

A High Performance AI-Powered Cache Mechanism for IoT Devices

Wenyun Dai
Computer Science
Fairleigh Dickinson University
Teaneck, NJ, USA
scorpiodwy@fdu.edu

Muge Zhang
Computer Science
Fairleigh Dickinson University
Vancouver, CA
m.zhang1@student.fdu.edu

Abstract—Various Internet of Things (IoT) devices bring great convenience to users. However, the privacy protection and overall performance still need improvements due to their limited storage and computation abilities. Cloud computing can be used to solve or lessen these issues by providing server-level security, unlimited storage, and high computation abilities. It is time-consuming to conduct all the calculation in local IoT devices. It is also not wise to offload everything to Cloud, because the data could be massive that takes longer time to transfer than to compute. In this paper, we design a framework in which the local IoT devices are the cache and the remote Cloud servers are the memory. To further improve system efficiency and adaptability, we use some lightweight Artificial Intelligence (AI) methods to predict users' behavior patterns, that can adapt to changes while maintaining optimal performance.

Index Terms—Cache mechanism, IoT device, Edge Computing, Artificial Intelligence

I. INTRODUCTION

Internet of Things (IoT) devices are increasingly popular nowadays. Smartphones can offer equal or even better services to users compared to traditional personal computers. Smart wearable devices, such as watches and rings, can monitor users' behaviors and health helping them build better lifestyle. Smart outlet and cameras bring great value of convenience and better security protection. Users can do payments using their phones, watches, and rings by tapping a POS machine or face recognition. To achieve the best convenience, a lot of data are stored in the devices. However, IoT devices has limited storage capacity due to they limited size.

Fortunately, cloud service works smoothly with IoT devices. Cloud storage can provide fine-grained access control and massive storage [1]. We have proposed a cloud-based framework for mobile devices that could release users from managing their data and privacy [2]. However, it is not wise to offload every task from devices to the cloud, especially some data intensive tasks that cost more on communication than calculation. There are some products focusing on using Cloud service for the users who do not have powerful devices and want to enjoy service anytime anywhere, such as Apple iCloud, Google Drive and Stadia, Nvidia GeForce Now, Sony PlayStation Now, Microsoft xCloud and OneDrive, etc. However, it is extremely hard to achieve non-latency operations due to the Internet speed.

Data compression techniques are not suitable for this scenario. They would bring more computation burden to local devices. It is also not practical to force all platforms to follow the same compression/decompression method for all kinds of data. Caching mechanism works better considering since it is platform-free and has better practical meaning for data access requests [3]. However, there are still several unsolved problems. First, IoT devices are not able to refresh their storage as the traditional volatile memory or cache. In other words, all the recent data are stored in the device regardless of whether they will be used in recent future. This contradicts the principle of locality obviously. In our design, we analogize IoT devices to the cache or main memory and the cloud storage works to the secondary memory. IoT devices only store popular and high-priority data, while all others are stored in the cloud storage. Then the data stored in IoT device can be erased or replaced based on the principle of locality. This introduces the second problem about the cache mechanism.

The cache mechanism consists of several parts including placement, replacement, and cleaning. It is common that the replacement policies should be based on the priority. In our design, the placement and cleaning policies are also based on the priority. In traditional systems, the priority values are applied to each piece of data. However, we only assign priority values to different types of data, not any single piece of data. It is also too complicated and time-consuming to manage the priority based on data for IoT devices with limited resource. As a result, we believe that it is better to pair priority values with data types, such as image, text, and video.

Then the priority is highly related to the user's behavior pattern. For example, a user works on daily basis with an assistant IoT device. The refresh frequency of the IoT device should be one day. In order to understand and learn user's behavior pattern, we apply some Artificial Intelligence (AI) methods. Thanks to recent significant development of IT industry and manufacturing, AI techniques are used in various places with relative low threshold. There are some AI algorithms that can provide very precise predication but also cost a lot of time. Meanwhile, the algorithm should be flexible because users' behavior patterns could change. As a result, we apply the Multi-Armed Bandits (MAB) and Linear Regression in our work to calculate and predict users' behaviors.

II. RELATED WORK

A. Privacy Protection and Access Control

There are quite a few research works about privacy protection and access control. Some of them apply the Blockchain to power the security [4]–[6], but we do not introduce the Blockchain considering the conflict between the decentralization of the Blockchain and the centralized management idea of our approach. Some research works use encryption techniques to ensure the security and some others take advantage of cloud system. N. Shamsuddin and S. Pitchay proposed a solution to use location-based cryptography to secure data in Cloud storage [7]. Their approach used Advanced Encryption Standard (AES) as the encryption and decryption and Dynamic Tolerant Distance (DTD) as the protocol. The geo-lock key and secret keyword are generated by Secure Hash Algorithm 2 (SHA 2) that were stored together in the Cloud storage. We share the similar idea of storing credential and important data in the Cloud storage since it can provide better access control, but our approach is not based on location information or cryptography. K. Routray and P. Bera proposed a secure and context-aware encryption technique for remote access control of IoT devices [8]. They combined the Ciphertext-Policy Attributed-based Encryption (CP-ABE) and dynamic parameters in IoT devices. They used fog-assisted system to verify the dynamic parameters for real-time requirements and efficiency. We do not use any encryption methods considering the extra overheads and the complexity of IoT devices.

