CREAT: Blockchain-Assisted Compression Algorithm of Federated Learning for Content Caching in Edge Computing

Laizhong Cui, *Senior Member, IEEE*, Xiaoxin Su, Zhongxing Ming, Ziteng Chen, Shu Yang, Yipeng Zhou, *Member, IEEE*, and Wei Xiao

***Abstract*—Edge computing architectures can help us quickly process the data collected by Internet of Things (IoT) and caching files to edge nodes can speed up the response speed of IoT devices requesting files. Blockchain architectures can help us ensure the security of data transmitted by IoT. Therefore, we have proposed a system that combines IoT devices, edge nodes, remote cloud, and blockchain. In the system, we designed a new algorithm in which blockchain-assisted compressed algorithm of federated learning is applied for content caching, called CREAT to predict cached files. In the CREAT algorithm, each edge node uses local data to train a model and then uses the model to learn the features of users and files, so as to predict popular files to improve the cache hit rate. In order to ensure the security of edge nodes' data, we use federated learning (FL) to enable multiple edge nodes to cooperate in training without sharing data. In addition, for the purpose of reducing communication load in FL, we will compress gradients uploaded by edge nodes to reduce the time required for communication. What is more, in order to ensure the security of the data transmitted in the CREAT algorithm, we have incorporated blockchain technology in the algorithm. We design four smart contracts for decentralized entities to record and verify the transactions to ensure the security of data. We used MovieLens data sets for experiments and we can see that CREAT greatly improves the cache hit rate and reduces the time required to upload data.**

***Index Terms*—Blockchain, edge computing, federated learning (FL), gradients compression.**

Manuscript received 1 June 2020; revised 13 July 2020; accepted 29 July 2020. Date of publication 5 August 2020; date of current version 8 August 2022. This work was supported in part by the National Key Research and Development Plan of China under Grant 2018YFB1800302 and Grant 2018YFB1800805; in part by the National Natural Science Foundation of China under Grant 61772345 and Grant 61902258; in part by the Major Fundamental Research Project in the Science and Technology Plan of Shenzhen under Grant JCYJ20190808142207420 and Grant GJHZ20190822095416463; in part by the Graph-Based Network Optimization Algorithm Project of Huawei under Grant YBN2019125156; and in part by the Pearl River Young Scholars Funding of Shenzhen University. *(Corresponding author: Zhongxing Ming.)*

Laizhong Cui, Xiaoxin Su, Zhongxing Ming, Ziteng Chen, and Shu Yang are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: cuilz@szu.edu.cn; suxiaoxin2016@163.com; zming@szu.edu.cn; 1810272054@email.szu.edu.cn; yang.shu@szu.edu.cn).

Yipeng Zhou is with the Department of Computing, Faculty of Science and Engineering, Macquarie University, Sydney, NSW 2109, Australia (e-mail: ypzhou@mq.edu.au).

Wei Xiao is with the Research Institute, Tsinghua University in Shenzhen, Shenzhen 518063, China (e-mail: xiaow@tsinghua-sz.org).

Digital Object Identifier 10.1109/JIOT.2020.3014370

I. Introduction

THE FUTURE Internet is rapidly developing toward the Internet of Things (IoT), which will potentially connect 50 billion IoT devices by 2020 according to the prediction of Cisco Internet Business Solutions Group [1]. IoT devices now permeate in our daily lives, providing important tools for measurement and collection to inform our every decision.

The massive number of IoT devices will lead to a rapid explosion of the scale of collected data, therefore it is difficult for IoT to process and analyze large amounts of data. Compared with cloud computing, edge computing migrates the computation or storage of data to the edge of the network close to the end devices to alleviate the pressure of data transmission [2], [3]. In edge computing, files caching is considered to be a promising method. As shown in Fig. 1, IoT devices requesting files directly from close edge nodes can reduce the propagation delay of the network. Therefore, in order to quickly obtain the requested files, we need to predict the files that IoT devices may request in the future and cache these files in the edge nodes to improve the cache hit rate. Due to the storage limitations of edge nodes, it is important to estimate the future popularity of files. However, traditional caching algorithms do not consider the popularity of files in the future [4], of which leads to the low cache hit rate.

Due to the poor security of communications in current IoT architectures, there are problems of data reliability, such as data loss, insertion of malicious data, and so on. We can use the blockchain technology to prevent these problems [5]. However, the current blockchain is limited in its ability to scale [6]. On the other hand, due to the interaction of heterogeneous edge nodes and data transmission across nodes [7], the security and privacy issues of edge computing urgently need to be resolved. Therefore, integrating blockchain and edge computing into a system can solve the above problems well.

Based on the above description, we propose a system, which combines IoT devices, edge nodes, remote cloud, and blockchain. In the system, we designed a new algorithm in which blockchain-assisted compressed algorithm of federated learning is applied for content caching, called CREAT to improve the cache hit rate in edge computing.

Due to the decentralized architecture, traditional edge computing platforms for the IoT world suffer distributed attacks. For example, the distributed IoT devices can conduct the

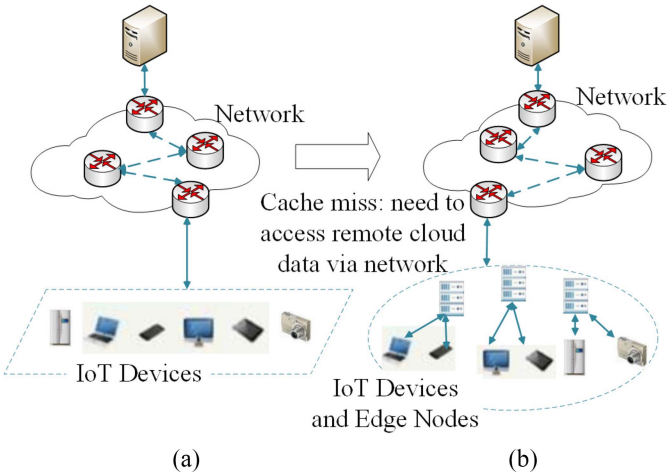


Fig. 1. Edge caching. (a) Cloud computing. (b) Edge computing.

