IEICE TRANS. COMMUN., VOL.E107–B, NO.2 FEBRUARY 2024

PAPER

A Recommendation-Based Auxiliary Caching for Mapping Record

Zhaolin MA[†*a)], **Jiali YOU**[†*b)], *and* **Haojiang DENG**[†*c)], *Nonmembers*

SUMMARY Due to the increase in the volume of data and intensified concurrent requests, distributed caching is commonly used to manage high-concurrency requests and alleviate pressure on databases. However, there is limited research on distributed record mapping caching, and traditional caching algorithms have suboptimal resolution performance for mapping records that typically follow a long-tail distribution. To address the aforementioned issue, in this paper, we propose a recommendation-based adaptive auxiliary caching method, AC-REC, which delivers the primary cache record along with a list of additional cache records. The method uses request correlations as a basis for recommendations, customizes the number of additional cache entries provided, and dynamically adjusts the time-to-live. We conducted evaluations to compare the performance of our method against various benchmark strategies. The results show that our proposed method, as compared to the conventional LCE method, increased the cache hit ratio by an average of 20%, Moreover, this improvement is achieved while effectively utilizing the cache space. We believe that our strategy will contribute an effective solution to the related studies in both traditional network architecture and caching in paradigms like ICN.

key words: *auxiliary caching, recommendation, ttl caching, distributed system*

1. Introduction

In the current era of big data, storage and computation systems are shifting from centralized to distributed. To enable rapid data distribution, streamline business processing, and enhance overall user experience, an increasing number of compute-intensive businesses and services are now relocating from core clouds to edge clouds. Due to the increase in data volume and concurrent requests, distributed caching is often introduced to handle high-concurrency requests and alleviate database pressure. A suitable caching system should possess higher cache efficiency, store more useful data, and filter out a greater portion of the back-end storage system's request load.

Currently, distributed caching can be mainly divided into two types: the first type is content caching, which is usually used in CDNs to alleviate the request pressure on the origin server. When data is hit on the CDN cache, it does not need to be retrieved from the origin server but is directly distributed to users by the CDN server closest to them. This type of cached data occupies a significant amount of storage space. Although unstructured, the cached data remains stable and does not have an expiration date. On the other hand, there is another type of caching for mapping records. This type of caching involves the storage of mapping records or metadata, which is commonly implemented in Domain Name System [1] and Information-centric network that rely on name resolution systems (NRS) [2]–[5]. The primary distinction between record cache and content cache lies in their object type. Record caching is structured, occupies smaller space but is often accompanied by a time-to-live (TTL) which entails cached records to expire after a certain period. The reason for this characteristic is that the existence and location of content usually change, so records often have a life cycle to ensure the freshness and effectiveness.

To our knowledge, most existing research studies focus on content caching, and little research has been conducted on mapping records caching. In addition, the primary objective is usually to design suitable cache replacement strategies to improve cache hit rates under limited cache space [6], [7]. In previous studies, it was usually assumed that user requests on the storage system were independent events. From a worldwide point of view, this presumption is reasonable since there usually exists a substantial number of users that access the system simultaneously. Moreover, it has been proven to be accurate when the cache capacity tends to infinity [8]. However, for small-scale user clusters (such as edge clouds or edge data centers), the *neighbor effect* [2] cannot be ignored. Typically, individual user requests have strong context dependencies because content requirements often appears in groups (such as requests for resources on web pages or requests for subsequent content blocks in videos). This phenomenon causes requests that occur in close temporal proximity to have a certain correlation, that is, temporal correlation of requests.

In distributed edge clouds, cache nodes are usually distributed in various locations and face requests that are often sudden in time and uneven in space. In order to improve the cache efficiency of the system and reduce concurrent traffic peaks for origin server requests, this paper mainly focuses on the aforementioned mapping record caching and proposes a recommendation-based adaptive auxiliary caching method (AC-REC). In this method, auxiliary caching is defined as delivering the primary cache record along with an incremental list of auxiliary records. This method uses the correlation between requests as the recommendation basis for auxiliary

Manuscript received July 7, 2023.
Manuscript revised August 31, 2023.
Manuscript publicized January 15, 2024.

[†]The authors are with National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing, 100190 China.
[*]Presently, with University of Chinese Academy of Sciences.
a) E-mail: mazl@dsp.ac.cn
b) E-mail: youjl@dsp.ac.cn
c) E-mail: denghj@dsp.ac.cn
DOI: 10.23919/transcom.2023EBP3117

caching, customizes the number of incremental cache entries and dynamically adjusts the TTL based on the cache space of each nodes and the historical popularity of cache records. This method is a data-driven caching method that is suitable for various application scenarios, such as caching of mapping records in information or content central network architectures, edge cloud caching, DNS record caching, and caching of flow table entries in SDN switches [9], [10]. The major contributions of this paper are as follows:

- We first design AC-REC, which utilizes a plug-and-play recommendation model to determine the candidate set of mapping records for auxiliary caching. It fully considers the occupancy of the cache space in the TTL-based cache system and exhibits greater robustness across different workloads.
- Trace-driven evaluations are conducted to measure the performance of our proposals. The results indicate that the AC-REC algorithm achieves a higher average cache hit ratio and a higher cache space utilization rate compared to traditional caching strategies, such as Leave-Copy-Everywhere (LCE).

The article is divided into five main sections. In the following Sect. 2, we introduce the research background, related work, and the motivation for choosing the auxiliary caching approach. In Sect. 3, we model the problem and provide theoretical explanations for the recommendation-based auxiliary caching algorithm. In Sect. 4, we design simulation experiments to compare the performance of the proposed algorithm with other algorithms and analyze the results. Finally, in Sect. 5, we summarize the article and provide an outlook on future research directions.