Q. Xu et al. proposed a privacy-preserving data access control scheme based on Ciphertext-Policy Attribute-based Signcryption to provide fine-grained access control in multi-authority cloud storage system [9]. The cloud system was responsible of securing the attributes of signcryptor and de-signcryptor. They designed to offload the massive pairing operations to the cloud in order to lower the overheads. H. Tian et al. addressed the drawbacks of traditional CP-ABE schemes and proposed a lightweight attribute-based access control with full privacy protection [10]. They used a lightweight two-party secure computing protocol between the users and the authority to generate secret keys. They used a policy hidden strategy during the access control policy setting. They used a hybrid authentication method without submitting attribute values to the Cloud. To lower the overheads, they applied online/offline encryption and outsourced decryption. In our work, we do not offload all calculations to the cloud, especially the ones can be processed efficiently in IoT devices. Meanwhile, our approach is based on the principle of locality that can further reduce the overheads.

B. IoT Cache Mechanism with AI

There are some research about applying AI techniques into cache policy to make wise decision [11], [12]. S. Maher et al. proposed a deep learning-based Device-to-Device approach to utilize Gated Recurrent Unit (GRU) model to predict the popularity of content in network and adjust the cache eviction policy [13]. Their experiments showed that GRU outperformed

traditional methods like LRU and FIFO. This work shared our concerns about cache hit rate and AI for content priority prediction but emphasized very little on the privacy aspect of the data handled. Furthermore, the content of data they focused on was quite different from the IoT data of our work, which has a significant social factor involved. T. Zong et al. employed an ensemble of constituent caching policies and selected the best-performing policy to control the cache management [14]. Their algorithm enhanced traditional cache policies with deep recurrent neural network based on time-series analysis. They developed a deep reinforcement learning agent that adaptively combined base caching policies according to their virtual hit ratios on parallel virtual caches. In their experiments, the overhead was 4.12 ms on average to select the best policy. However, we believe the extra cost depends on the size of data and complexity of requests.

X. Wang et al. designed a federated deep-reinforcement-learning-based cooperative edge caching (FADE) framework [15]. It enabled base stations to learn a shared predictive model by considering the first-round training parameters as the initial input of the local training. Then the near-optimal local parameters would be uploaded to base stations for the next round of global training. C. Zheng et al. proposed an unsupervised and privacy-preserving popularity prediction framework for Mobile-Edge Computing enabled Industrial IoT devices [16]. They introduced the concepts of local and global popularities and modeled the time-varying popularity of each user as a model-free Markov chain. They developed an unsupervised recurrent federated learning (URFL) algorithm to predict the distributed popularity while achieving privacy preservation and unsupervised training. C. Li et al. proposed a Predictive Edge Caching (PEC) system to predict the future content popularity using fine-grained learning models [17]. Their method mined sequential patterns in user content retrieval behaviors and prefetched popular contents using idle network bandwidth. Compared to the above research works, we only use simple and lightweight AI methods to guarantee quick response time and we model the priority/popularity based on data type for each user.

III. DESIGN AND IMPLEMENTATION

Our system consists of two main components, IoT devices and Cloud, as shown in Figure 1. Majority of data are stored and managed in the Cloud with fine-grained access control. The IoT device with enough storage and calculation abilities can store data and finish task locally. Meanwhile, IoT devices can offload heavy and complicated computation task to their corresponding Cloud. The cache mechanism is applied towards local IoT devices and the Cloud for better performance.

A. Access Control with Privacy Protection

We categorize data into three privacy levels, in which one indicates no privacy concerns and three means highly sensitive and private. We do not change the storage format or strategy in IoT devices, because it would be too tedious and impossible to cooperate among all manufacturers. We store data in the cloud

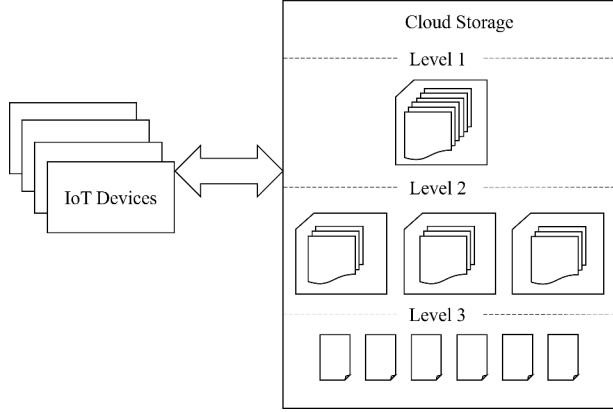


Fig. 1. System Framework with Access Control

storage into different folder structures based on the privacy level, shown as the right part in Figure 1.

All data with the lowest privacy level are stored in the same folders. Any authenticated request can access all data in the folder. All requests not in the block list are considered authenticated. Data with the second privacy level are stored in different folders based on the type and content. Only requests with two or three privacy level can access all data in the particular folder labeled the second privacy level. Highly private data are stored in a highly secure folder, and each request can only obtain one piece of data at a time.