Distributed-Denial-of-Service attack (or DDoS attack) or the flooding attack, which will lead to traffic congestion in the edge network [8]. Therefore, in the designed system, we will use blockchain to securely transmit data.

The CREAT algorithm uses federated learning (FL) [9], therefore edge nodes download models from remote cloud and use local data for model training without uploading local data. Communication problem is a bottleneck restricting the development of FL due to the poor network conditions of clients uploading gradients. In order to overcome this bottleneck, edge nodes will compress uploaded gradients in the CREAT algorithm which can significantly reduce the communication load. [10] shows that in each round of interaction in FL, because only a small fraction of gradients is far away from 0, we can selectively transmit gradients. We first use the K-means algorithm to detect important gradients of which we need to upload them precisely. After that, we use a clustering-based quantization algorithm to quantify these gradients to reduce the amount of data uploaded. Then, the rest gradients are approximated with their average value. When the edge nodes upload compressed gradients, some of them may upload forged gradients, which leads to the failure of model training. To avoid this, we will use blockchain to verify the uploaded data.

As mentioned above, blockchain provides an opportunity to solve security issues in edge computing [11]. Transactions in networks are recorded in the distributed ledger, which are immutable and traceable [12]. The distributed participants can verify the transactions to guarantee the validity of the transactions. Therefore, we will integrate the blockchain technology with the system. The data submitted by IoT devices and the gradients generated by edge nodes will be appended to the blockchain. Meanwhile, we also develop a consortium-based blockchain verification mechanism to verify the transaction correctness of the system, which reaches the balance of security and verifying efficiency.

The remainder content is organized as follows. We review the related studies in Section II. The designed system is introduced in Section III. We introduce the CREAT algorithm in Section IV. The experiment results are discussed in Section V before we conclude our article in Section VI.

II. Related Work

A. Edge Computing

IoT platforms can use cognitive radio to make full use of the spectrum to continuously connect smart vehicles and transmit data. However, vehicle communication modes and data QoS require effective data scheduling schemes. Zhang *et al.* [13] used a deep Q learning method to design the best data scheduling scheme to relieve the pressure of transmission. For IoT, the cloud-centric computing platform will cause high-latency data transmission and cannot meet real-time applications. Edge computing is considered promising to solve the problem of network delay. IoT devices can offload computing tasks to "closer" edge nodes, so computing latency is improved. Premsankar *et al.* [14] classified and investigates current edge computing architectures and platforms, and then describes the applications of IoT that benefit from edge computing.

By transferring computing tasks to resource-rich edge nodes, not only can the Quality of Service (QoS) and Quality of Experience (QoE) be greatly improved but also the capabilities of mobile devices can be enhanced to run applications with higher resource requirements. In recent years, a lot of research has been conducted on the design of computing offload strategies. Mach and Becvar [15] divided the researches of computing offload into three key areas: 1) decision of computing offload; 2) allocation of computing resources within edge computing; and 3) management of mobility. The article highlights lessons learned in the field of edge computing and discusses challenges to be solved by taking full advantage of potential provided by edge computing. Chen *et al.* [16] considered that multiple base stations can be selected for computational offloading in an ultradense sliced radio access network. Zhang *et al.* [17] considered various states of multiple edge nodes and various vehicular offloading modes in the mobile-edge computing architecture and uses deep Q-learning to perform effective task offloading and determine the target server and data transmission mode.

If files are fetched from a remote data center, the low latency of content access and requirements of different applications may not be satisfied. Fortunately, edge computing can be used as an effective framework to ease the pressure on the backhaul link by caching files to edge nodes. Mach and Becvar [15] proposed a cooperative edge caching solution, which is a new paradigm. In vehicle-edge computing and networks, with the help of flexible tripartite cooperation between macrocells, roadside units, and intelligent vehicles, the content placement and content delivery can be optimized. Recently, Wang *et al.* [18] have observed the emergence of promising mobile content caching and delivery technologies. Popular files are cached on the edge nodes, so users who request these files do not need to copy from the remote cloud, which greatly reduces redundant traffic.

Because modern wireless networks are content centric, the demand for multimedia services is growing rapidly, which poses serious challenges to the demand for large-scale content delivery on mobile networks. To overcome this challenge, edge caching has become a promising method, which can reduce the heavy burden of data transmission by caching and forwarding content at the edge of the network. Zhang *et al.* [19] proposed

a new type of collaborative edge caching architecture suitable for 5G networks, where mobile-edge computing resources are used to enhance edge caching capabilities. In this architecture, the authors focused on mobility-aware hierarchical caching, in which intelligent vehicles are used as cooperative caching agents for sharing content caching tasks with base stations. In view of the characteristics of mobile-edge caching and delivery technology that can reduce the pressure of the backhaul link, Wang *et al.* [20] conducted a comprehensive review of the key technologies of mobile-edge computing and caching and the authors reviewed the research progress of edge caching in content popularity, caching strategy, scheduling, mobility management, etc. As augmented reality applications become more and more popular, more and more data requirements require low latency. Mobile-edge computing is expected to be an effective solution to meet low latency requirements. Hao *et al.* [21] introduced a new concept of task caching. Task caching refers to caching completed task applications and related data in the edge cloud. Then, the authors study the joint optimization problem of edge cloud task caching and offloading under the constraints of computing and storage resources.

These characteristics and applications of edge computing demonstrate its prospects in IoT. In this article, we focus on edge caching strategies, which cache files based on their popularity to improve the cache hit rate.

B. Federated Learning

The centralized content caching algorithm requires edge nodes to upload local data to a server to train a model. However, this may lead to the disclosure of users' privacy. To avoid this problem, we can combine FL with caching strategies to train a model without uploading users' data.