2. Related Works and Motivation

Our design approach is inspired by the concept of soft cache hits proposed in [11], which defines soft-cache-hit as the requested content is not in the cache but is present in the recommended list. Motivated by this thought, we were intrigued whether integrating recommendation system functionalities to a distributed caching system could also enhance the cache hit ratio and diminish the number of concurrent requests sent to the origin server. Our research, as well as the arguments presented in [12], [13], have shown that data requests exhibit clear spatio-temporal characteristics, i.e., neighborhood effects. Moreover, objects closer to the network edge cache usually have higher access frequencies than those located upstream [14]. For example, there is similarity in the content acquirement by students in the same university in the same time period, and there is also similarity in the hot news topics followed by people in the same region. [15] proposes a region-based pre-caching strategy for vehicular edge networks, which includes algorithms for selecting pre-caching regions and nodes, which effectively improves the cache hit rate. Based on these arguments, we hypothesize that for the user group connected to the switch under the edge cloud, considering auxiliary caching for content related to that group's requests, rather than just caching content with higher global popularity, can improve the system's cache hit ratio.

A similar concept was previously introduced in [16] as incremental caching at the file level. This method calculates the access frequency of a file and caches the first block when the file is first requested. For subsequent requests, $2^n - 1$ blocks are pre-cached, where n is the number of times the file has been requested. However, applying this approach to finer-grained ICN or CCN-based block storage is not feasible. Additionally, this method only accelerates continuous requests for a file without considering related content from a modeling perspective.

Cache replacement algorithms have always been considered an important factor affecting cache hit ratio. Classic cache replacement methods for mapping records [7], such as LRU, LFU, and FIFO, are all designed to handle a single cache record. Yang et al. [6] propose a caching strategy that groups records and uses this grouping for replacement decisions that result in an improved cache hit ratio. This strategy achieves both high efficiency and high throughput. In addition, in mapping record caches, the TLRU method [17] adds expiration time to the cache to ensure the freshness of records, but this method does not fully utilize the cache space released by timeouts. If we can obtain the remaining cache space in advance and incrementally cache additional valid content besides the original records, subsequent cache requests can hit, improving the cache hit rate per unit time and reducing concurrent traffic to origin server. [18] uses the cache space parameter to predict the expected hit density of each object (hit density per space consumption), and filters out objects that contribute little to the cache hit rate during cache replacement. However, this method is mainly for content caching and is not suitable for mapping record caching (mapping records usually have the same space occupation). To make good use of the remaining cache space, it is necessary to obtain the operational status of the distributed cache. Fortunately, the SDN controller [9], [19] has laid the theoretical and practical foundation for this idea by providing awareness of the underlying network devices.

In addition, due to the significant long-tail effect of user requests [20], that is, a small number of content accesses account for a large amount of request traffic, records at the tail are usually cleared due to timeouts or LRU replacement strategies before triggering the next request hit. Traditional caching methods usually use fixed TTL, and the expiration of hot data at the same time can cause cache avalanche phenomenon, resulting in increased concurrent load on the backend storage system. The article [21] points out that using longer TTL in DNS systems can greatly reduce network transmission latency, because the longer expiration time of hot content filters out more traffic. However, even in the case of limited storage space, caching the long-tail content for a long time does not significantly improve the cache hit rate. Alici et al. [22] propose an adaptive TTL strategy for caching query results that has been shown to reduce the fraction of stale results served from the cache as well as the

fraction of redundant query evaluations on the search engine backend. Additionally, the proposed method outperforms a caching strategy that uses a fixed TTL value for all queries. Therefore, we believe that more aggressive strategies should be adopted for this part of the records, setting shorter TTL times to quickly recover cache space resources and to guarantee the record freshness.

3. Analytical Model and Assumptions

In this section, for the sake of illustration and analysis, we have made some assumptions and mathematical models for the research problem. In addition to considering the global characteristics of requests, such as popularity, this study also focuses on their spatiotempora characteristics. The idea of auxiliary caching is used to promote cache hits ratio in the next time period by taking advantage of cache loss in the current period. In summary, we describe the problem solved in this study as follows: *selecting a subset of records from set S for incremental caching to maximize the average cache hit rate*. This problem is an NP-complete "knapsack problem" that we divide into two sub-problems and propose a heuristic solution approach.

3.1 Assumption

Our approach abstracts the caching system into three network entities: the client who initiates the request, the cache nodes, which are typically ICN cache units or CDN service nodes, and the data source responsible for storing resources. The client is analogous to the user in a recommendation system, while the data source or origin server is analogous to the item repository in a recommendation system. Similarly, the data source suggests potential requests to the client, analogous to how a recommendation system suggests items of interest to users. Given the vast number of users in a network environment, we do not perform fine-grained recommendations for each user, but instead group units with similar features to increase the accuracy of recommendations and reduce system load. We use the user access node as the recommended cache unit, aggregate the temporal and spatial characteristics of requests, reduce the distribution pressure on the database or origin server, and improve caching speed.

3.2 Auxiliary Caching Algorithm

The algorithm consists of three steps, which is presented in Fig. 1 in the form of a flowchart to aid comprehension. The central part of the flowchart represents the main steps of the algorithm, while the left and right sides depict the input models, parameters, and hyperparameters required for each step. The specific operations for each step will be elaborated in separate subsections.