The most important question is how to assign privacy levels to data and who could take responsibility for this task. One consensus is that only users understand each piece of data is private or not. However, not all users could fully understand the privacy. For example, a common post on twitter with location tag seems innocent, but it exposes the user's geolocation that could be private. There is true story that a college student tracked Elon Musk's private jet on Twitter. Meanwhile, it is too tedious for users to assign privacy level to each piece of data. As a result, we design and implement the privacy level assignment function and make it changeable by users.

Our main idea of privacy level assignment is based on data type. For ordinary users, we consider text data are highly private, such as transaction records, SSN numbers, bank accounts, card numbers, personal addresses, etc. Text data consists of plain text, images, emails, messages, and contacts. Then images and videos showing people faces are the second level of private. Finally, all other data are with privacy level 1. Meanwhile, we consider images and video have the highest privacy level in the IoT devices used for home security. Furthermore, users always can change the directories of their data if they want to change the privacy level of some data.

B. Cache Mechanism

There are a lot of intensive and time-consuming tasks that makes it nearly impossible for IoT devices to finish them quickly and efficiently. Cloud computing is a good alternative,

but there is extra communication cost between IoT devices and remote servers [18]. The main objective of our cache mechanism is to reduce communication cost by predicting users' behavior patterns.

The IoT devices are the cache storage in the system. Similar to the traditional cache system, we store "popular" data within IoT devices based on the principle of locality. Meanwhile, we only remain data with the lowest privacy level in IoT devices considering the privacy protection. Normally, highly private or sensitive data are less frequently used in Blockchain transactions, compared to transaction records in the form of Hash codes. In one word, our goal is to remain popular but not private data in IoT devices.

There are multiple types of data stored in IoT devices and different user have different behaviors and habits of using them. For example, one user working in the stock market frequently uses stocked related data in weekdays but only uses multimedia data for entertainment in weekends. It should be better to assign different priorities to different types of data dynamically than treating all data equally. The users' preferences towards data type are updated by users' usage. The highest priority data should have the largest storage space and the least possibility being replaced. In the initialization phase, we set all data type the same priority which is 0 and they equally share the storage space.

1) *Cache Update and Replacement*: Algorithm 1 shows the strategy of updating the *priority_heap*, that is users' preference towards different types of data. The hit counter is assigned to be 0 initially. If the current request asks for the same type of data with the previous one, the hit counter will be added by 1. If the current request does not match the previous one, the hit counter starts over from 0. After getting the hit counter, we modify the priority value of the data type being requested currently. First of all, the reward is the exponent of the hit counter. The reward is 1, if the hit counter is 0. Then the estimated reward is calculated based on Exponential Moving Average (EMA) algorithm [19]. We set the learning rate to be 0.1. The new priority value of the requesting data types equals to 90% of the original value adding 10% of the new *reward* value. Finally, we update the *priority_heap* for future usage.

After obtaining the priority, we need to do the placement in order to ensure the locality. In traditional systems, there are partitioning, paging, or segmentation techniques to shard data into small chunks. However, we do not consider this level of low-end technique and only assign enough storage space for the request. Algorithm 2 shows the changes towards cache sizes of all types of data when the cache is not full. The cache size heap and the data type heap are sorted by the opposite ordering of the priority heap. In other words, the priority heap always tops the highest priority data type, while the cache size heap and the data type heap prioritize the lowest priority data type. Each time a request is accepted, we assign more storage space to that type of data if it is not with the least priority data type. The increment of storage space is linear and the increased size equals to the data size just being requested.

Algorithm 1 Update *priority_heap* with MAB Algorithm

Require: Priorities of data, *priority_heap*; new request data, *nr_data*; last request datatype, *lr_type*; hit counter, *c*; learning rate, α .

Initialization : All priorities are zero. $\alpha = 0.1$.

```
while nr_data != null do
  if nr_data.type != lr_type then
    c = 0 {new type request}
  else
    c ++ {same type request}
  end if
  reward =  $2^c$ 
  reward_est[nr_data.type] =
  reward_est.get(nr_data.type, 0) * (1 -  $\alpha$ ) +  $\alpha$  *
  reward
  priority_heap.get(nr_data.type) =
  reward_est[nr_data.type]
  lr_type ← nr_data.type
end while
```

The newly storage space comes from the data type with the lowest priority. If there is not enough space for the lowest priority data type, it comes the second lowest priority data type. In one-word, high priority data type “rob” storage space from low priority data type each time it is referenced.

Algorithm 2 Update cache size for each type of data when the cache is not full

Require: Priorities of data, *priority_heap*; cache sizes, *cs_heap*; new request data, *nr_data*.

Initialization : Divide the whole space to each data type equally.

```
while nr_data != null do
  if cs_heap.findMin() != nr_data.type then
    cs_heap.get(nr_data.type) += nr_data.size
    while cs_heap.findMin() > nr_data.size do
      cs_heap.deleteMin()
    end while
    cs_heap.findMin() -= nr_data.size
  end if
  Store data in the device
end while
```

When the cache is full, we will run the replacement algorithm. The real replacement towards the data with the highest priority never happens if there is some data with less priority stored in the device. If the real replacement occurs, it means the device is full of the highest priority data. We will apply the iconic Least Frequently Recently Used (LFRU) strategy.