FL enables multiple edge nodes to collaboratively train a machine learning model while keeping their training data private. The previous work [22] introduced the concept and characteristics of FL, and explained the applications of FL in related fields. In [23], a comprehensive survey of the situation of FL in mobile-edge networks was conducted, and the current challenges and corresponding measures of FL in edge networks were described in detail.

Communication issue is a key factor limiting the development of FL. Researchers have proposed various methods to mitigate communication bottleneck in FL. For example, in order to reduce the number of communication rounds, Yao *et al.* [24] proposed to use a two-stream model to increase the computation efficiency of each participating device.

Researchers also investigated various model compression algorithms to reduce communication loads. Model compression transforms the gradients of a model to be more compact through sparsification or quantization. To reduce communication loads, Konečnỳ *et al.* [25] proposed structured and sketched updates to reduce the size of original model updates sent by clients to the server.

Our work proposes a new model compression algorithm that can compress the amount of uploaded data and greatly reduce communication loads while slightly affecting the accuracy of the model. We apply this compression algorithm to the content caching algorithm combined with FL, and use experiments to verify the cache hit rate after compression.

C. Blockchain

Blockchain is an innovative distributed ledger technology that can solve the problem of lack of trust in applications. Lu *et al.* [26] proposed a unified blockchain as a service (uBaaS) platform. The platform can not only improve the efficiency of blockchain application development, but also support the design and deployment of blockchain-based applications. A single blockchain-based system needs to provide services for multiple tenants at the same time, so there are challenges in terms of data and performance isolation, and scalability. Weber *et al.* [27] proposed a scalable platform architecture which can ensure data integrity, data privacy, and performance isolation.

Since Bitcoin [28] and Ethereum [29], blockchain technology is considered promising to address security issues in edge computing. It provides consensus mechanisms (e.g., proof of work, proof of stake, etc.) for decentralized entities to achieve consensus, without introducing the centralized authorities. The blockchain participants share the distributed ledger, making the system transactions immutable, auditable, and verifiable. The integration of blockchain improves the security of traditional edge computing systems.

Nowadays, the blockchain technology has been adopted in numerous edge computing and edge intelligence applications. Zhang *et al.* [30] proposed a blockchain-based edge intelligence framework for the industrial IoT to improve the edge service cost and resist security issues. Lu *et al.* [31] also proposed a blockchain-based edge computing platform for FL. However, the existing blockchain-based edge computing platforms utilize inefficient consensus and verification mechanisms. For example, it may cost a long period to achieve the consensus of the system. Moreover, the decentralized participants may skip the transaction verification process to avoid the complicated computation [32]. Therefore, we will integrate the blockchain with our system to improve security, and propose a consortium-based verification scheme to enhance operational efficiency.

III. System Design

In this article, we propose a system, which is depicted in Fig. 2. The system combines IoT devices, edge nodes, remote cloud, and blockchain.

A. System Model

In the system, IoT devices will request files from edge nodes and uses these requests as data. When training a model, each IoT device sends collected data to the corresponding edge node as local data. However, in order to avoid traffic congestion problems caused by data transmission, IoT devices will transfer collected data to blockchain and edge nodes will obtain the data from blockchain.

We view edge nodes as wireless cache entities with reliable backhaul links to the Internet. Each cache entity has limited storage size, so we need to build a model to select files for

TABLE I
NOTATION

Notation	Meaning
F	number of files cached at the edge nodes
K	number of neighbour users used by hybrid filtering
t	number of global aggregations for federated learning
$\mathbf{W}_t$	global model of federated learning at iteration t
$\mathbf{W}_t^c$	local model of the $c-th$ client of federated learning at iteration t
$\mathbf{H}_t^c$	gradient to be uploaded by the $c-th$ client of federated learning at iteration t
$\mathcal{D}_i$	local data set of the $i-th$ client
$\mathcal{E}_t$	set of clients participating in training of federated learning at iteration t
α	global learning rate
N	dimension of the gradient vector
H_{tn}^c	$n-th$ element in the vector $\mathbf{H}_t^c$
C	number of clusters in the first clustering
u_{r_n}	centroid of the cluster r_n
r_n	cluster to which the gradient H_{tn}^c belongs
$\mathcal{R}_c$	set of gradients in the $c-th$ cluster
N_0	number of gradients in the second clustering
C_0	number of clusters in the second clustering

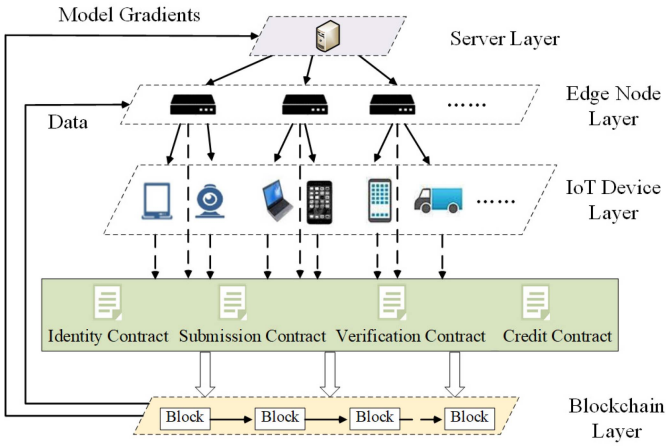


Fig. 2. Architecture of the system.

caching. We assume that each cache entity can store at most F files. When IoT devices are within the coverage of an edge node, they will request files from that node. If the requested files are cached in the edge node, the files will be directly transferred from the edge node to IoT devices.

In our system model, each edge node covers several IoT devices. After a period of time, edge nodes will connect to the server to update the cached files and model. In order to improve the cache hit rate and protect data privacy, the system is designed in conjunction with FL. In this system, each edge node independently computes updated gradients to the current global model by using local data. Moreover, each edge node can use its own trained model to obtain the features of users and files, thereby predicting the Fth most popular files. In order to ensure the credibility of data, the results calculated by edge nodes need to be appended to the blockchain through the verification of smart contracts, and then obtained from the blockchain by the remote cloud.