3.2.1 Determine Candidate List

The first step involves selecting a fixed candidate list size

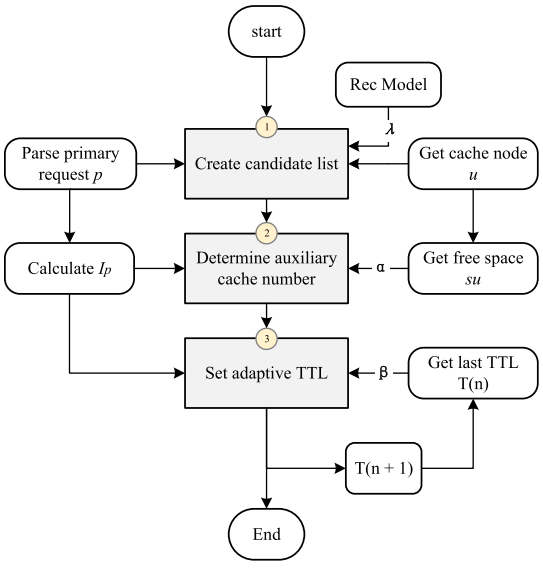


Fig. 1 Auxiliary caching algorithm flow chart.

and determining the additional content for caching, which is similar to the recall step in a recommendation system. Since the keys of cached content are usually unstructured IDs or hash values of metadata, we recommend using collaborative filtering-based recommendation algorithms such as itemCF, userCF, or other advanced model-based recommendation algorithms [23]–[25]. In our simulation, we utilize the CDAE algorithm (Collaborative Denoising Autoencoder) as the recommendation model [23]. This algorithm helps to effectively capture latent user feature representations, enhancing the accuracy of personalized recommendations.

In our algorithm, the acquisition, registration, and deregistration of the records on the cache nodes are analogous to the interaction behavior between "users" and "items". Here, we use item-based collaborative filtering as an example. As shown in the Fig. 2, Node1-Node4 are four caching nodes, and the Storage Controller is equivalent to the data source repository. We can use the nodes' historical interaction information with records to infer the similarity between records. For example, in the figure, Node2-Node4 have similar historical interaction information with records A and B, and we can infer that the similarity between rec-A and rec-B is high. When Node1 initiates a new request to the storage system, in addition to caching the primary requested record A, we incrementally cache its similar record B and respond in the form of (key, value), where value represents the recommendation index, i.e., the degree of similarity with the primary record.

The AC-REC strategy leverages the similarity between records and the ID of the requesting switch to recall similar records, which need not be confined to the recall approach of a particular recommendation system. Section 4 presents a comparison between the effectiveness of various recommendation algorithms and the methods of optimal and random recommendations, which serve as a reference for implementing and deploying this algorithm. The following section will

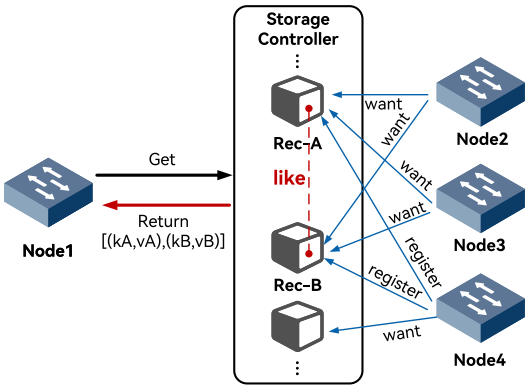


Fig. 2 Schematic diagram of recommendation-based auxiliary caching.

Table 1 Symbols in AC-REC algorithm.

Symbols	Definition
p	The current primary request key
u	The node that handling the request
K	The set of cached keys
L	The recommended candidate list
λ	The recall number ratio
k_i	The i-th request key
v_i	The recommendation index of k_i
n	The request number
I	The importance of a certain cache
A	The auxiliary cache entry size
T	The time-to-live value
s_u	The free space for node u

provide a detailed explanation of the structure and algorithm of the recommended candidate list, including the parameters and physical quantities involved, as shown in the Table 1.

Unless otherwise specified, we will use the example of caching in the form of (key, value) pairs, and the set of cached keys will be denoted as K. We define the recommended candidate list as L, the amount of data stored in the data source as m, and the recall ratio as λ. The size of the recommended candidate list is $len(L) = \lambda m$. We define the current request p as the main request initiated by node u, and the key of the current request as k_p. The key of the i-th cached content in the storage system is k_i, and its recommendation index is v_i. The recommendation index is a physical quantity that represents the importance of the cached record. The higher the recommendation index, the more likely the cache is to be incrementally cached. The recommended cache candidate list is predicted through the recommendation system's algorithm and can be represented as:

$$L = f(K, p, u, \lambda) \tag{1}$$

Here, f is the function that uses the recommendation system model for prediction. It recommends based on the input node u, the main request p, and the candidate list size λm, and returns a list containing the recommended key and its recommendation index. During predicting, model caching can be used to expedite the inference process of the model for requested recommendation results in practical production

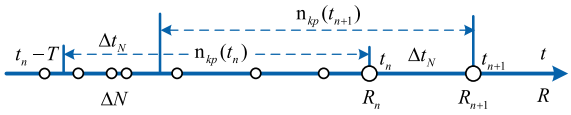


Fig. 3 The sliding window mechanism.

environments. The indices of the stored entries are denoted as $o_1, o_2, \ldots, o_i$, and they take the following form:

$$L : \{(k_{o1}, v_{o1}), (k_{o2}, v_{o2}), \ldots, (k_{o\lambda m}, v_{o\lambda m})\} \tag{2}$$

Although the training process of the recommendation model is beyond the scope of this paper, we describe a feasible training method in the simulation section and provide satisfactory results based on this approach.

3.2.2 Auxiliary Cache Number

The second step is to determine the number of auxiliary cache records and the TTL value of the them after the candidate list is determined. In our approach, we determine the number of entries for incremental caching based on two factors: the available space u_a on the current cache node and the importance of the primary cache, denoted I_p. We calculate I_p as the proportion of access record k_p's number of visits n_{k_p} during a unit time to the total number of visits n_k during that same period. To reflect the dynamic changes in the distribution of content requests, a time sliding window mechanism is used to dynamically calculate and update n_{k_p}. To simplify the computations, the specific value is converted as a probability distribution ranged in (0, 1), employing the sigmoid function in normalization in Eq. (5). The sliding window mechanism is shown in Fig. 3, Where R_n and R_{n+1} denote the nth and $(n+1)$th requests for caching k_p, respectively. The temporal interval between them is denoted as Δt_N and the requested counting interval is marked as T. ΔN represents the number of requests during window sliding Δt_N time.