2) *Cache Refresh*: Different to the traditional cache system, IoT devices are always online that never refresh themselves. However, computations about transactions in the Blockchain are not always permanent. In other words, old data, such as that happened a week ago, may not be useful to validate the current transaction. We design a dynamic cache refresh strategy with

several variables.

The first variable, f , indicates the frequency of cache refresh. This variable should practical period, such as one day, one week, ect. Meanwhile, this variable is dynamically changing based on users’ behaviors. For example, ordinary users have similar weekly behavior patterns. As a result, we can set the variable f to one week. To confirm the data will not be refreshed too frequently or infrequently, we can add the boundary to variable f as one day to one month.

The second variable, p , is the percentage of users’ favorite data being refreshed. For example, text is the data with the highest priority and the value of p is 80%. There are 1000 text files and each time cache refresh occurs, 800 text files are erased. The erasing policy is based on the traditional Least Frequent Recently Used (LFRU) algorithm [20]. Meanwhile, all other types of data are removed. If the variable p is zero, all the highest priority data are remained in IoT devices. If the variable p is 100%, all the highest priority data are removed each time cache refreshes. We analyze three cases to determine the value of p based on the priority list we got from Algorithm 1. Assume that the priority list for each type of data is a , b , c , d , and e .

- Case 1: the highest priority data has significant higher value than others. For example, a is 23 while b , c , d , and e are smaller than 10. In this case, it is better to keep all the highest priority data in device. We set the value of p to zero.
- Case 2: all types of data share same or similar priority values. For example, a is 15, and b , c , d , and e are 14, 12, 11, and 10. The user has no obvious preference to any kind of data. We set the value of p to 100% in order to refresh everything.
- Case 3: the highest priority data has higher value than others, but not significantly enough. For example, a is 20, and b , c , d , and e are 15, 10, 6, and 4. The highest priority data takes 57% thus we set the value of p to 43%. The value of p could change for each refresh cycle.

The tag, s , indicates the special usage of some large volume data. There are some special data are not frequently accessed in single cycle but are always uses in almost all cycles. For example, a user always listens to the same list of songs when he/she is commuting. This user does not listen to music during work hours. The list of songs is only used during 8 AM to 9 AM and 5 PM to 6 PM. As a result, we can remove the list of songs from the devices out of its active time. Fortunately, the data itself and users’ pattern are not hard to predict. We apply Linear Regression [21] to predict the user’s behavior and mark the data with s if it meets the criteria.

Figure 2 shows the workflow of the cache refresh. The transaction handling process keeps running. When the refresh timing, controlled by f , occurs, the system first checks the special data marked with s . If it is still within their active period, they will be remained in the device. Otherwise, they will be erased. Then the system starts to deal with the data with the highest priority. If the percentage p has met, all the

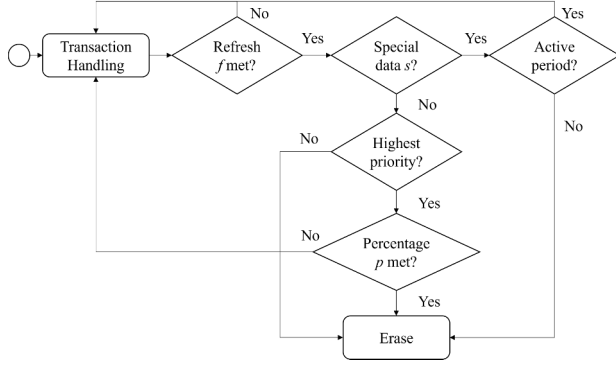


Fig. 2. Workflow of Cache Refresh

TABLE I
EXPERIMENTAL DATA

No.	Text	Image	Audio	Video	Others
A	60%	20%	5%	5%	10%
B	4%	20%	10%	60%	1%

following data will be erased. If the percentage p has not met, they data will be kept. Last, all other data are erased.

IV. EXPERIMENTS AND RESULTS

A. Experimental Environments

The experimental IoT devices in our experiments are one Google Pixel 4a and one Raspberry Pi 4. The Google Pixel 4a is with 128 GB storage and the Raspberry Pi is with 32 GB storage. The cloud storage in our experiment is a desktop computer with AMY Ryzen 5800X CPU, 32 GB RAM and 2 TB SSD storage. All the devices are connected within the same WiFi network. They communicate using RESTful APIs implemented by ourselves.

Meanwhile, we prepare two sets of testing data being stored in the devices and the server. The percentage of each data type are listed in Table I. Dataset A simulates the user focusing on text type of data using tablet device. Dataset B simulates a home security IoT device, such as a security camera, full of images and video. Furthermore, we assign different privacy levels to different datasets. In dataset A, text type of data have the highest privacy level, while all others are with lowest privacy level. In dataset B, images and video have the highest privacy level, while audio have the second privacy level. All others are not private.