Although the distributed architecture of edge computing has many benefits, the security of its data transmission is a major issue [33]. The interaction of heterogeneous nodes and the transmission of data across edge nodes may produce malicious behavior. In edge computing systems, data is stored in different nodes, which makes it easier to lose data packets or store errors. Therefore, it is difficult to ensure the integrity of data. Using the blockchain technology, we can build distributed controls on multiple edge nodes. The blockchain protects the accuracy, consistency, and effectiveness of data by mining and copying a large number of nodes. We have integrated blockchain into edge computing to take advantage of the similarity between the P2P data storage mechanism in edge computing and the effectiveness of blockchain's P2P decentralized data, as well as their complementarity in terms of storage capacity and security provision.

The blockchain serves as a distributed ledger that records the transactions in terms of models and training parameters. The decentralized entities will submit their data to the blockchain, as well as conducting verification processes periodically, in order to ensure the security of the system.

The remote cloud maintains a global model, which is sent to connected edge nodes for training. The remote cloud will receive updated gradients uploaded by edge nodes and aggregate these gradients to update the global model. In each communication round, the cost of uploading gradients is regarded as communication cost, which dominates the FL. Because edge nodes have faster processors, the computation cost is relatively small. Therefore, we can reduce communication cost by using the newly proposed compression algorithm to compress the amount of uploaded data. Finally, according to each user's recommendation list, the server selects the Fth most popular files to be cached for edge nodes.

Because of the combination of the blockchain technology and FL framework in the designed system, we need to consider the control overhead brought about by these two aspects. For modern advanced neural network models, such as convolutional neural networks (CNNs), there are millions of parameters in the models. We suppose that there are 1 million parameters. If we use 4 B to represent each parameter, each client needs to upload 4 MB of data in a round of communication of FL. We will upload the data to the blockchain after verification. Because the amount of data is acceptable to the blockchain, the additional overhead in storage is not very large.

In the system we designed, we will compress the gradients uploaded by the edge nodes. Although it takes some time for the compression algorithm of FL to run and the data transmission of blockchain, edge caching is not a delay-sensitive application. Generally, edge nodes will update the model and cache files at a fixed period (e.g., every hour in the subsequent experimental phase), so the additional time delay overhead caused by the compression algorithm of FL and data transmission of blockchain is acceptable.

B. Blockchain Integration

To secure the system, we utilize the blockchain technology as the distributed ledger, where the transactions generated by decentralized entities will be recorded and verified. The data structure of a transaction consists of hash values of data

TABLE II
DATA STRUCTURE OF A TRANSACTION

Data Hash	Model Gradients Hash	IoT Device Identity	Edge Node Identity	IoT Device Signature	Edge Node Signature	Time
0xc5b3f9b9...	0xf3bhd6fc...	ID00000001	EN00000001	0xgfg49156d..	0xffg89f0d...	2020-6-01 10:10
0xb72f1bcg...	0xc6f2840f...	ID00000003	EN00000005	0x4230f21f...	0x76d22f70...	2020-6-01 10:20
...	...	...	...	...	...	...

submitted by IoT users, and the hash value of the model gradients trained by edge nodes. Meanwhile, the identities and signatures of the corresponding IoT devices and edge nodes involved in the FL process are included in a transaction. Finally, the timestamp of a transaction is included as well. The data structure of a transaction in the system is shown in Table II.

We design four smart contracts for decentralized entities to record and verify the transactions: 1) identity contract that is responsible for the management of each entity, including the distributed IoT users and edge nodes; 2) submission contract that provides the interface for edge nodes to submit their intermediate gradients to the blockchain; 3) verification contract that is responsible for the election of supervisory consortiums and transaction verification; and 4) credit contract that is responsible for the token rewarding and punishment of network participants.

For the Identity Contracts, Submission Contracts, and Credit Contracts, they are mainly responsible for recording data and interacting with the blockchain. The implementation of the process is relatively simple, so they will not be described here. The Verification Contract in the blockchain design is the key contract in our system to ensure the security of data transmission. The implementation of Verification Contract is as follows.

1) In each round, we generate several random numbers in a decentralized manner. In order to ensure security, we use the relevant algorithm in [34] to generate random numbers.
2) We will select a fixed number of nodes as the consortium members based on the generated random numbers.
3) Consortium members will vote to determine whether the transaction is valid.

We are trying to develop the smart contracts by using Solidity language and deploy them on the Ethereum. To ensure the security of the system, distributed nodes will utilize the smart contracts developed to maintain the distributed ledger and verify transactions. To enhance system security and throughput, we modify and apply the reputation-based PoS consensus mechanism, which will be discussed later.

The running process of the smart contracts is depicted in Fig. 3. When edge nodes compute the intermediate gradients, they will call Submission Contract to upload the data to the blockchain. The Submission Contract will first call the Identity Contract to validate the identities of edge nodes and IoT users, then upload the parameters to the distributed ledger as a transaction. The transactions will be verified randomly and periodically. To avoid the "verifier dilemma" [32] and improve the efficiency of the system, we refer to [11] and develop the consortium-based verification scheme. To be specific, the system will periodically call the Verification Contract