$$n_{k_p}(t_{n+1}) = n_{k_p}(t_n) + 1 - \Delta N \tag{3}$$

$$n_k = \sum_i n_{k_i} \tag{4}$$

$$I_p = sigmoid(n_{k_p}/n_k) \tag{5}$$

The inference can be drawn that the primary cache's significance relates to the frequency of requests over a particular time period, also known as content popularity in some literature [26]. The remaining space of a cache node, denoted as s_u, is measured by mapped record entries, including reclaimed space due to timeouts, and space occupied by unreclaimed timed-out records. The data can be sent to the storage controller through the request message (CDN architecture), or via the packet-in message (SDN architecture). The allocation of the auxiliary cache space for this request is determined by the importance of the primary cache record and the current remaining space of the cache node. To restrict the proportion of cache space that can be used for

auxiliary caching, we set a constant $\alpha \in [0,1]$ to represent the space occupancy rate. In the extreme case where $\alpha = 0$, it indicates that the remaining space of cache node u cannot be used for caching. The formula for calculating the number of incremental cache entries A_{pu} is:

$$A_{pu} = \alpha I_p s_u \tag{6}$$

3.2.3 Adaptive TTL

Once the number of incremental cache entries is determined, we select the top A_{pu} recommended cache entries from the candidate list and denote them as set L_A. We then allocate TTL to each cache in L_A based on the TTL of the primary cache and the recommendation position of the cache entry. The higher the recommendation position of the mapping record, the longer the TTL allocation time. The design is based on the assumption that if the primary cache contains a popular record, then its related records or subsequent records are likely to be accessed soon after. The benefit gained from incrementally saving these records is thus higher. We denote the TTL of auxiliary cache k_i as T_i and update the TTL of cache i using a time-based sliding average value, with the calculation formula as follows:

$$\begin{aligned} &T_i(n+1) = \beta T_i(n) + (1-\beta) v_i I_p t_0 \\ &s.t. \quad k_i \in L_A, 0 \le \beta < 1 \\ &\qquad T_i(0) = t_0 \end{aligned} \tag{7}$$

Here, t_0 represents the standard TTL value, which is a constant and its value can be referred to in the literature [27]. β is the weight of the sliding average and its value ranges from $[0,1)$. As can be seen from the above formula, the value of T_i is generally less than t_0, which means that the TTL of the content in the auxiliary cache will not exceed the standard cache TTL. Using the above formula (7), we calculate the TTL for each record in L_A to obtain the final cache distribution list L_{out}:

$$L_{out} = \{(k_{o1}, t_{o1}), \ldots, (k_{oA}, t_{oA}), (k_p, t_0)\} \tag{8}$$

To facilitate comprehension, the pseudo code for the above algorithm is shown in algorithm 1.

Algorithm 1 AC-REC algorithm

```
Require: p, λ, u, α, β, t0
Ensure: Lout
Initialization;
model ← load pre-trained RECmodel
function getRelatedContent(u, p, k, tp)
    initialize resTList, resPList to empty;
    tl ← getTimeStampBefore(tp);
    for t in reversed(tl) do
        if t + 10 < tp then
            guriList ← getGroupURI(t)
            for uri in guriList do
                resTList.append(uri)
            end for
        end if
    end for
    resPList ← model(u, p)
    candList ← resTList[: k/2] + resPList[: k/2]
    return candList
end function
function getCandidateSize(α, u, p)
    su ← getAvailableSize(u)
    Ip ← getUriPop(p)
    Apu ← int(α * su * Ip)
    return Apu
end function
function assignTTL(β, t0, u, Ip, candList, Apu)
    sort(candList)
    candList ← get first Apu item in candList
    for c, v in candList do
        t ← β * getTTL(u, c) + (1 - β) * v * Ip * t0
        Lout.append((c, t))
    end for
    return Lout
end function
function Main(p, λ, u, α, β, t0)
    k ← λm, α ← 0.5, t0 ← 600
    tp ← getTime(p)
    cl ← getRelatedContent(u, p, k, tp)
    A ← getCandidateSize(α, u, p)
    Lout ← assignTTL(β, t0, u, Ip, cl, A)
    return Lout
end function
main(p, λ, u, α, β, t0)
```

4. Simulation and Analysis

This section will conduct simulation experiments on the method proposed in Sect. 3 and analyze the results. An auxiliary caching algorithm was implemented based on the Icarus [28] simulation platform.

4.1 Environment and Data Preprocessing

4.1.1 Dataset Collection

In order to assess the efficacy of the auxiliary caching algorithm on a practical network traffic model, a week-long data collection process was conducted. Request data from CDN servers located underneath a cluster node owned by a cloud service provider was recorded from nginx logs. Next, we utilized Logstash to format the nginx log files and subsequently preserved them in Elasticsearch. The structured information was extracted and preserved as a CSV file for training the recall model.

As the daily request volume was large (about 6 million requests), we only took the first one million data and filtered out the top 50 cities with popular requests as our initial dataset. The arrival of network requests usually follows a long-tailed or heavy tailed distribution [20], which was also verified in our dataset. We plotted the distribution of the dataset as shown in the Fig. 4, which displays the request model for one of the seven days. Even though we only counted and plotted the top 100 URIs, the long-tail effect of the request distribution is still evident.

Next, we preprocess the initial dataset and construct a

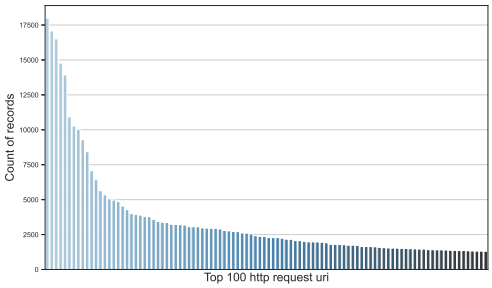


Fig. 4 Request distribution.

