CIT. Journal of Computing and Information Technology, Vol. 29, No. 4, December 2021, 219–234
doi: 10.20532/cit.2021.1005295

A Cloud-based Mobile Privacy Protection System with Efficient Cache Mechanism

Wenyun Dai[1] and Longbin Chen[2]

[1]Fairleigh Dickinson University, Teaneck, New Jersey, USA
[2]Alation Inc, Redwood City, California, USA

People increasingly rely on their mobile devices and use them to store a lot of data. Some of the data are personal and private, whose leakage leads to users' privacy harm. Meanwhile, mobile apps and services over-collect users' data due to the coarse-grained access control approach utilized by the mobile operating system. We propose a cloud-based approach to provide fine-grained access control toward data requests. We add privacy level, as a new metadata, to data and manage the storage using different policies correspondingly. However, the proposed approach leads to performance decreases because of the extra communication cost. We also introduce a novel cache mechanism to eliminate the extra cost by storing non-private and popular data on the mobile device. As part of our cache mechanism, we design a user-preference-based ordering method along with the principle of locality to determine how popular some data are. We also design a configurable refresh policy to improve the overall performance. Finally, we evaluate our approach using a real phone in a simulated environment. The results show that our approach can keep the response time of all data requests within a reasonable range and the cache mechanism can further improve the performance.

ACM CCS (2012) Classification: Security and privacy → Human and societal aspects of security and privacy → Privacy protections

Information systems → Information storage systems → Storage management → Hierarchical storage management

Computer systems organization → Architectures → Distributed architectures → Cloud computing

Keywords: security and privacy, access control, cloud based storage, hierarchical storage management

1. Introduction

Smart devices, especially smartphones, are nowadays increasingly popular. Some users consider their smartphones as personal computers, that could provide equal services to using traditional computers. Various kinds of data are stored in the devices, and some of them are sensitive and private. For example, bank account numbers and passwords, images with commercial confidentiality, physical addresses, and user profiles. Meanwhile, mobile apps must access these data to provide services, including some heavy-customized functionalities. The truth is that mobile data are over-collected due to the coarse-grained permission authorization and file management in mobile operating systems [1]. The data over-collection behaviors are not easily completely solvable. Firstly, only the user knows which particular data should be accessed. Then, there is no formal standard or law restricting the usage of mobile data. A lot of IT companies provide itemized terms and conditions about their usage of mobile data, but users barely read and review them carefully before "accepting all". It is not fair to expect all users to have enough patience and abilities to read and understand all the terms. This issue should lean more on to the technical side. It is our responsibility to provide users convenience not to add burdens.

The latest iOS, iOS 14, enhances privacy protection by introducing App Tracking Transparency function [2]. Users can check how their mobile data are accessed and tracked by apps and

websites. Meanwhile, this approach asks users to prepare data into a new folder before submitting or uploading online. This improvement helps reduce the data over-collection behaviors but cannot completely solve the problem. It is heavily user-involved, tedious, and not detailed enough. It is too complicated to figure out and detail intensive for ordinary users.

We have proposed a cloud-based data access control system, in which data are stored in the cloud storage and their access are fine-grained controlled [3]. Data are organized into different directories with different accessing policies based on their privacy levels [4]. All data requests are received by the control system. If the request and the application have been registered and granted before and the application's privacy level is not less than the data's privacy level, the requested data are returned. If the app's privacy level cannot match the data's privacy level, the request is denied, and the user must be involved to confirm the permission authentication. This operation updates the app's privacy level and is recorded in the control system to avoid repeated user involvements.

There are a lot of widely used cloud storage services nowadays, including business-oriented, such as Apple iCloud, Microsoft OneDrive, Google Drive, and Dropbox, and entertainment-related, such as Google Stadia, Nvidia GeForce Now, Microsoft xCloud, and Sony PlayStation Now. These products are user-friendly, convenient, and inexpensive. Meanwhile, a lot of companies and organizations provide their employees with free-to-use business accounts to use in order to store their data in the cloud storage on various devices anywhere [5]. These cloud storage services release the burden of storing massive data on mobile devices, but bring in extra communication costs [6]. It is impossible to achieve seamless performance theoretically compared to accessing local data. There are so many users complaining about the bad and unstable performance of cloud storage services, especially streaming services.

In this paper, we aim to use cloud storage with fine-grained access control to protect users' privacy of mobile data. We assign data and apps with different privacy levels. Different accessing policies are applied based on the privacy level. Furthermore, we optimize the control system for high performance. Considering the dynamic locality and time efficiency, we design and implement a cache mechanism to keep non-private and "popular" data within the mobile device. We further add a configurable refresh policy to improve the cache hit ratio. The rest of this paper is organized as follows. We include all the related work in Section 2. Section 3 discusses the design and implementation of our approach. Then we evaluate our approach and explain the results of the experiments in Section 4. We address the limitations of our current work in Section 5. Finally, we conclude our research in Section 6.

2. Related Work

2.1. Mobile Data Privacy Protection

We defined the data over-collection behaviors of mobile apps in our previous research [6]. We analyzed its motivation, common behaviors, and risks. We proposed a general solution of using cloud storage and did some simulation experiments. In this paper, we implement our approach on a real device and improve it with other strategies, including privacy-based storage policies and a cache mechanism.

C. Rottermanner *et al.* [7] analyzed several popular messaging apps on Android. They analyzed the encrypted communication in the transmission layer and attacks targeting the metadata. They also checked the message storage on mobile devices. They found that all observed apps request over 7 permissions that could leak private information about users. They were only focused on messengering apps, such as WhatsApp, Line, WeChat, Telegram, and TextSecure, but similar privacy issues are also common in other apps. In our research, we directly deal with the mobile data, that could be used by various apps. Z. Almusaylim and N. Jhanjhi [8] studied privacy protection in location-aware services of mobile cloud computing. They analyzed the challenges and addressed some possible solutions with multi-location queries, multi-authority, location compression, and user revocation. The similarity of our research is that we provide management of multiple data types, while they focused on location data, and we deal with all mobile data that can be stored in the cloud storage.

B. Zhang and H. Xu [9] aimed to help users make decision towards privacy-related permis-

sion. They applied two permission interfaces, the frequency nudge, and the social nudge, into app settings of smartphone. Their research can raise users' attention about privacy. The permission granting is user-involved, which is not suitable for all ordinary users. The two interfaces they proposed can only work on installed apps. Our approach protects users' privacy with fewer users' involvements. C. Yin et al [10] applied a logistic regression for local differential privacy protection. Their approach was based on human-centric computing and heavily relied on preprocessing and formatting. These two steps were time-consuming, let alone the machine learning method itself spent time to model and test. There are some other related research works using machine learning techniques, such as a reinforcement learning approach proposed by M. Zhang et al [11]. We also apply one machine learning method, which is time-series regression, to predict the usage pattern of large data types. We only need to track the periods without any preprocessing or formatting, so our approach is simple and fast.