B. Privacy Protection

First, we set the unit size of each type of data as:

- Text: 2 KB
- Image: 2 MB
- Audio: 3 MB
- Video: 100 MB
- Others: 1 KB

TABLE II
AMOUNT OF EACH TYPE OF DATA IN TWO DEVICES

Data Type	A		B	
	Pixel	Pi	Pixel	Pi
Text	38400000	9600000	2560000	640000
Image	12800	3200	12800	3200
Audio	2133	533	4267	1067
Video	64	16	768	192
Other	12800000	3200000	1280000	320000

Then we calculate the amounts of data for each type and list them in Table II. We only test the privacy protection improvement towards data with highest privacy level. Data with first and second privacy levels remain the same as the original system. If we do not use Cloud storage to protect the highly private data, each leaked request towards them will do 60%, 20%, and 60% harm to the whole data with dataset A, and B when the request is for text, image, and video respectively. For the Pixel with dataset A, text data have the highest priority, and they are stored individually in the remote server. Each request can only obtain one piece of data. Even if one piece of data is leaked, it only harms $4 \times 10^{-14}\%$ of the whole data. Each leaked request in Pi with dataset A only harms $6.5 \times 10^{-13}\%$ of the whole data. In dataset B, images and video have the highest privacy level. Each leaked request to image and video in Pixel only harms $1.56 \times 10^{-6}\%$ and $1 \times 10^{-4}\%$ of the whole data. Each leaked request to image and video in Pi only harms $1.95 \times 10^{-6}\%$ and 0.0016% of the whole data. In summary, the leakage of each request has near zero harm to the whole data.

C. Response Time

In our system, transactions involve local devices and the Cloud storage in different scenarios. The first one is that the computation along with all the associated data are in the local device. It is the same as the original system. The response time could be long if the computation is heavy. The second case is that the computation along with all the associated data are in the Cloud storage. It is the best case since cloud storage can provide way better computation ability and there is no extra communication cost about data. The third one involves the local device and cloud storage, which is the most frequent. If the computation is not tedious, it should be done within the local device. Otherwise, it is better to offload the task to the Cloud. However, offloading consumes extra cost due to the slow networking speed. We prepare the workload with majority heavy load tasks. We define 100 intervals for the whole workload and record the average response time within one interval to smooth the results. We fix the refresh frequency f to 20 intervals and p to 0.75. We do not consider s tag data in this section.

Figure 3 shows the results of our experiments running dataset A in Pixel and Pi. Pixel has better computation and storage abilities than Pi. The average response time in Pixel are $6.3ms$ and $5.9ms$. The average response time in Pi are $22.5ms$ and $17.9ms$. In both experiments, our approach has



Fig. 3. Experimental Results of Response Time running Dataset A

longer response time during the warm-up phase after cache refreshes, such as interval 0, 20, 40, and 80, but it catches up with and pass the original approach quickly. Overall, our approach shortens the response time by 7.15% and 19.9% in Pixel and Pi. Our approach works better in Pi than in Pixel, since Pi has smaller storage size which will offload more data and tasks to the server end.

Figure 4 shows the results of running dataset B in Pixel and Pi. Compared to the results of running dataset A, the response time of running dataset B are more stable. There are a lot of multimedia data in dataset B in which images and video are with the highest privacy level. In our approach, each data request can only obtain one image or video and they are stored in the Cloud storage. The cache strategy does not reduce response time, but slows it down due to inevitable extra communication cost. There is a spike of very high response time if the requested image or video is not found locally. However, the performance change trends to be gentle after the spike due to the locality. The average response time in Pixel are 0.55s and 0.56s. The average response time in Pi are 1.61s and 1.65s. Our approach degrades the performance by 2.9% and 2.6%. There are some research about reducing the size of multimedia data to decrease the communication cost, but we do not include any in this research. We plan to design and implement some strategies only for large size multimedia data in the future.

D. Cache Hit Ratio

There are two variable and one special tag used in our cache mechanism. The frequency of cache refresh, f , and the special tag, s , are highly user-dependent. The best timing to refresh the cache is when the user has done all the tasks and is about to start a new cycle. The best way to determine whether one piece of data should be tagged s is also based on the user's behavior. We set up experiments about different values of f and p . We also pre-marked some multimedia data as s . Furthermore, we simulate several different user behaviors.

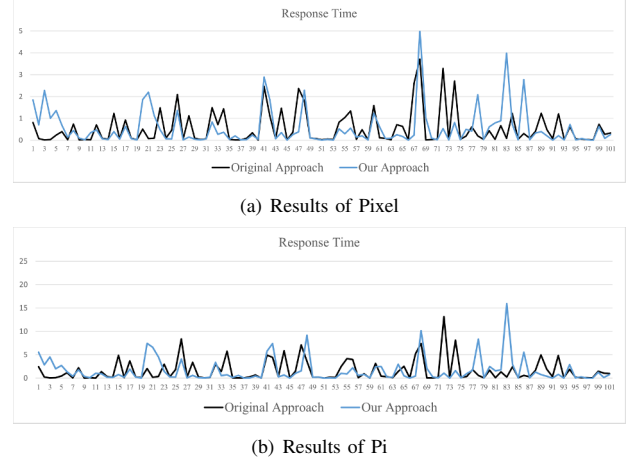


Fig. 4. Experimental Results of Response Time running Dataset B

1) *Cache Refresh Frequency f* : To only test the different results about f , we fix the value of p to be 0. All the favorite data remain in the cache while all other data are erased. The four different user behaviors can be predicted by the Linear Regression method. The values of f are 0, 0.1, 0.2, and 0.5. f is 0 means that the user does not follow any pattern. He/she uses the IoT device fully randomly without any locality. It is the original approach. 0.1, 0.2, and 0.5 mean that the users' behavior repeat in 10, 20, and 50 intervals in our experiments. These values also determine the cache refresh timing. The higher the value of f is, the more frequently the cache refreshes.