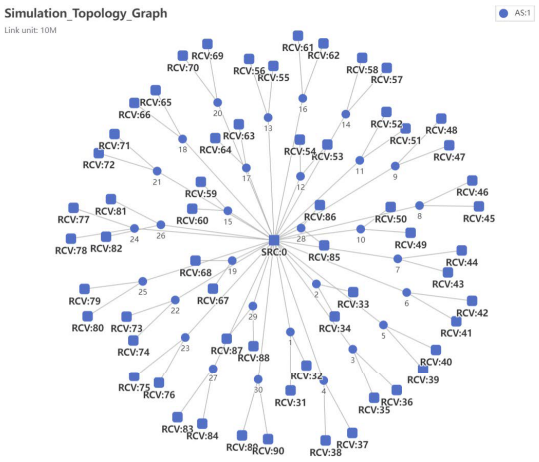


Fig. 5 Simulation topology.

simulated request workload. We divide the original dataset into 10 parts, with each part consisting of the first 50,000 requests as a group for experimental workloads. The URI and client IP are mapped to the content ID and client ID of the requesting node for the simulation experiment, respectively. The city of the requester is mapped to the access switch of the simulation node. The timestamp indicates the relative time represented by the 00:00 on the day of the experiment in the capture log. This allows our simulation traffic model to preserve the time series and geographic characteristics of real traffic.

4.1.2 Topology

Our simulation employs a star topology, where the data source is at the center, connected directly to 30 cache nodes. The bandwidth of each link is set to a fixed 10 Mb per second. We assume that the data source possesses an infinite storage capacity to accommodate all requested resources, and hence a cache miss in the source is unlikely to occur. The reason for using a star network topology in the simulation is that our application scenario is focused on SDN, which involves centralized devices (SDN controller) that determine caching strategies and serve as the data source. We are more concerned with the effects of caching different records via auxiliary caching rather than the placement location of cache entries. Thus, we streamlined the tangible connection between nodes and data sources within the topology. Instead, we employed a logical link to denote the connection between cache nodes and data sources.

In order to streamline the time taken to construct the shortest routing path during simulation, we have attached two client nodes to each cache node. Data requests are sent by randomly selecting a pair of client nodes that are connected to each cache node. It is worth noting that the number of clients will not affect the results of the experiment, as our algorithm recommends based on the granularity of edge caching nodes rather than client. The topology is illustrated in the Fig. 5.

4.2 Algorithm Parameters Setting

This section mainly describes the main parameter settings and reasons for the three steps of the auxiliary caching algorithm.

The initial step involves obtaining a recommended list of candidates for the current request. This is done by taking into account both the ID of the sending switch and the specifics of the current request. In the AC-REC method, the selection of the candidate list takes into account both temporal and geographical relevance. For the temporal-based recommendation part, this simulation uses a historical request group to recommend the URI group requested within 10 seconds after the first 3 requests that resolved the primary URI. During the system's cold start phase, a cache node lacks historical temporal information. For this reason, geographical coordination becomes the primary factor for obtaining a recommended list of candidates. To simplify the recall process, we set a fixed recall candidate list size as a hyper parameter k. We will analyze the impact of k on the simulation experiment results in later sections, and in other experiments, we set the k value to the default value of 1000.

For the recommendation based on geographical relevance, we use the CDAE algorithm for recall. When training the recall model, the long-tail distribution characteristics of user request traffic are evident, and the large amount of data in the long-tail section can lead to a large and sparse similarity matrix. To speed up the training process, we filtered out switch nodes with less than 20 requested URIs and URIs that were requested less than 10 times. To prevent overfitting, we trained a total of 1000 epochs and set an early stopping mechanism to end training when the NDCG@10 metric did not increase for 100 consecutive values. The parameter indicators used for training are shown in the Table 2.

In step 2, we set the switch space occupancy limit α to 0.3. In addition, we designed a set of experiments to compare the impact of α on cache hit rate and average remaining space on switches. In the simulation experiment, the total cache space on all switches was initialized to 20% of the total number of requests, and the cache space on each switch was evenly distributed. Before each time we calculate the remaining available cache space on the switch, we perform a dump operation to ensure that we obtain the correct available space.

In step 3, we use the moving average to update the TTL

Table 2 Parameters setting for training.

Parameter	Description	Value
minItemPerUser	Filter using the minimum number of items each user has interacted with	20
minUserPerItem	Filter using the minimum number of user each item has been interacted with	10
holdoutUsers	Use holdout cross-validation method	1000
testRatio	ratio of the sample for test	0.2
epochs	Max number of epochs	1000
batchSize	The number of training examples utilized in one iteration of gradient descent.	256
earlyStop	Early stop when NDCG@10 indicator does not increase for 100 times	100

of the mapping records in the incremental cache, and we set β to 0.5 to balance historical and real-time information. As a hyper parameter, we will discuss the impact of t_0 on simulation results in the following Sect. 4.4. In general, we set t_0 to 600 seconds, which is the default expiration time of DNS records.

4.3 Performance in Different Algorithms

In order to ensure the reliability of the results, we calculated the average cache hit rate of 10 workloads in each group of experiments for each strategy, and compared the strategies based on data from 7 consecutive days. We used the LCE [29] method as the baseline, which does not perform additional content caching and only caches the mapping record of the current request. We compared several different auxiliary caching methods, including AC-POP, AC-OPT, AC-RAND, and AC-REC. The AC methods above are all belong to the same category. These methods all fall under the auxiliary caching scope, which represents a form of proactive caching initiated by the data source. We only differentiate them based on the form of content retrieval in proactive caching. AC-POP utilizes usage records' popularity as an indicator for auxiliary caching, which is commonly considered in previous studies [30], [31]. AC-RAND is a naive random strategy, whereas AC-OPT is the optimal strategy for incremental caching based on the request traffic model. These methods use the same parameters in the second and third steps of the auxiliary caching algorithm, and the only difference is the method used to obtain the candidate incremental caching list. For the value of hyper parameters, we choose reasonable parameter values discussed in the next Sect. 4.4.