More recently, there are some research works applying blockchain technology to protect mobile users' privacy. Z. Sun *et al.* [12] used a double disturbance localized differential privacy algorithm to disturb the location information. Then they uploaded all the sensing data to the blockchain through edge nodes, which would be processed by the cloud and returned to the requester. Y. Chen *et al.* [13] used the K-anonymity and searchable encryption techniques for medical data sharing among medical institutions and users. T. Feng *et al.* [14] combined zero-knowledge proof and smart contract to verify the availability of data between data owners and the cloud service providers. They further used proxy re-encryption technology for secure sharing among authorized cloud service providers. H. Wang *et al.* [15] proposed a credit value solution using the multiple-attribute decision-making algorithm on the blockchain. The behaviors of requestors and participants could lead to rewards or punishment. Blockchain technologies, especially hashing encryption and distribution, help with privacy protection, but it also bring in extra communication and calculation costs. Our goal is to achieve nearly seamless performance mainly relying on the access control policies but not encryption.

2.2. Fine-Grained Access Control

T. Baseri *et al.* [16] proposed a multi-authority attribute-based access control scheme to support the coexistence of authorities. The proposed scheme used the dynamic location of mobile users as contextual information about those users, employed location range constraints as a policy in attribute-based encryption, and authorized users with dynamic locations satisfying access policies. Their work focused on location data and access control. There are some other research works using attribute-based access control [17, 18, 19, 20, 21]. Our approach works for all data and provides fine-grained access control without encryption.

S. Saroiu *et al.* [22] proposed an approach letting users attach Terms of Service (ToS) to their data being uploaded to the cloud. They implemented their approach using policy-carrying data that guaranteed that the cloud providers claimed they were compliant with the ToS before accessing the data. This approach is heavily user-involved, which is good for professional users but not ordinary users. S. Lee *et al.* [23] proposed a platform to protect confidentiality when employees used their own apps to create, edit, and share corporate documents. Their approach provided fine-grained data object sandboxes and access control in the form of documents. Every document should be assigned with detailed authentic information for all the participants. The access control towards some data was managed by its owner. However, in their platform, one app instance cannot read or write multiple documents, and they were focused on documents. In our research, we deal with all kinds of data, and the data operation is separated from the data itself. We focus on the permission authorization that occurs before data operation.

2.3. Cache Hit Ratio

The most iconic cache replacement algorithm is the Least Recently Used (LRU) policy. There are some modified versions of LRU to further improve the cache hit ratio in mobile computing. Y. Ryu [24] proposed a PRAM-aware block-based LRU scheme. His research focused on minimizing the amount of write operations on PRAM and the number of erase operations on the flash memory. His approach worked in an offline hybrid memory environment, including

the traditional main memory and the flash memory. We consider the mobile device as the cache and the cloud storage as the memory. We also consider the network communication cost. G. Hasslinger *et al.* [25] combined the simple update effort of the LRU policy with the flexibility to keep the most important contents in the cache. They used a predefined score function to rate the importance of web content. In our research, we use users' preferences with data types along with pure LRU to determine the importance of data. Meanwhile, their work focused on the web contents, while we worked on the mobile data.

There are some other research works about the improvement of the cache hit rate for mobile data streaming. C. Li *et al.* [26] proposed a QoE-driven mobile edge caching placement optimization for dynamic adaptive video streaming. They maximized the aggregate average video distortion reduction of all users while minimizing the additional cost of representation downloading from the base station, subject not only to the storage capacity constraints of the edge servers but also to the transmission and initial startup delay constraints of the users. E. Zeydan *et al.* [27] introduced a proactive caching architecture for the 5G network. They processed massive data on a big data platform and used machine learning tools for content popularity predictions. A. Rocha *et al.* [28] designed a data stream caching algorithm, DSCA, to maximize the cache hit rate of Content-Centric Networks (CCN) by incorporating content popularity in caching decisions. DSCA coped with dynamics in content popularity while operating under the memory and high processing rate constraints of CCN network routers. They used a data streaming algorithm to identify the most popular contents in a windowed manner. X. Li *et al.* [29] characterized the problem of power consumption and video streaming in mobile systems. They proposed GreenTube to reduce power consumption with a dynamic cache management algorithm to adjust the high threshold value of network speed and expected accessing time. Our research work is focused on all mobile data, not only streaming types. Furthermore, we not only consider the popularity but also the privacy levels.

There are also some researchers focusing on cache management in the Internet of Things (IoT) and Named Data Network (NDN). M. Naeem *et al.* [30] proposed a hybrid strategy for efficient data delivery. They aimed for average latency, cache hit ratio, and average stretch ratio. T. Peng *et al.* [31] considered the characteristics of cache files to avoid cache pollution problems. They introduced the file cache value and a file cache value-aware cache replacement algorithm correspondingly. J. Hou *et al.* [32] proposed a Graph Neural Network (GNN) based cache strategy to improve caching performance in NDN. They first extracted time-series features and then applied GNN to make cache probability predictions. In our research, we have not considered NDN, since we mainly focus on the personal data stored in cloud storages and local devices. We use cloud storage as the remote storage without caring about its own infrastructure. However, NDN is a topic worth considering that we might study in the future.

3. Design and Implementation

Our access control system consists of two main functionalities along with several supporting services. The first main functionality is to apply different access policies towards data and requests with different privacy levels. The second one is to respond to data requests with the optimized cache mechanism. The entire system framework is shown in Figure 1. The Privacy Control module manages data referring to their privacy levels and configures the storage policies. The Access Control module determines whether a data access request from an app should be authorized. Then the Mapping module checks the storage location of the requested data. If it is stored locally on the device, the app can directly access the data without further communication with the cloud storage. If the data is stored remotely, the Mapping module generates the real directory for data with the help of metadata and privacy control.

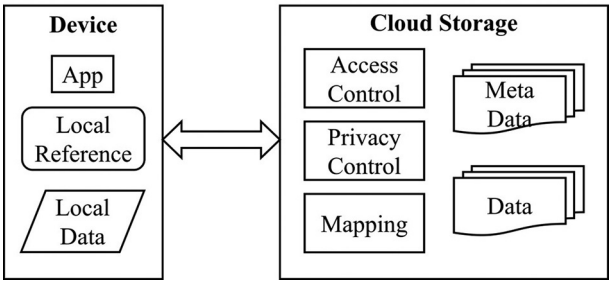

Figure 1. System Framework.

