An Online Learning Approach for ICN Caching. In *6th ACM Conference on Information-Centric Networking (ICN '19), September 24–26, 2019, Macao, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3357150.3357410

1 INTRODUCTION

As an inherent feature in ICN, in-network caching is of paramount importance to reduce network traffic and improve the quality of experience of users by temporarily storing contents for future request [2] [4]. To enhance the ability of caching, many research efforts have been made to find the optimal cache content placement policy with assuming that future content popularity is available [3]. Valuable studies have estimated popularity using sophisticated machine learning approaches [1], but those popularity-learning techniques would introduce extra complexity to in-network caching.

Instead of predicting accurate popularity distribution, in this poster, we explore another special type of data that are seldom discussed, i.e., *once-request* data. Once-request data are such data that will be accessed only once *within a short time*. They have little value of caching since it cannot be *re-used*, but could waste cache space and cause frequent replacement operations.

*Corresponding author: Tian Song, songtian@bit.edu.cn

ICN '19, September 24–26, 2019, Macao, China

ACM ISBN 978-1-4503-6970-1/19/09…$15.00
https://doi.org/10.1145/3357150.3357410

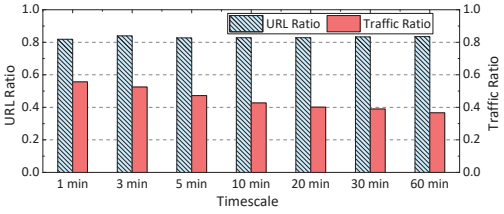


Figure 1: The ratio of once-request data

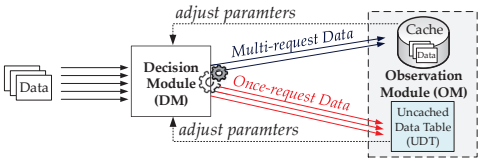


Figure 2: Online learning-based caching architecture

The behavior of those once-request data was observed from real trace with 1.06 million requests, which was collected from an egress router of a campus network over one hour. Each request contains a URL to indicate the requested data, and we transform each URL into a hierarchical name in the form of NDN naming. Fig. 1 depicts the average ratio of once-request names within a period from 1 min to 60 mins. It reveals that more than 83% names are requested only once. Those once-request data took up a large percentage of, more than 56%, one-minute traffic (i.e., the total request number).

To avoid holding such caching-valueless data, an intuitive method is to count accessing times of each content before caching them down, and only to cache data that has come before or by certain times. Particularly, Bloom Filter is employed in CDN system to fast count and check the previous access. However, this count-based caching approach makes caching decisions after counting previous accessing behavior, and cannot make timely caching decisions.

Different from previous work, we propose a method to make timely caching decisions without counting previous accessing times. We propose an online learning approach to recognizing those once-request data based on name features, and use a once-request-aware caching policy to filter those data and boost caching performance.

2 DESIGN

2.1 Architecture Overview

The architecture of proposed online learning-based caching is demonstrated in Fig. 2. Two modules are designed for online learning, Decision Model (DM) and Observation Module (OM).

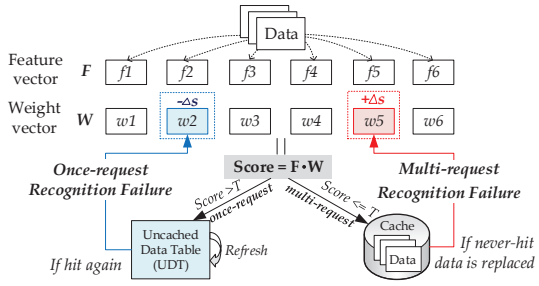
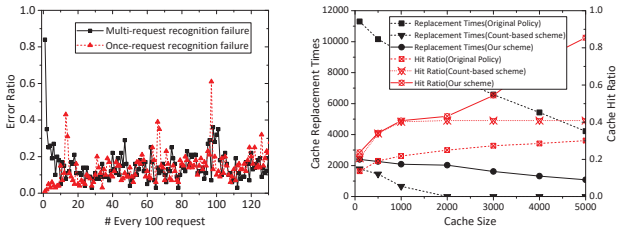


Figure 3: Online learning process



(a) Recognition error ratio (b) Cache Performance

Figure 4: Evaluation on online learning-based caching

DM learns the name features of once-request data, and decides whether to cache based on its learning parameters. It divides coming data into two categories: multi-request data and once-request data. The former will be cached, and latter will be recorded in Uncached Data Table (UDT) without caching. The OM observes whether the previous recognition is correct and is responsible for adjusting learning parameters if recognition error occurs.