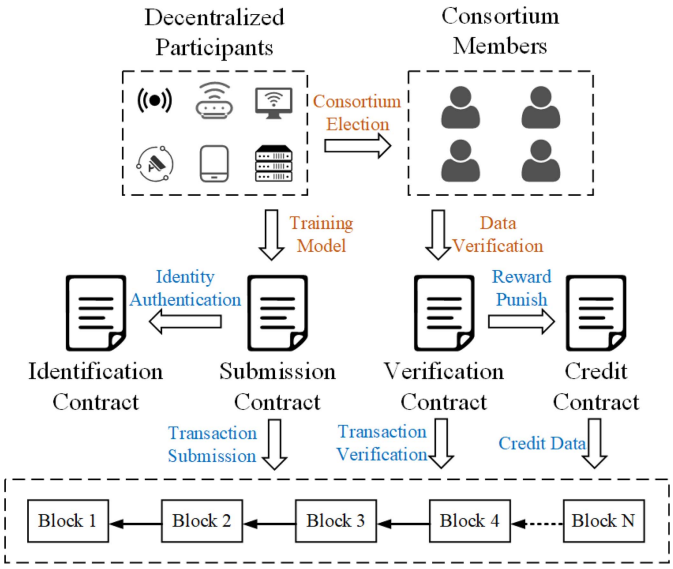


Fig. 3. Blockchain integration.

and elect the supervisory consortiums, which is responsible for the transaction verification process. The consortium members will vote for the transactions. If a transaction receives the majority of votes, it will be regarded valid. Otherwise, it will be considered invalid, and the corresponding entities will be punished with blockchain tokens, by calling Credit Contract. Cui *et al.* [11] proved that the consortium-based verification scheme is secure to resist misbehaviors, when the honest party has more computing power than the dishonest party. In this case, the security of the system can be guaranteed.

According to [12], we find that in a traditional blockchain system like Bitcoin, there are only seven transactions per second (TPS), so the network throughput is too low to meet the demand. In order to improve the throughput of blockchain in the CREAT algorithm, we used the Proof of Stake (PoS) consensus algorithm based on reputation mentioned in [11] and modified it, in which the decentralized parties with high reputation are more likely to be elected to maintain the network and append the transactions to the blockchain. In this case, the throughput of CREAT can be greatly improved by eliminating the time-consuming calculation required for traditional bitcoin mining. At the same time, the reputation-based PoS algorithm will encourage participants to behave honestly in the verification process to win the opportunity to propose blocks and obtain rewards, which is conducive to the honesty, security, and efficiency of CREAT.

IV. BLOCKCHAIN-ASSISTED COMPRESSION ALGORITHM OF FEDERATED LEARNING FOR CONTENT CACHING

In this section, we will introduce our proposed CREAT algorithm, which is divided into two parts. First, we will

introduce the FL-based proactive content caching algorithm (FPCC) [35]. Later, we will introduce the compression algorithm applied to the caching algorithm. Moreover, in order to ensure the credibility of the data uploaded in the algorithm, we will combine the blockchain to ensure the credibility during the training process.

A. Federated Learning-Based Proactive Content Caching Algorithm

For content caching, we need to use IoT devices' requests to learn caching decisions. The edge nodes should understand the popularity of files in order to cache the most popular files for IoT devices. The core problem is finding potential features of users and files, which are obtained from the history of users' requests, and use these features to compute the similarity between users and between files. We can use a neural network model called autoencoder to solve this problem. In order to facilitate the subsequent description, we define this kind of the FL-based cache algorithm without using the compression algorithm as FPCC [35].

In the above process, IoT devices need to upload historical requests as data to the corresponding edge nodes for training. The edge nodes need to send the intermediate results from the training process to the server for aggregation. During this process, malicious users may send incorrect information, resulting in incorrect training results. In order to avoid this situation, we need to combine the blockchain technology to verify the uploaded data, so as to ensure the security of uploaded data.

1) Autoencoder: The autoencoder is an unsupervised learning neural network model, which trains a single hidden layer neural network to reconstruct input data. The neural network can learn the potential features of data well. Given input data set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, there are m data samples in this data set, and the dimension of each data sample is n, that is, $x^{(i)} \in \mathbb{R}^n$. After the data sample $x^{(i)}$ is input to the neural network, an encoder h_1 is responsible for mapping the input $x^{(i)}$ to the potential features, and then a decoder h_2 remaps the potential features to reconstruct the input data $x^{(i)}$. Therefore, the model is essentially training a function $h_2(h_1(x^{(i)})) \approx x^{(i)}$. By training the autoencoder neural network, potential features of training data can be obtained, and then these potential features are used for hybrid filtering.

2) Hybrid Filtering: The similarity of users and the similarity of files are calculated based on the potential features of users and files extracted from two autoencoders. Based on the files' similarity of the requesting history of active users and their K neighbor users, a recommendation list of popular files for caching is generated. The whole process of edge nodes predicting popular files is as follows.

The autoencoder explores the potential features of users and files based on the history of users' request. Using the cosine similarity, the similarity between users and the similarity between files are calculated according to the obtained potential features. We assume that the current user is an active user. Based on the users' similarity matrix, we can determine K nearest neighbor users of the active user. The similarity of each file between the active user's request history and K nearest neighbor users' request history is obtained through the files' similarity matrix. Selecting the Fth most similar files generates a recommended list of popular files for caching.

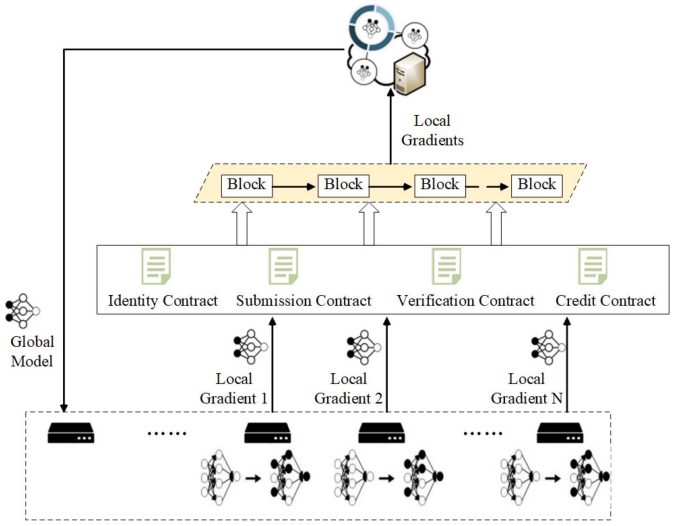


Fig. 4. FL training process.