In the methods we focus on, AC-REC is the prominent method (proposed in Sect. 3) which uses recommendation to obtain the recommendation list. AC-RAND uses random sampling to obtain the recommendation list. AC-POP is a classic method that recommends the mapping record with the highest global content popularity. AC-OPT is the theoretically optimal method for incremental caching, which directly uses the subsequent requests of the current primary request as the candidate recommendation list. The present method is not feasible for real-world systems as it is not possible to predict future query requests with complete accuracy.

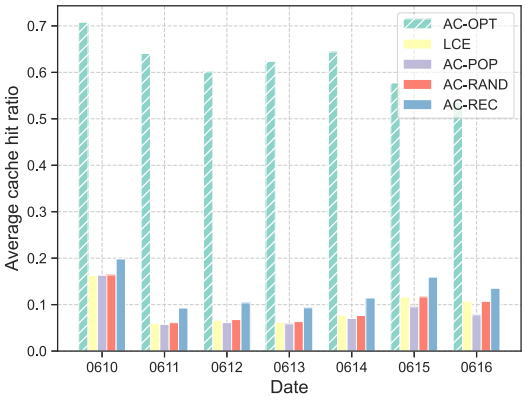
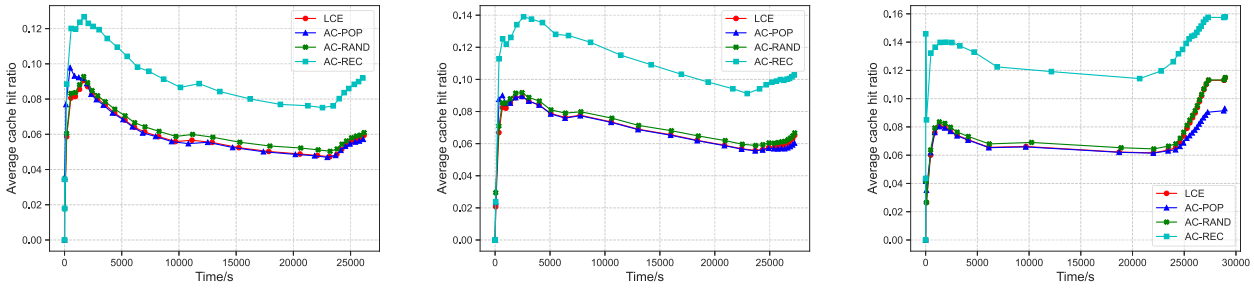


Fig. 6 Average cache hit ratio for different algorithm.

We only use it as a theoretical upper bound for comparison in simulation results. The average cache hit ratio of these methods is shown in Fig. 6.

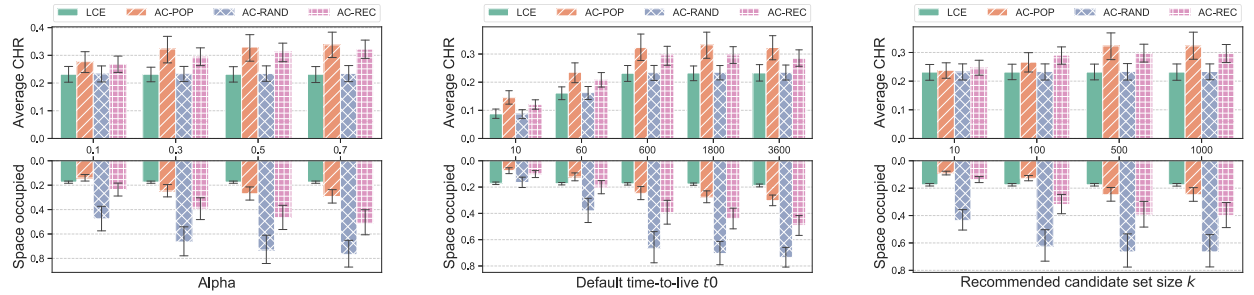
First, it is worth mentioning that the cache hit ratio of the AC-OPT algorithm outperforms other caching methods, suggesting the high effectiveness of our auxiliary caching approach. If the recommendation list is sufficiently consistent with the current traffic distribution model, the cache hit ratio can be improved by 5–6 times compared to the traditional LCE method. Second, we compared the AC-POP, AC-RAND, and AC-REC algorithms and found that the AC-REC method, which obtains its list of candidates by a recommendation algorithm, has a higher average cache-hit ratio over a seven-day period. For example, on June 10th, the cache hit rate of AC-REC reached 19.38%, which is a 19.08% increase compared to the 16.28% of LCE, and there was also about a 20% increase on other days. The AC-RAND algorithm exhibits only a marginal improvement over LCE, with minimal effect. This could be because the hit rate of subsequent requests after the primary request is not improved by the randomly constructed candidate set. The AC-POP method has a lower cache hit rate than LCE, indicating that global hot resources have a time limit. Pushing the same hot records repeatedly to each distributed edge cloud not only fails to increase the cache hit ratio but also has the opposite effect. This is because auxiliary caching, due to caching additional records, will to some extent exacerbate the replacement process in the cache. The long-tail nature of requests implies that recommending only popular records leads to a loss of diversity in the long-tail section of records, resulting in suboptimal performance for real traffic models.

To observe the trend of cache hit ratio over time in a single simulation experiment, we selected request data spanning three days, tracked them, and calculated the cache hit ratio with simulation time. The results are shown in the following Fig. 7. At the initialization of the simulation experiment, there were no cached records on each cache node, and the cache hit rate was 0. As the experiment progressed, the average cache hit rate increased rapidly, with the cache hit rate of the auxiliary caching method increasing slightly faster than that of LCE, and the cache hit rate of the AC-REC algorithm



(a) (b) (c)

Fig. 7 Cache hit ratio with time.