3.1. Access Policies with Privacy Levels

We assign data and requesters with several privacy levels, from 1 to 3. The privacy level of data indicates how private the data is. To release users' burden of setting up the privacy level for each piece of data, we assign the default privacy level values to data based on their types. Mobile data can be categorized into three groups, persistent, temporary, and intermediate. The persistent data should be stored in non-volatile storage and requested multiple times. Some typical persistent data are images, files, contacts, and messages. The temporary data are not necessary to be stored permanently, and they are normally requested just once. For example, location data are temporary, since new location data should be captured and returned every time a request is made. The intermediate data are managed within apps or system functions, such as memes in a chat app. These data are stored in some special directory within the mobile device. Our research focuses on persistent data. We set the default privacy level of the persistent data to 1. Meanwhile, we assign portions of data with the 2 and 3 privacy levels. We also apply different workloads with different privacy level assignments in the experiments.

The privacy level of a requester records the highest privacy-level data it has successfully accessed. The default privacy level value is set to 1. The user must be involved to permit if some app requests some higher privacy level data than its own privacy level. If the user confirms the access permission, the app's privacy level will be updated to the privacy level of the data. For example, an app with privacy level 1 requests a piece of data with privacy level 2. If the user authorizes the access, this app's privacy level will be 2. Otherwise, it is still 1. Any data access request from an equal or higher privacy level app is automatically granted without the user's involvement.

Data are stored in the cloud storage under different folders based on their privacy levels. We apply different storage and access policies in the form of the folder. Firstly, the same type of data with the lowest privacy level, 1, are stored in one folder. Once a request from some app has been authorized, this app can access all data under this folder. In other words, we return the access permission to the whole folder back to the requesting app. This is the same way the current mobile operating systems do. Secondly, the same type of data with the privacy level 2 are further separated into subfolders based on their contents. For example, images can be separated into people, scenery, document, and others. Once a request from some app has been authorized, this app can only access all data under the content subfolder. Lastly, the same type of data with the privacy level 3 are stored in a special folder, in which each piece of data is returned to one particular request.

We apply a RESTful interface to encapsulate the apps' requests which consist of request information, app information, and data information. The Access Control module, shown in Figure 1, checks the registration status of the app and data. If the app has not been registered before, the user must be involved. The user can select whether to grant this particular access request. If the app has been registered but is on the blacklist, its request gets rejected automatically. Otherwise, the Access Control module compares the app's privacy level with the requested data's privacy level, as discussed in the above paragraphs and the Algorithm 1.

Algorithm 1. Granting app's data accessing requests and updating related information.

Algorithm PrivacyCheck(*a*, *d*):
Input: The requesting app *a* and the requested data *d*;
Registered app list, *aList*;
Blacklist, *bList*.
Output: Boolean result of whether to grant app *a*'s request towards data *d*.

if *a* is in *aList*
  if *a* is in *bList*
    return *false*
  else if *a.pLevel* ≥ *d.pLevel*
    return *true*
else
    involve the user to decide
    isAllowed ← user's selection
if *isAllowed* = *true*
  a.pLevel ← *b.pLevel*
return *isAllowed*

3.2. Cache Mechanism

We design our own "cache" mechanism to improve the system performance by reducing the

communication cost between mobile devices and cloud storage. We keep some data stored in the mobile device locally. Any legit app requesting some local data can access the data directly. The data storage is implemented based on locality and user's behavior, along with consideration of privacy, to make judicious decisions. Meanwhile, we introduce a refresh mechanism to improve the cache hit ratio.

3.2.1. Cache Replacement Strategy

The cache in our system is the mobile device itself. We leave some non-private data in the mobile device, which is reasonable considering the trade-off between extra communication cost and privacy protection. Fortunately, more private data are less likely to be used frequently. For example, we do not need our SSNs every day. As a result, the first requirement of the local data is that its privacy levels are 1. The second aspect is about the cache replacement strategy, which data should be kept in the mobile device and which data can be removed when the cache is full. The most commonly used rule is the principle of locality [33]. The performance is better if more "popular" (most recently or frequently accessed) data are stored locally. We try to keep the most recently and frequently used data in the mobile device. However, the frequency is counted for each data type, not single data, considering the specialty of our system and users' behaviors.

Different users have different behaviors using mobile devices. For example, a user may use music data most frequently on weekdays and may use pictures and videos most frequently on weekends. Compared to the traditional memory-cache systems, that apply paging to divide cache, memory, and data into the same fixed-size blocks, our system does not divide data to keep the integrity of data. If we divide data into smaller pieces, it is easy to apply replacement, but it is too complicated to store it in the mobile device and the cloud storage. It is also very tedious to maintain the integrity and to keep everything reliable when it comes to dealing with cloud storage. As a result, we keep the data at its original size and design a new method to solve the replacement problem based on the user's behavior.

To analyze the user's behavior, we record data access requests at the beginning of each refresh cycle and equally assign cache space for all data types. First, we obtain the user's preference for different data types. Based on the user's preference, we give higher priority to the data type that the user prefers. The higher priority data type can preempt some space from the lower priority data type. If the assigned cache space for the highest priority data type is full, we will migrate some space from the space of the lowest priority data type. We try to keep all the highest preferred data until the whole cache is full.

Algorithm 2. Push the newly requested data to cache and update related information.

Algorithm UpdateCache(*d*):
Input: The new requested data *d*; data type frequency list *dtfList*; Deque for each data type *dtDq*; Cache size for each data type *csList*.
Output: store data *d* to cache

if *csList(d.type)* < *d.size*
  t ← the minimum in *dtfList*
  while *csList(t)* < *d.size*
    pop from *dtDq(t)*
    csList(t) ← *csList(t)* − *d.size*
    csList(d.type) ← *csList(d.type)* + *d.size*
push *d* to *dtDq(d.type)*
dtfList(d.type) ← *dtfList(d.type)* + 1

Algorithm 2 shows the detailed steps of pushing the newly requested data to cache and updating the related information. First, we check whether the corresponding cache can host the new data. If it is large enough, we will push the new data to the corresponding deque and update the frequency list for the type of the new data. If the corresponding cache is not large enough, we will borrow some space from the lower priority cache space. We find the minimum frequency data type has the lowest priority. Then we start the iteration to pop data from the lowest priority cache space until it has enough space for the new data. Then we move the newly empty space from the lowest priority cache to the cache for the requested data's type. Eventually, we push the new data to the corresponding deque and update the frequency list for the type of the new data.