The recommended list generated by edge nodes will be appended to the blockchain to ensure the security of data. Then, the server aggregates all the calculation results from the blockchain and selects the Fth most popular files as cached files of edge nodes.

3) Federated Learning: The edge nodes need to upload the gradients obtained from the training model to the remote cloud which will aggregate all the results of edge nodes. A major advantage of FL is that edge nodes use local data to train the model without uploading data to a central server. It can significantly reduce security and privacy risks. The FL training process is shown in Fig. 4. For the convenience of description, we will next introduce the training process of FL at iteration t. In order to train the global model, edge nodes need to download the global model parameter $\mathbf{W}_t$ first, and use local data to train the model $\mathbf{W}_t^1, \mathbf{W}_t^2, \mathbf{W}_t^3, \ldots, \mathbf{W}_t^c$, where $1, 2, 3, \ldots, c$ represent the indices of the edge nodes participating in the training, and t represents the communication round of training. The update that the edge node i needs to upload is defined as $\mathbf{H}_t^i = \mathbf{W}_t^i - \mathbf{W}_t$.

Finally, in order to prevent malicious edge nodes from uploading erroneous data, edge nodes do not directly send gradients to the server. The gradients of the edge nodes will be uploaded to the blockchain through the verification of four designed smart contracts, and the server will obtain the corresponding data from the blockchain.

The server receives the model updates uploaded by the edge nodes, and uses the federated average algorithm to aggregate and update the global model. The specific aggregation formula is shown in (1), where $\mathcal{D}_i$ represents the data set on edge node i, $|\mathcal{D}_i|$ represents the number of samples in data set $\mathcal{D}_i$, and $\mathcal{E}_t$ represents the set of edge nodes participating in training at iteration t. α is the learning rate. At the same time, the server

aggregates recommendation lists uploaded by the edge nodes to generate a list of popular files for caching

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t + \alpha \sum_{\forall i \in \mathcal{E}_t} \frac{|\mathcal{D}_i|}{\sum_{k \in \mathcal{E}_t} |\mathcal{D}_k|} \mathbf{H}_t^i. \tag{1}$$

B. Design of the Compression Algorithm

In the previous section, we introduced the content caching algorithm to predict cached files. However, the cost of communication is a serious problem in FL. Therefore, we incorporate a new compression algorithm into the content caching algorithm. This algorithm can reduce the communication cost by compressing the uploaded data and speed up the model training process. In this section, we will introduce the design of the compression algorithm and analyze the compression effect achieved by it.

To simplify the discussion, we define $\mathbf{H}_t^i$ to represent the gradients vector to be uploaded by the edge node i at iteration t. We define the dimension of this vector as N, so $|\mathbf{H}_t^i| = N$. Let H_{tn}^i represents the nth element in the vector $\mathbf{H}_t^i$.

The compression algorithm consists of two steps. First, we consider the gradients whose absolute value much greater than 0 as important gradients. These gradients have a great impact on the update of the corresponding parameters. The edge nodes should upload these gradients as precisely as possible. On the contrary, because those gradients close to 0 have little effect on model updating, we replace these gradients with a default value. Second, we use the K-means clustering algorithm to cluster important gradients, and use the centroid value of each cluster to approximate the gradient in the same cluster. Then, each edge node only uploads the centroid values instead of the exact gradient values to reduce the communication traffic. In addition, the uploaded compressed data needs to be verified using 4 smart contracts designed in blockchain to avoid malicious attacks by edge nodes.

1) Detecting Essential Gradients: We use the K-means clustering algorithm to detect important gradients.

We use the K-means algorithm to classify gradients close to each other into the same cluster. This algorithm can divide gradients into a given number of clusters by minimizing the distance between each gradient and its nearest centroid.

Assuming that N gradients are divided into C clusters, the total distance between the gradients and their nearest centroids is

$$J = \sum_{n=1}^{N} \left(|H_{tn}^i| - u_{r_n}\right)^2. \tag{2}$$

Here, u_{r_n} represents the centroid of the cluster r_n, and r_n represents the cluster to which the gradient H_{tn}^i belongs. Equation (2) represents the sum of the squares of the distances between all gradients and their corresponding centroid values. Hence

$$r_n = \arg\min_{1 \le c \le C} ||H_{tn}^i| - u_c| \tag{3}$$

and

$$u_c = \frac{1}{|\mathcal{R}_c|} \sum_{n \in \mathcal{R}_c} |H_{tn}^i|. \tag{4}$$

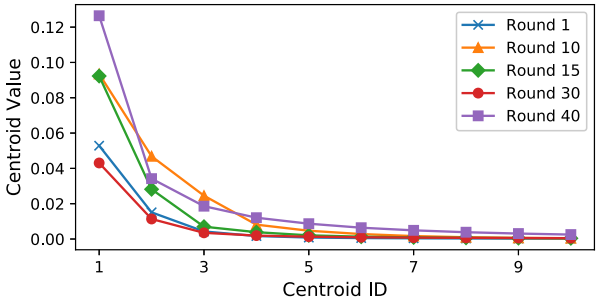


Fig. 5. Example to show the distribution of centroid values by clustering gradients in interaction rounds 1, 10, 15, 30, and 40.

Here, $\mathcal{R}_c$ represents the set of gradients in the cth cluster, u_c represents the centroid value of the cth cluster. Equation (3) represents that for each gradient, we take the nearest centroid value as the class of the gradient. Equation (4) represents that we use the average value of the gradients in the same class as the centroid value of the class. Because the importance of the gradient is determined by its absolute value, the computation here ignores the sign of H_{tn}^i.

We also need to consider how to determine the number C of clustering categories. We use the elbow method to enumerate different C's and then select the appropriate C based on the total distance calculated by (2).