(a) The impact of space utilization quota α (b) The impact of standard time-to-live t_0 (c) The impact of candidate set size k

Fig. 8 The impact of different hyper parameters.

always higher than that of other methods. In the middle of the simulation experiment, the long-tail distribution effect of requests began to show, and the increasing number of missed requests led to a decrease in the overall cache hit rate, which stabilized after a period of time.

4.4 Discussion

In addition, we conducted grouped experiments on the three main hyperparameters related to auxiliary caching to analyze the impact of each parameter on the average cache hit rate and the average cache space utilization. We divided one day's request workload into ten groups based on time averaging, which were used for 10 independent repeated experiments. For each group of experiments, we compared representative LCE, AC-RAND, AC-POP and AC-REC algorithm. The results are shown in Fig. 8, which show the bar charts of cache hit rate and cache space utilization rate under different α, k, and t_0, respectively. The length of the error bars in each bar chart was determined by calculating the 90% confidence interval of the data.

Figure 8(a) illustrates that the AC-POP and AC-REC method outperforms the LCE and AC-RAND methods in terms of cache hit rate, with a greater difference as α increases. This is because the space utilization quota α determines the size of the space in which incremental caching can play a role. When α is small, the number of entries in incremental caching is limited, making it difficult to achieve cache hit for subsequent requests. When α is large, the number of entries in incremental caching increases, making it easier to achieve cache hit, and the corresponding cache space utilization of the switch also increases. However, even when α is 0.7, thanks to the adaptive dynamic TTL of this method, the average cache space utilization of the AC-REC method does not exceed 50%. This level of utilization represents a favorable and tolerable load state for the device. In contrast, the cache space utilization of the AC-RAND method has exceeded 70% when α is 0.7, but it has not achieved a good cache hit rate improvement. The cache hit rate of the LCE method is not high, and the cache space utilization rate is also low under the influence of the TTL timeout mechanism. In addition, we found that the cache hit rate of AC-POP is slightly higher than AC-REC and occupies less space. This proves that using global content popularity as an auxiliary cache can effectively improve cache hit rate when the total number of records is small (only 1/10 of the requests were used in each experiment). However, the effectiveness of this method is constrained by the accuracy of prior knowledge and it yields poor performance when dealing with a large number of records which is shown in Fig. 6. Furthermore, implementing it in practical production environments is challenging.

Another important parameter that affects the results is the initial lifetime of the cached record, t_0. Since we use the sliding average to update the TTL of the cached record, the initial lifetime will greatly affect the TTL value of the primary cache and the auxiliray cache records. From Fig. 8(b), we can see that the average cache hit rate increases with the increase of t_0 when t_0 is small, and it stops increasing after reaching 600s. After that, a too high t_0 will cause the average

cache hit rate to decrease, which is reflected in all methods. A larger t_0 will cause the mapping record not to be cleared even if it times out, thus consuming a large amount of cache resources. Many requests are subject to the long-tail effect, meaning that they are not frequently requested in a short time frame. Consequently, a significant amount of cache is rendered invalid. The LRU cache update strategy removes caches that are truly effective due to a lack of space, resulting in a reduced cache hit rate. That why it is essential to define a justifiable value for t_0 when deploying the cache system in real network architecture.

Finally, we analyzed the impact of the candidate list size k on the cache hit rate and cache space utilization rate. From Fig. 8(c), we can see that increasing the value of k has little effect on improving the cache hit rate, and when k reaches a certain value, increasing k will not bring any gain in cache hit rate or increase the cache space utilization rate of the switch. This is due to the fact that the actual number of entries generated for auxiliary caching is influenced by parameters such as the cache space size and popularity, resulting in a typically small number of entries. Therefore, as long as the size of the recommendation list is not too small, it can affect the recommendation. It is worth noting that AC-REC exhibits a higher cache hit rate than AC-POP when the number of recommended candidate sets k is small, while AC-POP performs slightly better when k is large. This indicates that the recommendation strategy of AC-REC, which is based on temporal and content relevance, is superior to the strategy of AC-POP, which uses global popularity, in terms of recommendation accuracy.

5. Conclusion

This paper proposes an approach to auxiliary caching, based on proactive recommendation, for edge distributed caching systems. Specifically, we introduce a recommendation list acquisition method based on joint temporal and geographical characteristics. This method aims to expand the coverage of the candidate set and accommodate the cold start state of the system. Then, We use the remaining available space of the cache node and the importance of the current request to the primary cache to determine the number of auxiliary cache entries. In addition, we dynamically calculate the TTL of the cache entries using a sliding average algorithm. Finally, we use a real dataset to experimentally test the algorithm on the icarus simulation platform. The experiments demonstrate that the proposed algorithm exhibits excellent performance in terms of reliability and dynamic adaptability. Moreover, it outperforms other algorithms regarding the utilization of cache space and cache hit ratio. We firmly believe that the caching strategy proposed in this article can bring about a significant positive impact on distributed edge caching and mapping record caching in ICN.

Our future improvements will focus on the following aspects. Firstly, we plan to explore more suitable recommendation algorithms to model the request pattern. Moreover, we aim to incorporate additional multidimensional features, such as total packet volume and request type, in addition to user behavior to further enhance the recommendation accuracy and improve the effectiveness of the caching mechanism. Furthermore, we intend to apply feedback mechanisms to our prediction model and iterate continuously to enhance its adaptive capabilities.

Acknowledgments

Funding: This work was funded by the Strategic Priority Research Program of Chinese Academy of Sciences: Standardization research and system development of SEANET technology (Grant No. XDC02070100).

Thanks: We would like to express our gratitude to YanXia Li, Yang Li, and WenHan Lian for their meaningful support for this work.

References

[1] P. Mockapetris and K.J. Dunlap, "Development of the domain name system," Symposium proceedings on Communications architectures and protocols, pp.123–133, 1988.