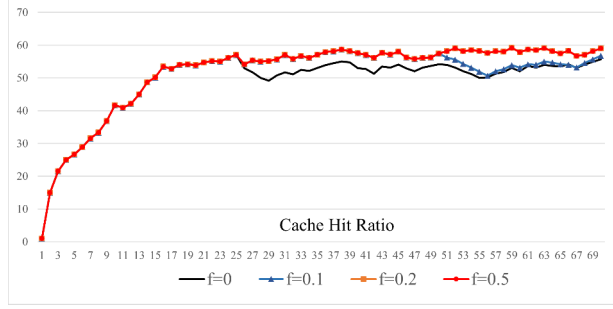
Figure 5 shows the results of cache hit ratio in Pixel and Pi running dataset A and B. We record 1 to 70 intervals, since all results of the remaining 71 to 100 are duplicated. During interval 1 to 70, $f = 0.1$ case refreshes 6 times. $f = 0.2$ case refreshes 3 times, and $f = 0.5$ case only refreshes once. The results from interval 1 to 10 are same because there was no cache refresh happened. The black line is the results of no refresh strategy. The red line is the results of refresh once at interval 50. These two lines are the same until interval 50. The blue line is $f=0.1$ case in which the cache refreshes the most frequently. The orange line is $f=0.5$ case. The cache hit ratio drops every time the cache refreshes, but it catches up quickly and even achieves better hit ratio afterwards.

The average cache hit ratio for each cases in Pixel and Pi running dataset A are:

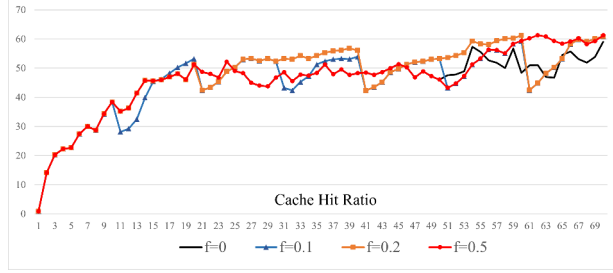
- $f=0$: 48.7% and 45.16%
- $f=0.1$: 50.43% and 45.79%
- $f=0.2$: 51.63% and 47.58%
- $f=0.5$: 51.61% and 46.34%

$f=0.2$ has the best performance, because it matches the workload the most. $f=0.1$ has better performance than $f=0$, but worse than larger values of f do. Meanwhile, the average cache hit ratio for each cases in Pixel and Pi running dataset B are:

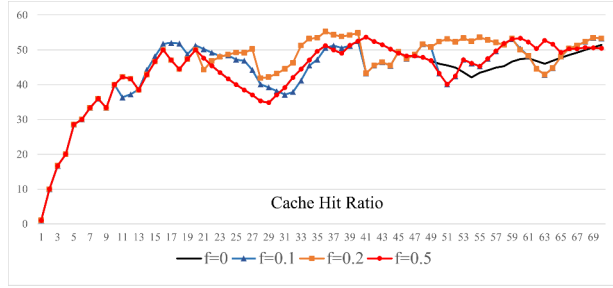
- $f=0$: 42.91% and 43.78%



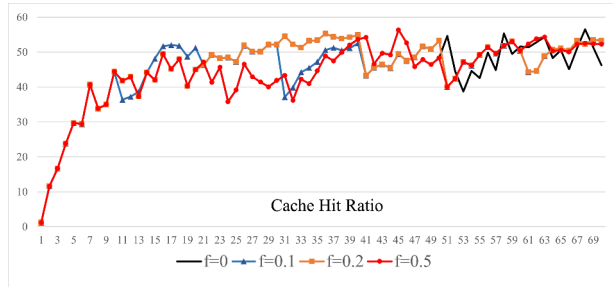
(a) Dataset A in Pixel



(b) Dataset A in Pi



(c) Dataset B in Pixel



(d) Dataset B in Pix

Fig. 5. Experimental Results of Cache Ratio Regarding variable f

- $f=0.1$: 43.7% and 45.11%
- $f=0.2$: 45.42% and 45.73%
- $f=0.5$: 43.61% and 44.04%

The results of dataset B are similar to those of dataset A, but $f = 0.1$ is slightly better than $f=0.5$.

The overall cache hit ratio of dataset A is better than that of dataset B. Pixel has better and more stable hit ratio than Pi. Pixel has larger storage size that delays the real cache replacement and lessens the possibility of thrashing. As we mentioned before, the refresh rate is user-dependent. With the popularization of AI chips, we believe that the refresh frequency can be easily calculated and managed.

2) *Favorite Data Refresh Percentage p* : In section IV-D1, we fix the value of p to be 1. In this section, we fix the value of f to be 0.2 which has the best performance to test different values of p . The range of p is 0 to 1. If $p = 0$, all the favorite data remain in the cache. It is the original approach. Referring to our design about the three cases in section III-B2, we test four different values of p , 0.2, 0.5, 0.75, and 1. The clean policy we use is based on LFRU algorithm. In other words, the least frequently recently used data are removed.