After clustering all the gradients, we sort the clusters from 1 to C in descending order of u_c. We plotted the relationship between centroid c and centroid value u_c to observe the distribution of centroid values, as shown in Fig. 5.

We can observe that only the centroid value of the first cluster is significantly greater than 0. For all other clusters, their centroid values are close to zero. Therefore, we need to accurately upload the gradient in the first cluster.

2) Clustering-Based Quantization: The quantization step clusters important gradients to determine the centroid values to be uploaded by each edge node.

Recalling the content mentioned above, the gradient in cluster 1 is much greater than 0, so we define these gradients as important gradients. We define N_0 gradients in the first cluster and divide these gradients into C_0 clusters. Through this clustering operation, we only need to upload cluster IDs and centroid values when uploading data. We will further explain the cluster quantization process through a specific example.

As shown in Fig. 6, suppose we need to upload 16 gradients. Without compression, if each gradient occupies 4 B than we need to upload 64 B of data. After clustering and quantization, the amount of data to be uploaded is reduced to $4 \times 4 + 16 \times ([\log_2 4]/8)$ B. Here, 4×4 represents the uploaded centroid values and $16 \times ([\log_2 4]/8)$ represents the uploaded cluster IDs.

In the experiment, we set C_0 to 64 as such a setting can train a model with good performance. In practice, we need to determine the C_0 value based on the tradeoff between network conditions and model performance.

The quantization operation we mentioned above is performed on the gradient in the first cluster. For all other gradients close to 0, the edge node only uses their average value for uploading.

Fig. 6. Example to illustrate the clustering-based quantization.

Algorithm 1 Compression Algorithm

Input: Gradient vector $\mathbf{H}_t^i$ consisting of N gradients on client i; Hyperparameter C_0

Output: Vector $\mathbf{u_0}$ consisting of C_0 centroid values; Vector $\mathbf{r_0}$ consisting of N_0 cluster IDs of gradients; Vector $\mathbf{p}$ consisting of N_0 parameter IDs of gradients; *Avg*, the average of the rest unimportant gradients.

$C = ElbowMethod(\mathbf{H}_t^i)$
Define a vector $\mathbf{u}$ consisting of C centroid values
Define a vector $\mathbf{r}$ consisting of N cluster IDs
$\mathbf{r}, \mathbf{u} = Cluster(\mathbf{H}_t^i, C)$
Define a vector $\mathbf{n}_0$ consisting of gradients in cluster 1.
$N_0 =$ the number of gradients in $\mathbf{n}_0$
Define a vector $\mathbf{u_0}$ consisting of C_0 centroid values and a vector $\mathbf{r_0}$ consisting of N_0 cluster IDs
$\mathbf{r_0}, \mathbf{u_0} = Cluster(\mathbf{n}_0, C_0)$
$\mathbf{p}$: retrieves parameter IDs of gradients in the vector $\mathbf{n}_0$
Define a vector $\mathbf{n}_1$ for the rest gradients not in $\mathbf{n}_0$
$Avg = average(\mathbf{n}_1)$
return $\mathbf{u_0}, \mathbf{r_0}, \mathbf{p}, Avg$

3) Compression Algorithm and Analysis: Rather than uploading N gradients, we specify what an edge node needs to upload as follows: C_0 centroid values, N_0 cluster IDs and parameter IDs,[1] and a centroid value to replace the rest gradients. The specific algorithm details are shown in Algorithm 1. In this algorithm, the function *ElbowMethod*() enumerates the different numbers of clusters and calculates the loss value at this time using (2). When the number of clusters increases and the loss value does not decrease significantly, it means that the number of clusters at this time is appropriate. The function *Cluster*() will perform K-means clustering based on the input vector and the specified number of clusters, and return the cluster IDs vector and centroid values vector.

Next, we will analyze the compression ratio that the compression algorithm could be achieved. Without compression, if the edge node needs to upload N gradients and each gradient occupies 4 B, and the amount of uploaded data is $4N$ B.

After compressing the data with the compression algorithm, we need to upload C_0 centroid values, which will occupy $4C_0$ B. Communication traffic is $N_0([\log_2 C_0 + \log_2 N]/8)$ B to upload N_0 parameter IDs and cluster IDs. And we need to upload a default value to replace the unimportant gradients. Therefore, the amount of data to be uploaded by each edge node after compression is $4C_0 + N_0([\log_2 C_0 + \log_2 N]/8) + 4$ B. Let Γ denote the compression ratio, then the compression ratio achieved by the compression algorithm is

$$\Gamma_{\text{ClusterGrad}} = \frac{4N}{4C_0 + N_0 \frac{\log_2 C_0 + \log_2 N}{8} + 4}. \tag{5}$$

For the security of data, the compressed data obtained by edge nodes will be verified by four smart contracts of the blockchain designed above. When the blockchain verifies that the data is safe, the data will be recorded in the blockchain and retrieved by the server.

V. Performance Evaluation

In the experimental stage, we will verify the performance of the CREAT algorithm. We will look at the cache hit rate of the FPCC algorithm and compare it with the cache hit rate of CREAT applying the compression algorithm. We also simulated the network environment to detect the communication optimization of the CREAT algorithm.

A. Experiment Setups

We used real-world data sets MovieLens [36] in our experiments. In the MovieLens 1M data set, 6040 users rated 1 000 209 ratings on 3883 movies, while 1682 movies in the MovieLens 100k had 100 000 ratings from 943 users. To simulate the content request, we assume that the rated movie is the file requested by the user. Each movie rating corresponds to a request. Müller *et al.* [37] used a similar method to simulate the process of users' requests.

For the simulation of the system, we assume that there is a server and 100 edge nodes. The server is co-located at the center of the cell with a radius of 2 km, and the edge nodes are uniformly distributed in the cell.

According to the network simulations in [38], the average and maximum throughput of edge nodes θ_k are 1.4 and 8.6 Mb/s, respectively, which are realistic values in LTE networks. We consider that the average throughput is stable. However, to take into account the small changes in short-term throughput during the upload process, each time the edge node uploads the model, we will sample the throughput from Gaussian distribution with the mean and standard deviation given by the average throughput and its 10% value.