3.2.2. Cache Refresh Strategy

There are several important differences between our cache mechanism with the traditional memory-cache strategy. The most important one is that the mobile device storage does not refresh as the memory does. Once we turn off the computer, all contents in the memory are erased. However, the data stored in mobile device are permanent even when the device is turned off. In other words, the contents stored in the mobile device won't be erased or refreshed. In our design, we refresh the mobile storage with configured variables that are calculated based on the user's behavior. Three factors are discussed as following:

1. Percentage of refreshed data. After obtaining the user's preference towards data types, we use the variable, α, to be the percentage of the most frequently used data type being refreshed. For example, some user uses pictures the most, and, thus, we set the α to value 0.75. Once a refresh occurs, we will remove 75% of pictures from the cache. For the data type that is not the most frequently used, we erase of it from the cache.
2. Refresh frequency. We use the variable, β, to determine how often to refresh the cache periodically. This value is flexible and it is mainly determined by the user's behavior pattern. For example, β is one week if the user's behavior pattern changes during a week. Meanwhile, we set up the lower and upper bound of β to be is 1 day and 1 month respectively.
3. Timeliness of special data is marked with δ. In our previous research, we found that there were some data that were not so frequently used within one refresh cycle, but rather they were used in almost all refresh cycles. For example, someone always listens to a music list stored on the mobile phone on the way to work. However, that person does not listen to music during working hours. If the refresh frequency is one day, all the music is loaded every morning and gets erased every night. It is obvious that it does not make much sense to keep the data in the device until the cache is full. To make the cache space more efficient, we use a simple time-series regression method to predict the timeliness. We do not use other sophisticated machine learning methods that can provide more accurate predictions, since the running time is very important in our system. Meanwhile, we just consider the timeliness factor for large data, such as music and videos. These types of data are more likely to follow the special pattern as we mentioned above.

In summary, the refresh workflow is shown in Figure 2. Basically, we keep handling data requests until the refresh rate β occurs. Meanwhile, we check large data to see whether they follow the special usage pattern mentioned above. We mark the data with δ if they do and will erase them once they are not used without starting a new refresh phase. The refresh occurs every at a β time interval, and it erases α of the most frequently used type of data and all other types of data.

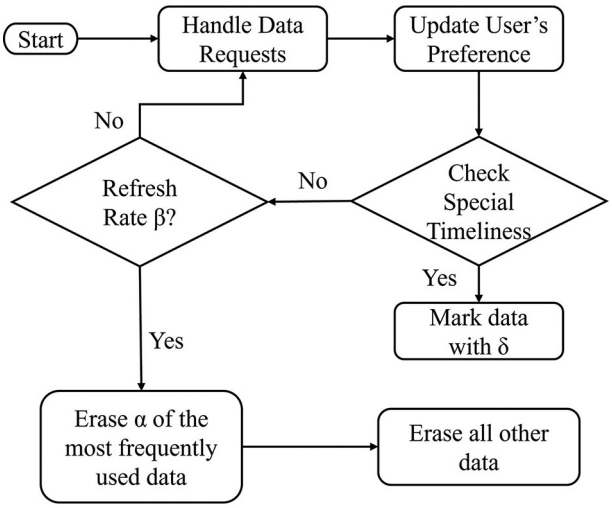

Figure 2. The workflow of the refresh process.

3.3. Mapping Mechanism

The Mapping module in our system has several functionalities. The first one is to keep local and remote references towards data consistent. In our system, data could be stored in cloud storage or mobile device. We design a set of standardized interfaces to request data and receive data. The local references provide the directory of local data stored in the mobile device. They are synchronized with the Mapping module in

the form of RESTful resources. Meanwhile, the data stored in the cloud storage are also accessed by the same format of RESTful protocol.

The second functionality of the Mapping module is to extract the metadata from data to improve privacy protection. The metadata of data along with the privacy level, added into our system, is managed by the Mapping module but also stored in the cloud storage. We use XML files to describe the metadata in order to match the RESTful requests from mobile apps. The metadata are used by the Privacy Control module to determine the granting of permissions. After that, the Mapping module will send the real directory to the requesting app.

The third functionality is to work with the Access Control module to determine which folder directory a piece of data should be stored in and be returned to the requesting app. As we mentioned in Section 2.1, data are stored in different folders based on their content types and privacy levels. If some data are with privacy level 1, the Mapping module stores them in the common folder and will return the whole folder directory to the requested app. If some data are with privacy level 2, the Mapping module stores them in the corresponding content subfolder and will return the content subfolder directory to the requested app. For the data with privacy level 3, the Mapping module stores them individually in the special folder and will only return the requested data.

4. Experiments and Results

4.1. Experimental Environments

We use one Google Pixel 4a (5G) as the experimental mobile device. It is running Android 11 and with 128GB storage. We define the cache to be 2GB in size. We prepare a server to simulate the cloud storage. We can fully customize storage policies and communication interfaces with our own RESTful protocol. The server and the mobile device are connected to the same router. We implement a set of testing mobile apps with our own APIs in Android Studio 4.1.1. These apps request different data using the same format of the RESTful protocol. Meanwhile, we prepare several different kinds of data, including images, audio, videos, contacts, and emails. We generate testing pseudo data with the same size for each type. Table 1 shows the unit size and amount for each data type.

Table 1. Experimental data size and amount.

Data Type	Unit Size	Amount	Total Size
Image	4 Mb	1000	4000 Mb
Audio	8 Mb	300	2400 Mb
Video	500 Mb	50	25 Gb
Contact	1 Mb	1000	1000 Mb
Email	2 Mb	1000	2000 Mb

Meanwhile, we assign the experimental data with different privacy levels. The percentage of five types of data with different privacy levels are shown in Table 2. We consider that 60% of images are not private, such as portraits and scenery. Furthermore, 30% and 10% of images are private and extremely private respectively. For example, images with personal addresses are private and copies of personal information are extremely private. Most audio records are not private, while there are some of them that can be considered private or extremely private, such as private call recording. Half videos are not private. They could be movies or downloaded videos. We consider that 30% of videos are private, such as homemade videos that should not be shared with others, and 20% of videos are extremely private, such as confidential meeting recordings. Most contacts are private or extremely private because they include a lot of personal information. Over half of emails are private or extremely private, such as business or confidential emails.

Table 2. Experimental data privacy levels.

Data Type	Privacy Level 1	Privacy Level 2	Privacy Level 3
Image	60%	30%	10%
Audio	80%	15%	5%
Video	50%	30%	20%
Contact	10%	40%	50%
Email	30%	30%	40%

We record three users' real usage regarding the five types of data, mentioned in Table 1, and then delete the duplicate information and generate the workload to simulate different behaviors. We prepare three sets of workloads of accessing mobile data based on real users' behaviors. The first workload is image intensive, in which there are 70% of requests for images. The second workload is emails and contacts intensive, in which there are 80% requests for contacts and emails. In the third workload, data requests are equally distributed to the five data types.