[2] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A hierarchical name resolution service for information-centric networks," Proc. ACM SIGCOMM workshop on Information-centric networking, pp.7–12, 2011.

[3] J. Wang, G. Chen, J. You, and P. Sun, "SEANet: Architecture and technologies of an on-site, elastic, autonomous network," J. Netw. New Media, vol.6, pp.1–8, 2020.

[4] T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," Proc. 2007 conference on Applications, technologies, architectures, and protocols for computer communications, pp.181–192, 2007.

[5] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (NetInf) — An information-centric networking architecture," Computer Communications, vol.36, no.7, pp.721–735, 2013.

[6] J. Yang, Z. Mao, Y. Yue, and K. Rashmi, "{GL-Cache}: Group-level learning for efficient and high-performance caching," 21st USENIX Conference on File and Storage Technologies (FAST 23), pp.115–134, 2023.

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

[8] B. Jiang, P. Nain, and D. Towsley, "LRU cache under stationary requests," ACM SIGMETRICS Performance Evaluation Review, vol.45, no.2, pp.24–26, 2017.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol.38, no.2, pp.69–74, 2008.

[10] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," Journal of Network and computer Applications, vol.67, pp.1–25, 2016.

[11] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," IEEE J. Sel. Areas Commun., vol.36, no.6, pp.1300–1313, 2018.

[12] C. Dannewitz, M. D'Ambrosio, and V. Vercellone, "Hierarchical DHT-based name resolution for information-centric networks," Computer Communications, vol.36, no.7, pp.736–749, 2013.

[13] C. Dannewitz, H. Karl, and A. Yadav, "Report on locality in dns requests–evaluation and impact on future internet architectures,"

Univ. Paderborn, Paderborn, Germany, Technical Report TR-RI-12-323, p.19, 2012.
[14] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," IEEE Commun. Surveys Tuts., vol.17, no.3, pp.1473–1499, 2015.
[15] H. Guo, L.l. Rui, and Z.p. Gao, "A zone-based content pre-caching strategy in vehicular edge networks," Future Generation Computer Systems, vol.106, pp.22–33, 2020.
[16] K. Cho, M. Lee, K. Park, T.T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," 2012 Proceedings IEEE INFOCOM Workshops, pp.316–321, IEEE, 2012.
[17] M. Bilal and S.G. Kang, "Time aware least recent used (TLRU) cache management policy in ICN," 16th International Conference on Advanced Communication Technology, pp.528–532, IEEE, 2014.
[18] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," NSDI, pp.389–403, 2018.
[19] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," Proc. third workshop on Hot topics in software defined networking, pp.1–6, 2014.
[20] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," ACM SIGCOMM Computer Communication Review, vol.40, no.1, pp.92–99, 2010.
[21] G.C. Moura, J. Heidemann, R.d.O. Schmidt, and W. Hardaker, "Cache me if you can: Effects of DNS time-to-live," Proc. Internet Measurement Conference, pp.101–115, 2019.
[22] S. Alici, I.S. Altingovde, R. Ozcan, B.B. Cambazoglu, and Ö. Ulusoy, "Adaptive time-to-live strategies for query result caching in web search engines," Advances in Information Retrieval: 34th European Conference on IR Research, ECIR 2012, Barcelona, Spain, April 2012. Proceedings 34, pp.401–412, Springer, 2012.
[23] Y. Wu, C. DuBois, A.X. Zheng, and M. Ester, "Collaborative denoising auto-encoders for top-N recommender systems," Proc. Ninth ACM International Conference on Web Search and Data Mining, pp.153–162, 2016.
[24] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," Proc. Web Conference 2021, pp.1785–1797, 2021.
[25] H. Steck, "Embarrassingly shallow autoencoders for sparse data," The World Wide Web Conference, pp.3251–3257, 2019.
[26] Y. Li, J. Wang, and R. Han, "PB-NCC: A popularity-based caching strategy with number-of-copies control in information-centric networks," Applied Sciences, vol.12, no.2, 653, 2022.
[27] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networks," 2012 Proceedings IEEE INFOCOM Workshops, pp.268–273, IEEE, 2012.
[28] L. Saino, I. Psaras, and G. Pavlou, "Icarus: A caching simulator for information centric networking (ICN)," Proc. 7th International ICST Conference on Simulation Tools and Techniques, SIMUTOOLS'14, ICST, Brussels, Belgium, Belgium, ICST, 2014.
[29] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," Computer Networks, vol.57, no.16, pp.3128–3141, 2013.
[30] W.K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," NETWORKING 2012: 11th International IFIP TC 6 Networking Conference, Prague, Czech Republic, May 2012, Proceedings, Part I 11, pp.27–40, Springer, 2012.
[31] W. Zhao, Y. Qin, D. Gao, C.H. Foh, and H.C. Chao, "An efficient cache strategy in information centric networking vehicle-to-vehicle scenario," IEEE Access, vol.5, pp.12657–12667, 2017.

Zhaolin Ma received his B.S. degree in electronic information science and technology form Nankai Univresity in 2019. He is now a Ph.D. candidate in signal and information processing from Institute of Acoustics, Chinese Academy of Science (IACAS). His research interest include ICN resolution systems, software-defined networking (SDN), and distributed systems.

Jiali You received the Ph.D. degree in signal and information processing from the Institute of Acoustics, Chinese Academy of Sciences (IACAS), in 2008. From January 2015 and January 2016, she was a Visiting Scholar with the University of Massachusetts Amherst. She is currently a Professor in the National Network New Media engineering Research Center, IACAS. Her research interests include in-network processing and future network.

Haojiang Deng received the M.S. degree from 510 Institute of China Academy of Space Technology, Beijing, China, in 1998, and the Ph.D. degree from the Institute of Semiconductors, Chinese Academy of Sciences, Beijing, in 2001. He is currently a full Professor. His current research fields are digital signal processing in audio and video, and broadband multimedia communication.