2.2 Feature Selection

For each incoming data, a *feature vector* **F** will be extracted based on the content name and evaluated by DM to recognize once-request data and decide whether to cache or not.

A typical once-request data is requested with a name like "xx/xxx/xxx/201905/image/extensions/jpg/b75_5075186_64d75e_ef3b_0b8b61d_1", which is a relatively long name and has a long last component. In addition, some user-specific requests containing special symbols and words are also likely to be requested only once. After investigating names of once-request data, the following six features stand out and are selected to constitute a feature vector **F** .

- f_1 Length of request name
- f_2 Number of request name components
- f_3 Length of the last name component
- f_4 Length of the next to last name component
- f_5 Number of special symbols(such as "@, ?, #, ., &, %")
- f_6 Number of special words (such as "key, $client\_id$")

2.3 Online Learning Process

Our approach learns to recognize once-request data by adjusting the weight vector **W** (see Fig. 3). DM decides whether to cache according to the weighted score ($Score \leftarrow \mathbf{F} \cdot \mathbf{W}$). T is a threshold value for recognition. If $Score > T$, it will recognize as once-request data, and record it in UDT without caching. Otherwise, it would cache as multi-request data.

The weight vector **W** starts with **0** and is adjusted by OM, when it observes recognition failure. If it occurs, a random feature value f_{rand} will be selected to update with a step Δs. There are two cases that OM would adjust the weight vector: *a) Once-request Recognition Failure*: If a once-request data record in UDT is hit again, a random feature f_{rand} will be reduced by Δs. *b) Multi-request Recognition Failure*: If a cache replacement operation is issued to replace never-hit data, a random feature f_{rand} will be added with Δs.

UDT records previous uncached data within a period, which is a Bloom Filter structure. Specifically, refresh operation is periodically conducted to clear last-period locality.

3 PERFORMANCE EVALUATION

We conduct simulation with one-minute trace. Cache is defaulted with $size = 3000$. Learning-related parameters are set as $\Delta s = 0.01$ and $T = 10$. The trace contains 13191 requests in total and 7345 for once-request data. Our learning approach encounters 1322 multi-request recognition failures (false negatives) and 1268 once-request recognition failures (false positives). Furthermore, the recognition performance of our approach is calculated as $Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = 80.4\%$, $Precision = 82.7\%$ and $Recall = 82.1\%$. Then, the recognition performance during learning process is shown in Fig. 4(a). At the beginning, our approach has a relatively high ratio of multi-request recognition failure because it starts with caching most of the coming data. But with more data coming and being observed, it quickly learns an appropriate **W**.

Fig. 4(b) compares cache performance of our proposed caching approach under different cache sizes. By filtering those recognized once-request data, our approach can reduce 76.4% replacement operations and improve 151% cache hit ratio in average, compared to original NDN cache policy. In addition, count-based approach (with BloomFilter) is compared, which gets less replacement times since it never cache once-request data. But it has less cache hit ratio than our approach when larger cache size is allowed. Because count-based approach only cache data that arrive at their second time, leaving some second-request data in the cache but those data will not be accessed anymore.

4 FUTURE WORK

Our future work will explore and evaluate more features of once-request names from different types of real traces, and further investigate the benefits of filtering *N-request* data with the online learning-based caching approach.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China under grant No.61672101 and No.U1636119.

REFERENCES

[1] Z. Chang et al. 2018. Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era. *IEEE Wireless Communications* (2018).
[2] I. U. Din et al. 2018. Caching in Information-Centric Networking: Strategies, Challenges, and Future Research Directions. *IEEE Commun. Surv. & Tut.* (2018).
[3] S. Li et al. 2016. Popularity-driven content caching. In *INFOCOM*.
[4] G. Paschos et al. 2018. The role of caching in future communication systems and networks. *IEEE JSAC* (2018).