4.2. Privacy Protection Experiments

To evaluate the strength of privacy protection of our approach, we use a value, risk grade, by multiplying the possibility of leakage and the range. We assume that a single piece of data has a 50% chance of being leaked if it is stored on a mobile device. Meanwhile, it has a 20% chance of being leaked if it is stored in the private folder and 10% for the special extremely private folder in the cloud storage. A high-risk grade indicates that the user's privacy is highly likely to be harmed. In other words, the lower the risk grade is, the better the user's privacy is protected. For the original approach, once some app gets permission to access some type of data, it can access all data with the same type. The range is the total amount of stored data in the same folder.

Table 3 shows our experimental results regarding the risk grade. For the original approach, there is no separation of privacy levels, thus we set their privacy levels to be 0. The possibility of each type of data being leaked is the same as that of a single piece of data, but the range is large. Due to the coarse-grained access control, the leakage of one piece of data can lead to the leakage of all data of the same type. As a result, the risk grades are very high, as the first five cells of the last column in Table 3. From the remaining cells of the last column, we can see that our approach decreases the risk grades significantly for all types of data. The first benefit of our approach is the data separation based on the privacy level. This method reduces the possible leakage range greatly. Even if we leave all non-private data in the mobile device, the risk grades are lower than that of the original approach. For example, non-private images in our approach have 300 risk grade that is less than the 500 risk grade in the original approach. For the private and extremely private data, our approach can decrease the risk grades tremendously. That is the second benefit, which is the fine-grained access control provided by cloud storage. Extremely private images, audio, videos, contacts, and emails have risk grades of 10, 1.5, 1, 50, and 40 respectively. Compared to the original risk grades 500, 150, 25, 500, and 500, our approach improves the privacy protection 10 to 100 times.

4.3. Response Time Experiments

Our approach can improve mobile data privacy protection, as shown in Table 3, but there is always a tradeoff between performance and complexity of access control. Our approach utilizes cloud storage to provide fine-grained access control. There must be extra communication costs that degrade the performance. We use the response time to evaluate the performance. We divide the whole workload into 100 intervals and record the average response time of all requests within one interval in Android Studio. We set the refresh rate α to be every 10 intervals and the most frequently used data type being erased, β, to be 0.5. The mark δ is only used for audio and video data.

Figure 3 shows the results of response time running the first workload using the original approach, shown by the dashed line, our approach without cache, shown by the dashed line with triangle, and our approach with cache, shown by the solid line with squares. Basically, the original approach has the smallest and the most stable response time. The average response time of the original approach is 6.3 ms. The average response time of our approach without cache is 15.5 ms and that with cache is 11.4 ms. The cache mechanism achieves about 26.5% performance improvement. At the beginning, there is no huge difference between our approach with and without cache. However, the performance increases once some data have been stored in the cache. For example, from workload 70 to 101, the gap between without cache and with cache is huge, and our approach with cache has a closer response time than the original approach. Even though our approach spends more

time obtaining data, it is still reasonable. A response time under 100 ms can offer users an instant response. Furthermore, our approach can achieve nearly the same performance if the data are stored in the mobile device. The slight delay is because our approach uses RESTful protocol to request and fetch data, which is slightly slower than the original API calls in Android. There are four recorded response times greater than 20 ms since there are audio or video data requested within those intervals. Downloading these large data is time-consuming, but it runs much faster if these data have been stored in a cache.

Then the experiment results of running the second workload, which is typical business-like behavior, are shown in Figure 4. Even the original approach has better performance, but our approach works better than during the experiment on the first workload. The average response time of our approach with cache is 2.7 ms, the average response time without cache is 4.3 ms, and the average response time of the original approach is 1.1 ms. The cache mechanism achieves about 60% performance improvement over the without cache approach. The majority of the second workload requests contacts or emails, that are relatively small. As a result, downloading them from cloud storage is not time-consuming. That makes our approach faster.

Table 3. Risk grade experiment results.

Approach	Data Type	Privacy Level	Possibility	Range	Risk Grade
Original	Image	0	50%	1000	500
	Audio	0	50%	300	150
	Video	0	50%	50	25
	Contact	0	50%	1000	500
	Email	0	50%	1000	500
Our Approach	Image	1	50%	600	300
		2	20%	300	60
		3	10%	100	10
	Audio	1	50%	240	120
		2	20%	45	9
		3	10%	15	1.5
	Video	1	50%	25	12.5
		2	20%	15	3
		3	10%	10	1
	Contact	1	50%	100	50
		2	20%	400	20
		3	10%	500	50
	Email	1	50%	300	150
		2	20%	300	60
			10%	400	40

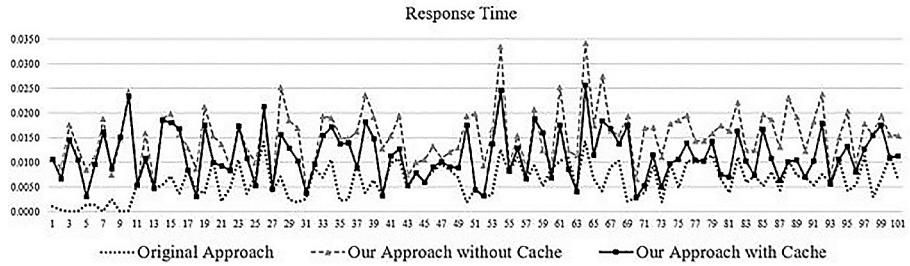

Figure 3. Experiment results of response time running the first workload.

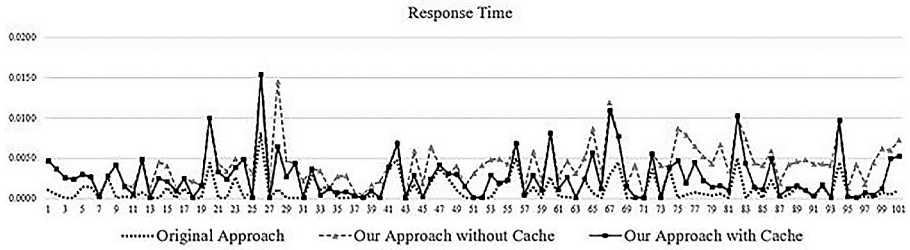

Figure 4. Experiment results of response time running the second workload.

The last workload is the one with equal data requests among five types. Figure 5 shows the results of this experiment. The average response time of our approach with cache is 7.8 ms and 10.6 ms without cache. The average response time of the original approach is 3 ms. Our approach works the worst compared to the original approach among the three workloads. In fact, this workload cannot reflect reality, and it is generated by us just for experimental purposes. It is nearly impossible that some user accesses the data in an equally distributed manner. There are always behavioral patterns, that may not be consistent forever. We keep this workload as a part of theoretical analysis.

4.4. Cache Hit Ratio Experiments

In the previous experiments, we set α to the value of every 10 intervals and β to 0.5. However, these variables should +impact the cache hit ratio performance. In this set of experiments, we aim to see the relationship between the cache hit ratio and variables α and β.