Figure 6 shows the results of cache hit ratio in Pixel and Pi running dataset A and B. The hit ratio drops dramatically each time the cache refresh when the $p = 1$, as the green line shows. The green line is the benchmark without partial erase and it has the worst performance. The black line shows the results of $p=0$ which is the same as the results in section IV-D1. It has the best overall performance. Setting $p=0$ means that all users' favorite data are remained. Users' behaviors normally follow the principle of locality, that makes the cache hit ratio the best. The blue line shows the results of $p = 0.2$. It takes around 5 intervals to catch up with the black line. Sometimes, $p = 0.2$ performs better than $p = 0$, such as interval 25 to 40 in Fig. 6(a) and 6(b). The orange line shows the results of $p = 0.5$. It takes about 10 intervals to catch up with the black line. The red line shows the results of $p = 0.75$. Once it catches up with other lines, it has similar performance with others. However, it drops a lot every time the cache refreshes, thus its overall performance is not competitive.

The average cache hit ratio for each case in Pixel and Pi running dataset A are:

- $p=0$: 48.7% and 45.16%
- $p=0.2$: 48.55% and 45.7%
- $p=0.5$: 45.91% and 40.76%
- $p=0.75$: 44.54% and 34.85%
- $p=1$: 38.81% and 32.35%

The average cache hit ratio for each case in Pixel and Pi running dataset B are:

- $p=0$: 42.91% and 43.78%
- $p=0.2$: 42.41% and 41.46%
- $p=0.5$: 40.77% and 38.53%
- $p=0.75$: 37.91% and 34.03%
- $p=1$: 33.97% and 31.43%

Our partial refresh strategy improves the average performance by 14.76% to 25.5% in Pixel and 7.72% to 41.26% in Pi for

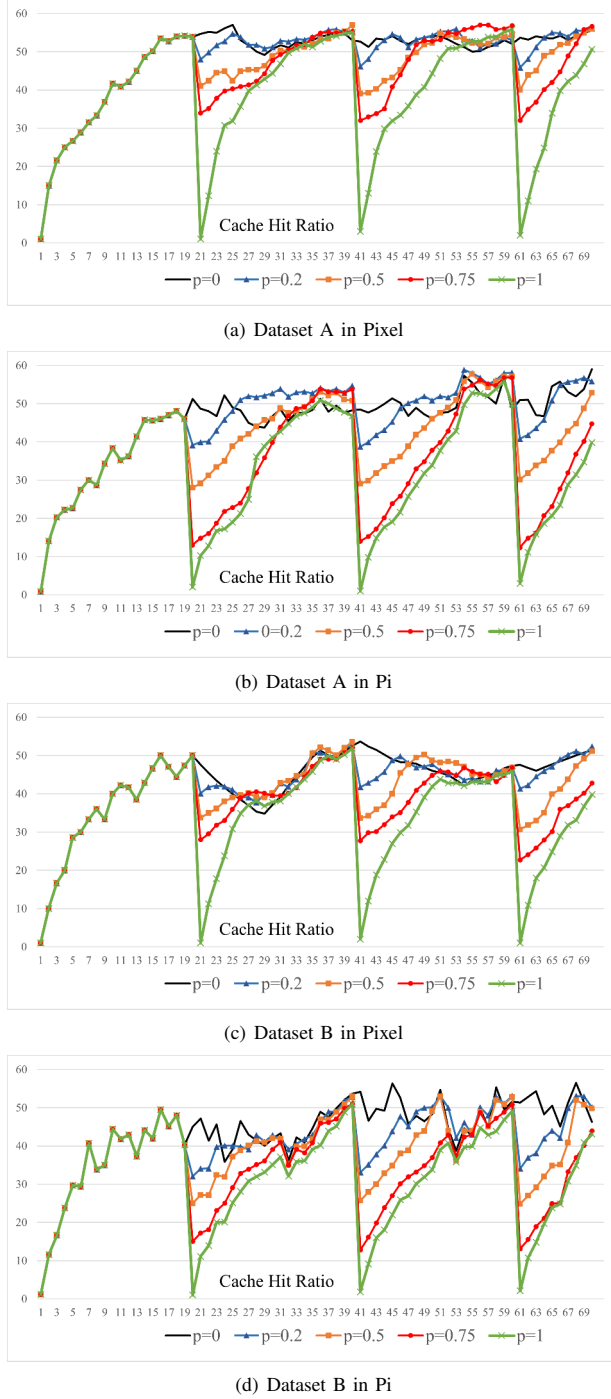


Fig. 6. Experimental Results of Cache Ratio Regarding variable p

dataset A. The numbers are 11.93% to 26.69% in Pixel and 8.25% to 39.27% in Pi for dataset B. We can see that Pixel has more stable cache hit ratio than Pi do if we compare the results horizontally. We believe that the reason is storage size. Pixel has larger storage size than Pi, thus it can store more data and has less probability of cache miss. Meanwhile, our partial refresh strategy has the best performance improvement in Pi running dataset B when $p=0$, which is 39.27%, over the refresh all approach. Our partial refresh strategy has the least improvement also in Pi running dataset when $p=0.75$. The dataset B is full of large size multimedia data and the Pi has small storage size. As a result, the amount of data are small. If $p=0$, nearly all user-favorite data are kept in device and it is less possible to miss. If $p=0.7$, majority of user-favorite are erase when the cache refreshes. The percentage of data remained in the devices over all data is low. It is higher possible to miss.