4.4.1. Variable α Experiments

We first fix β to be 1, indicating each refresh cycle will erase all data in the cache, to test the different α values. We set α to 0, 0.25, 0.5, and 0.75 in our experiments. We only run the first and second workloads for this part of the experiments, because the cache hit ratio is related to the workload. Meanwhile, the workload should reflect the user's behavior pattern, but the third workload does not follow some pattern.

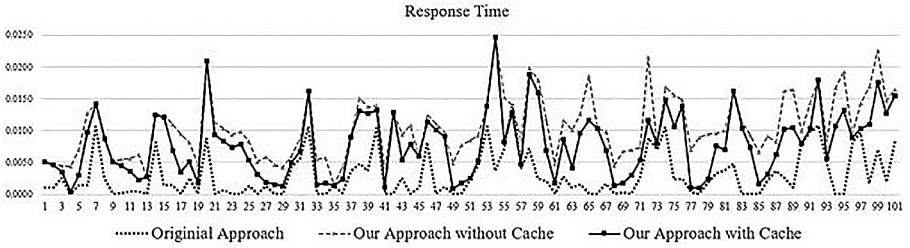

Figure 5. Experiment results of response time running the third workload.

Figure 6 shows the results of running the first workload with $\alpha = 0$, 0.25, 0.5, 0.75, and $\beta = 1$. It is straightforward that the cache hit ratios for each configuration are the same if no refresh has occurred. For example, from workload intervals 0 to 25, all four configurations share the same performance. At workload intervals 25, 50, and 75, we can see a cache hit ratio to decline. That is caused by the refresh, which erases all data in the cache. After these declines, the cache hit ratio starts to incline very soon. Furthermore, there are some other interesting findings. If we refresh the cache more frequently, the following cache hit ratio increases faster. For example, from workload interval 25 to 50, the configuration with $\alpha = 0.25$ increases faster than others. We believe that the reason is that the cache has been emptied and made more room for recent data. Similarly, from workload interval 80 to 100, the configuration with $\alpha = 0.75$ catches up with the configuration with $\alpha = 0$ and performs better after workload interval 88.

From the results of the configuration $\alpha = 0$, we can see that the cache hit ratio starts to decline around workload intervals 20 and 40. After workload interval 60, the cache hit ratio becomes stable. As a result, we set the α to 0.2 and compare the results with $\alpha = 0.25$, which has the best results in the previous experiment. The results are shown in Figure 7. Value of $\alpha = 0.25$ indicates more refresh, so there is more pronounced cache hit ratio decline than in case when $\alpha = 0.2$. However, it always catches up quickly and surpasses $\alpha = 0.25$. On average, $\alpha = 0.2$ can achieve a higher cache hit ratio than $\alpha = 0.25$. If we can accurately know the user's behavior pattern, which directly affects the refresh rate, we can obtain the best cache hit ratio. However, it is nearly impossible to know this information in advance. One alternative way is to predict users' behaviors based on their historical data usage records, but this method does not work well in streaming-related services.

Furthermore, we use the same configuration, in which $\beta = 1$ and $\alpha = 0$, 0.25, 0.5, and 0.75, to run the second workload. To show their differences clearer, we filter the results by only showing workload intervals from 20 to 100. All configurations share the same performance before refresh occurs, which happens at 20 at the earliest. First, the average cache hit ratio of the second workload is higher than that of the first workload. It is because the cache can store much more contacts and email data than images, audio, and videos. Then, there is no obvious pattern found in the second workload. The user may frequently access a pool of contacts and emails, and that person also uses other contacts and emails with less frequency. The configuration with $\alpha = 0$ has the best cache hit ratio most of the time, but $\alpha = 0.75$ surpasses it at the

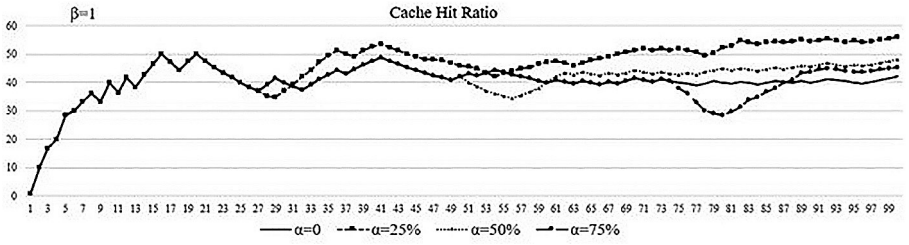

Figure 6. Experiment results of cache hit ratio running the first workload with $\beta = 1$.

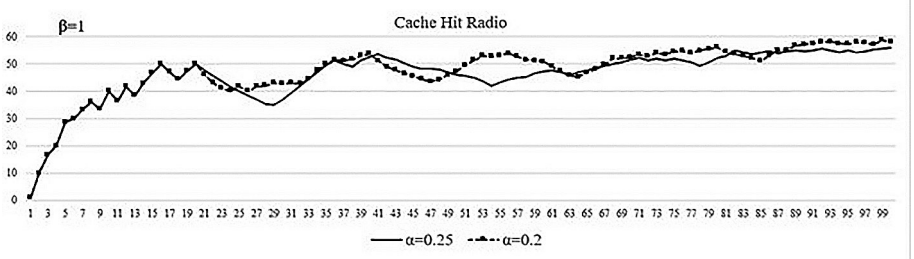

Figure 7. Experiment results of comparison between $\alpha = 0.25$ and $\alpha = 0.2$.

workload interval 87. We can conclude that it is not always good to refresh the cache without considering the usage pattern.

4.4.2. Variable β Experiments

For the first workload, we fix α to be 0.2 and test the cache hit ratios with different values of β including 0, 0.25, 0.5, and 0.75. Value $\beta = 0$ means all the data with the most frequently used type are kept in each refresh cycle.

To compare the performance of different β values, we also include $\beta = 1$ in the experiment, which has been discussed in Section 3.4.1. The results are shown in Figure 9. The configuration with $\beta = 0.25$ has a very similar performance with $\beta = 0.5$. If β is 0.75 or 1, the cache hit ratio declines at the beginning of each refresh cycle. That is because new data requests cannot find local references in the cache. The cache hit ratio of $\beta = 0.75$ is still much better than $\beta = 1$, since there are some popular data left over in the cache that can satisfy some data requests. Comparing $\beta = 0.5$, 0.25, and 0, we can find that their performances also decline at the beginning phase of each refresh cycle, but the decline happens at different times. The first performance decline of $\beta = 0.5$ happens at workload interval 23. The first performance decline of $\beta = 0.25$ happens at workload interval 25. The first performance decline of $\beta = 0$ happens at workload interval 24. Meanwhile, if the performance decline happens later, the following performance incline is better. Considering this point, there is best performing configuration for all test cases. At the first refresh cycle, $\beta = 0$, $\beta = 0.25$, and $\beta = 0.5$ are similarly good. At the second refresh cycle, $\beta = 0.5$ is the best. At the third refresh cycle, $\beta = 0.25$ and $\beta = 0.5$ are almost the same good. At the last refresh cycle, $\beta = 0.25$ has the best performance.

5. Limitations and Future Work

There are several limitations of our current research work. First, we have not applied any encryption strategy in the system considering the extra cost brought by the encryption and decryption methods. We aim for all mobile devices, and we do not expect that all of them have powerful computation abilities. We follow the design idea to keep mobile devices only with their original and most fundamental functions. However, encryption has great importance to data protection. Meanwhile, mobile devices are increasingly powerful and widely proliferated with the help of rapidly growing manufacturing. In section 2, we mention some related work using encryption methods. In the future, we plan to apply some lightweight encryption and decryption policy to further improve privacy protection.

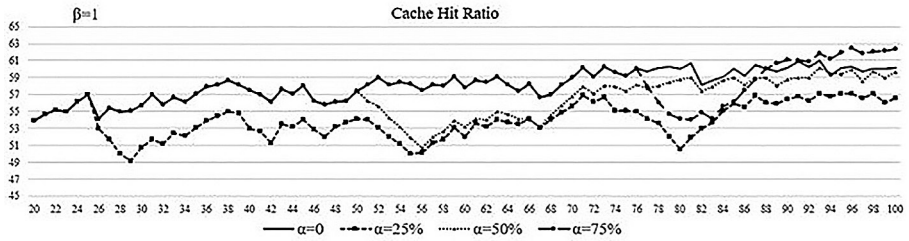

Figure 8. Experiment results of cache hit ratio running the second workload with $\beta = 1$.

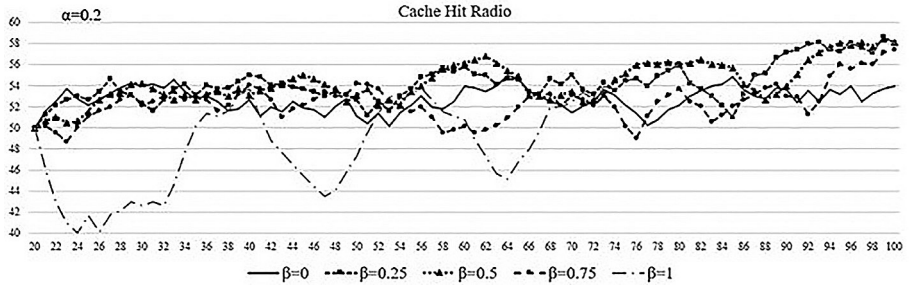

Figure 9. Experiment results of cache hit ratio.

Secondly, we use two variables, α and β, in the cache refresh strategy. The variable α is about users' preference towards data types, while the variable β heavily depends on the accuracy of users' behavior pattern prediction. Theoretically, it is impossible to guarantee 100% accuracy and may bring massive extra computation costs. Thus, we only test some pre-defined values. In the future, we also plan to find the best trade-off between accuracy and cost for these two variables, especially β.

Lastly, the cloud storage and mobile data are simulated in our experiments. We use a server that we can fully control to execute our system, but we never have equivalent rights to using a public cloud storage service practically. In other words, our system does not work without efforts from the cloud storage service providers if users choose to use any public cloud storage service. We generated the testing mobile data with one size for one type. For example, all images are 4Mb. The mobile data usage is real but from a small group of users. This also cannot reflect reality. In the future, we will acquire additional mobile data and more real behavior from a larger group of users to evaluate our system.

6. Conclusion

In this paper, we proposed a novel cloud-based storage and fine-grained access control system. Mobile data are separated based on new metadata and privacy level, and they are stored in different folders with different access granularities. To eliminate extra communication costs, we introduced a cache mechanism to store popular and non-private data on mobile devices. We apply users' preferences towards data types, along with a configurable refresh policy, to improve the cache hit ratio. Results of experiments showed that our approach with the cache mechanism could maintain efficient performance while greatly improving users' privacy protection.

References

[1] W. Dai *et al.*, "DASC: A Privacy-Protected Data Access System with Cache Mechanism for Smartphones", in *Proc. of the IEEE Wireless and Optical Communications Conference (WOCC)*, 2020, pp. 1–6. https://doi.org/10.1109/WOCC48579.2020.9114939

[2] Apple, "App Tracking Transparency", Apple, 2021. https://developer.apple.com/documentation/apptrackingtransparency

[3] W. Dai *et al.*, "DASS: A Web-Based Fine-Grained Data Access System for Smartphones", in *Proc. of the IEEE International Conference on Smart Cloud (SmartCloud)*, 2017, pp. 238–243. https://doi.org/10.1109/SmartCloud.2017.45

[4] W. Dai *et al.*, "A Privacy-Protection Data Separation Approach for Fine-Grained Data Access Management", in *Proc. of the IEEE International Conference on Smart Cloud (SmartCloud)*, 2017, pp. 84–89. https://doi.org/10.1109/SmartCloud.2017.20

[5] W. Dai *et al.*, "Cloud Infrastructure Resource Allocation for Big Data Applications," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 313–324, 2018. https://doi.org/10.1109/TBDATA.2016.2597149

[6] W. Dai *et al.*, "Who Moved My Data? Privacy Protection in Smartphones", *IEEE Communications Magazine*, vol. 55, no. 1, pp. 20–25, 2017. https://doi.org/10.1109/MCOM.2017.1600349CM

[7] C. Rottermanner *et al.*, "Privacy and data protection in smartphone messengers", in *Proc. of the Int. Conference on Information Integration and Web-based Application & Services (iiWAS'15)*, 2015, pp. 1–10. https://doi.org/10.1145/2837185.2837202

[8] Z. Almusaylim and N., Jhanjhi, "Comprehensive Review: Privacy Protection of User in Location-Aware Services of Mobile Cloud Computing", *Wireless Personsal Communication*, vol. 111, pp. 541–564, 2020. https://doi.org/10.1007/s11277-019-06872-3

[9] B. Zhang and H. Xu, "Privacy Nudges for Mobile Applications: Effects on the Creepiness Emotion and Privacy Attitudes", in *Proc. of the ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW'16)*, 2016, pp. 1676–1690. https://doi.org/10.1145/2818048.2820073

[10] C. Yin *et al.*, "Local Privacy Protection Classification Based on Human-centric Computing", *Human-centric Computing and Information Sciences*, vol. 9, no. 1, p. 33, 2019. https://doi.org/10.1186/s13673-019-0195-4