Im sum, the partial refresh or even non refresh towards users' favorite data can improve the average cache hit ratio by 7.72% to 41.26%. Due to the difference between IoT device storage and traditional cache/memory storage, it is practical to remain some data in devices. The experiment results match the principle of locality.

V. CONCLUSION

With the rapid development of technology and users' requirements, it is a trend that IoT devices work with Cloud computing for convenience and cost-efficiency. However, the privacy protection and performance are not flawless. We design an approach using Cloud computing to provide fine-grained access control of massive storage and powerful computation ability. To boost the performance, we design a cache mechanism to reduce the extra communication cost between the Cloud and IoT devices. We use some AI algorithms to predict users' behaviors and implement new designs about the privacy, priority, and replacement algorithms. Our approach cannot stop users' data being leaked, but can minimize the leakage harm to near zero. Meanwhile, experiments shows that our approach can the performance by 7.15% to 19.9%. In the future, we plan to implement more AI algorithms to achieve even fewer overheads and better performance.

REFERENCES

- [1] W. Dai, L. Chen, M. Qiu, A. Wu, and M. Liu, "DASS: A web-based fine-grained data access system for smartphones," in *2017 IEEE International Conference on Smart Cloud (SmartCloud)*. IEEE, 2017, pp. 238–243.
- [2] W. Dai, L. Qiu, A. Wu, and M. Qiu, "Cloud infrastructure resource allocation for big data applications," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 313–324, 2016.
- [3] W. Dai and L. Chen, "A cloud-based mobile privacy protection system with efficient cache mechanism," *CIT Journal of Computing and Information Technology*, vol. 29, no. 4, pp. 219–234, 2021.
- [4] J. Sengupta, S. Ruj, and S. Das Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot," *Journal of Network and Computer Applications*, vol. 149, p. 102481, 2020.
- [5] C. Yang, L. Tan, N. Shi, B. Xu, Y. Cao, and K. Yu, "Authprivacychain: A blockchain-based access control framework with privacy protection in cloud," *IEEE Access*, vol. 8, pp. 70 604–70 615, 2020.
- [6] N. Wu, L. Xu, and L. Zhu, "A blockchain based access control scheme with hidden policy and attribute," *Future Generation Computer Systems*, vol. 141, pp. 186–196, 2023.

- [7] N. S. M. Shamsuddin and S. A. Pitchay, "Implementing location-based cryptography on mobile application design to secure data in cloud storage," *Journal of Physics: Conference Series*, vol. 1551, no. 1, p. 012008, may 2020.
- [8] K. Routray and P. Bera, "Fog-assisted dynamic iot device access management using attribute-based encryption," in *Proceedings of the 25th International Conference on Distributed Computing and Networking*, ser. ICDCN '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 346–352.
- [9] Q. Xu, C. Tan, Z. Fan, W. Zhu, Y. Xiao, and F. Cheng, "Secure multi-authority data access control scheme in cloud storage system based on attribute-based signcryption," *IEEE Access*, vol. 6, pp. 34 051–34 074, 2018.
- [10] H. Tian, X. Li, H. Quan, C.-C. Chang, and T. Baker, "A lightweight attribute-based access control scheme for intelligent transportation system with full privacy protection," *IEEE Sensors Journal*, vol. 21, no. 14, pp. 15 793–15 806, 2021.
- [11] C. Zhong, M. C. Gursay, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, 2018, pp. 1–6.
- [12] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed deep learning at the edge: A novel proactive and cooperative caching framework for mobile edge networks," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1220–1223, 2019.
- [13] S. M. Maher, G. A. Ebrahim, S. Hosny, and M. M. Salah, "A cache-enabled device-to-device approach based on deep learning," *IEEE Access*, vol. 11, pp. 76 953–76 963, 2023.
- [14] T. Zong, C. Li, Y. Lei, G. Li, H. Cao, and Y. Liu, "Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 208–219, 2023.
- [15] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, 2020.
- [16] C. Zheng, S. Liu, Y. Huang, W. Zhang, and L. Yang, "Unsupervised recurrent federated learning for edge popularity prediction in privacy-preserving mobile-edge computing networks," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 24 328–24 345, 2022.
- [17] C. Li, X. Wang, T. Zong, H. Cao, and Y. Liu, "Predictive edge caching through deep mining of sequential patterns in user content retrievals," *Computer Networks*, vol. 233, p. 109866, 2023.
- [18] W. Dai, M. Qiu, L. Qiu, L. Chen, and A. Wu, "Who moved my data? privacy protection in smartphones," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 20–25, 2017.
- [19] F. Klinker, "Exponential moving average versus moving exponential average," *Mathematische Semesterberichte*, vol. 58, pp. 97–107, 2011.
- [20] Z. Zhang, C.-H. Lung, X. Wei, M. Chen, S. Chatterjee, and Z. Zhang, "In-network caching for icn-based iot (icn-iot): A comprehensive survey," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 595–14 620, 2023.
- [21] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 275–294, 2012.