[11] M. Zhang *et al.*, "Dynamic Pricing for Privacy-Preserving Mobile Crowdsensing: A Reinforcement Learning Approach", *IEEE Network*, vol. 33, no. 2, pp. 160–165, 2019. https://doi.org/10.1109/MNET.2018.1700468

[12] Z. Sun *et al.*, "A Two-Stage Privacy Protection Mechanism Based on Blockchain in Mobile Crowdsourcing", *International Journal of Intelligent Systems*, vol. 36, no. 5, pp. 2058–2080, 2021. https://doi.org/10.1002/int.22371

[13] Y. Chen *et al.*, "A Blockchain-Based Medical Data Sharing Mechanism with Attribute-Based

Access Control and Privacy Protection", *Wireless Communicaitons and Mobile Computing*, vol. 2021, pp. 1–12, 2021. https://doi.org/10.1155/2021/6685762

[14] T. Feng *et al.*, "Blockchain Data Privacy Protection and Sharing Scheme Based on Zero-Knowledge Proof", *Wireless Communicaitons and Mobile Computing*, vol. 2022, pp. 1–11, 2022. https://doi.org/10.1155/2022/1040662

[15] H. Wang *et al.*, "A MADM Location Privacy Protection Method Based on Blockchain", *IEEE Access*, vol. 9, pp. 27802–27812, 2021. https://doi.org/10.1109/ACCESS.2021.3058446

[16] Y. Baseri *et al.*, "Privacy Preserving Fine-Grained Location-Based Access Control for Mobile Cloud", *Computer & Security*, vol. 73, pp. 249–265, 2018. https://doi.org/10.1016/j.cose.2017.10.014

[17] S. J. De and S. Ruj, "Efficient Decentralized Attribute Based Access Control for Mobile Clouds," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 124–137, 2020. https://doi.org/10.1109/TCC.2017.2754255

[18] W. Li *et al.*, "Flexible CP-ABE Based Access Control on Encrypted Data for Mobile Users in Hybrid Cloud System", Journal of Computer Science and Technology, no. 32, pp. 974–990, 2017. https://doi.org/10.1007/s11390-017-1776-1

[19] A. Koe *et al.*, "Fine-Grained Access Control System Based on Fully Outsourced Attribute-Based Encryption", *Journal of Systems and Software*, vol. 125, pp. 344–353. https://doi.org/10.1016/j.jss.2016.12.018

[20] R. Zhang *et al.*, "Revocable Outsourcing Multi-Authority ABE for Medical Data in Mobile Cloud", in *Proc. of the 2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2021, pp. 338–345. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics53846.2021.00061

[21] Z. Ying *et al.*, " Reliable Policy Updating under Efficient Policy Hidden Fine-grained Access Control Framework for Cloud Data Sharing", *IEEE Transactions on Services Computing*, 2021. https://doi.org/10.1109/TSC.2021.3096177

[22] S. Saroiu *et al.*, "Policy-Carrying Data: A Privacy Abstraction for Attaching Terms of Service to Mobile Data", in *Proc. of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile'15)*, 2015, pp. 129–134. https://doi.org/10.1145/2699343.2699357

[23] S. Lee *et al.*, "Privacy Preserving Collaboration in Bring-Your-Own-Apps", in *Proc. of the 7th ACM Symposium on Cloud Computing (SoCC'16)*, 2016, pp. 265–278. https://doi.org/10.1145/2987550.2987587

[24] Y. Ryu, "A Buffer Management Scheme for Mobile Computers with Hybrid Main Memory and Flash Memory Storages", *International Journal of Multimedia and Ubiquitous Engineering*, vol. 7, no. 2, pp. 235–240, 2012.

[25] G. Hasslinger *et al.*, "Performance Evaluation for New Web Caching Strategies Combining LRU with Score Based Object Selection", *Computer Networks*, vol. 125, pp. 172–186, 2017. https://doi.org/10.1016/j.comnet.2017.04.044

[26] C. Li *et al.*, "QoE-Driven Mobile Edge Caching Placement for Adaptive Video Streaming", *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2018. https://doi.org/10.1109/TMM.2017.2757761

[27] E. Zeydan *et al.*, "Big Data Caching for Networking: Moving from Cloud to Edge", *IEEE Communications Magazine*, vol. 54, no. 9, pp. 36–42, 2016. https://doi.org/10.1109/MCOM.2016.7565185

[28] A. Rocha *et al.*, "DSCA: A Data Stream Caching Algorithm", in *Proc. of the 1st Workshop on Content Caching and Delivery in Wireless Networks (CCDWN'16)*, 2016, pp. 1–6. https://doi.org/10.1145/2836183.2836191

[29] X. Li *et al.*, "GreenTube: Power Optimization for Mobile Videostreaming via Dynamic Cache Management", in *Proc. of the 20th ACM International Conference on Multimedia (MM'12)*, 2012, pp. 279–288. https://doi.org/10.1145/2393347.2393390

[30] M. Naeem *et al.*, "Hybrid Cache Management in IoT-Based Named Data Networking", *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7140–7150, 2022. https://doi.org/10.1109/JIOT.2021.3075317

[31] T. Peng *et al.*, "Value-Aware Cache Replacement in Edge Networks for Internet of Things", *Transactions of Emerging Telecommunications Technologies*, vol. 32, no. 9, 2021. https://doi.org/10.1002/ett.4261

[32] J. Hou *et al.*, "A GNN-based Approach to Optimize Cache Hit Ratio in NDN Networks", in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6. https://doi.org/10.1109/GLOBECOM46510.2021.9685872

[33] P. Denning, "The Locality Principle", *Communication Networks and Computer Systems*, 2006, pp. 43–67. https://doi.org/10.1109/GLOBECOM46510.2021.9685872

Received: August 2021
Revised: July 2022
Accepted: August 2022

Contact addresses:
Wenyun Dai
Fairleigh Dickinson University
Teaneck
New Jersey
USA
e-mail: scorpiodwy@fdu.edu

Longbin Chen
Alation Inc
Redwood City
California
USA
e-mail: longbin.chen@alation.com

Wenyun Dai is an assistant professor of computer science at Gildart Haase School of Computer Sciences and Engineering, Fairleigh Dickinson University, Metropolitan Campus. He received his PhD in Computer Science from the Pace University. He received the master's degree from the Shanghai Jiao Tong University and the bachelor's degree from the Xiamen University. His research interests include distributed systems, cloud computing, mobile computing, privacy protection, and optimization.

Longbin Chen is currently a software engineer with Alation, Inc., Redwood City, CA, USA. He was previously working at IBM Hybrid Cloud Platform. He received his PhD degree from the Pace University, New York, NY, USA, and the MSc degree from San Jose State University, San Jose, CA, USA.