We compare our algorithm with two reference algorithms described as follows.

1) *Random:* Random algorithm randomly select F files from all files for caching. The algorithm does not consider the popularity of the files, and only randomly selects them. Therefore, it provides the worst cache efficiency. We use the cache efficiency achieved by this algorithm as a lower bound.
2) *Thompson Sampling:* Beta distribution is a family of continuous probability distributions defined in the interval [0, 1]. Beta distribution can describe various events in the 0-1 interval, so it is particularly suitable for modeling the probability of something happening. The principle of Thompson sampling is the Beta distribution.

[1]The server updates the corresponding parameters according to the parameter IDs.

Thompson sampling assumes that the probability of each file being cached is sampled from the beta distribution. The two parameters of the distribution are: a) hit and b) miss. We select the F file with the highest value to the cache to the edge node. Each time the cache file is updated, the beta distribution is updated according to the number of hits and misses. If a file is requested many times, then the sum of the two parameters corresponding to the file is large. So its Beta distribution will be very narrow. At this time, the probability that the file is cached to the edge node is basically determined. If the file has many cache hits, the Beta distribution value of the file will be very large, that is, there is a high probability that the file will be cached in the edge node. If a file is requested very few times, the Beta distribution of the file is very wide. Therefore, a large random number may be generated, so that the file is cached to the edge node to play an exploration role. According to the above description, we know that the Thompson sampling will decide whether to cache the file according to the previous request of the file. And Thompson sampling also explores files that are rarely requested. Therefore, Thompson sampling can achieve good cache efficiency.

We integrate the blockchain technology with the designed system. More specifically, we develop the four smart contracts based on Ethereum, which are mentioned in Section III. The decentralized participants will submit the task information, model gradients, and conduct transaction verification by electing consortiums randomly and periodically. In this case, the creditability of our system can be guaranteed.

B. Performance Evaluation

We use a ratio of the number of cache hits to the number of user requests called cache efficiency as a performance metric to evaluate our algorithm. Our experiment includes two autoencoders to find potential features of users and files to calculate similarity matrixes. Both autoencoders have a hidden layer activated by using ReLu. The edge nodes will update the cache files every hour. We investigated the cache efficiency of caches of different sizes between 50 and 400 files. In each round of communication, 30 edge nodes participate in the model training. As shown in Figs. 7 and 8, as the cache size increases, its efficiency of different algorithms also increases. We can observe that the cache efficiency of the CREAT algorithm is significantly better than the random algorithm and the Thompson sampling algorithm. Moreover, compared with the FPCC algorithm, the CREAT algorithm that compresses the gradients to be uploaded has no significant difference in cache efficiency.

Next, we study the time required for data upload after compression using the CREAT algorithm, and compare it with the time required for the FPCC algorithm. The network throughput at the time of uploading is simulated as the environment mentioned before. The result is shown in Fig. 9. It can be seen that the time required for uploading after compression has been greatly optimized.

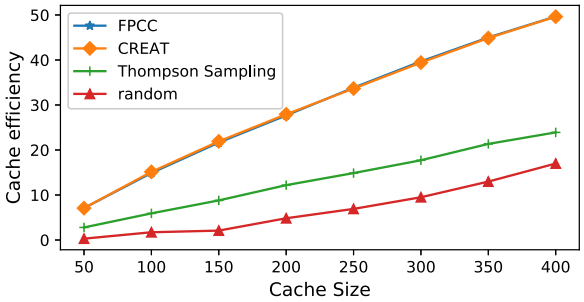


Fig. 7. Cache efficiency with different cache sizes (MovieLens 100k).

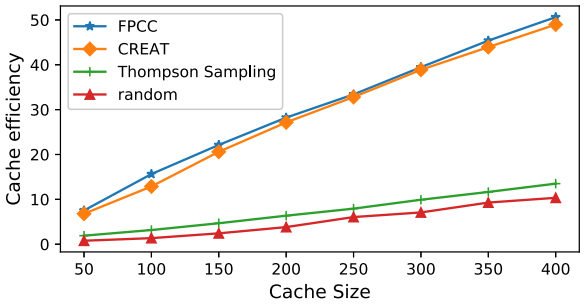


Fig. 8. Cache efficiency with different cache sizes (MovieLens 1M).

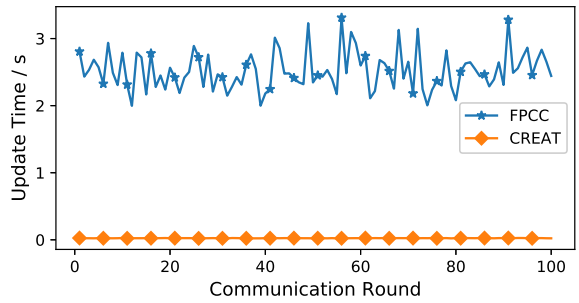


Fig. 9. Upload time with different algorithms ($r = 20$).

Figs. 10 and 11 depict the cache efficiency against the cache size with different numbers of participated clients in FL. On both 100k and 1M data sets, we can observe that the training effect after compression processing is not much worse than the uncompressed training effect. Therefore, it shows that the compression of the CREAT algorithm will not greatly affect the cache efficiency. Moreover, with the increase in the number of clients participating in training, the cache efficiency has not been greatly optimized. Therefore, increasing the number of clients participating in training in FL will not significantly optimize training results.

VI. Conclusion

In this article, we propose the CREAT algorithm, which can improve the caching efficiency of edge nodes and the communication efficiency of FL. The algorithm caches files by predicting the popularity of different files, thereby speeding up the response to requests from IoT devices. At the same time, the algorithm combines blockchain technology to ensure the security of the data transmitted by IoT devices and the gradients uploaded by edge nodes. In addition, we combined a

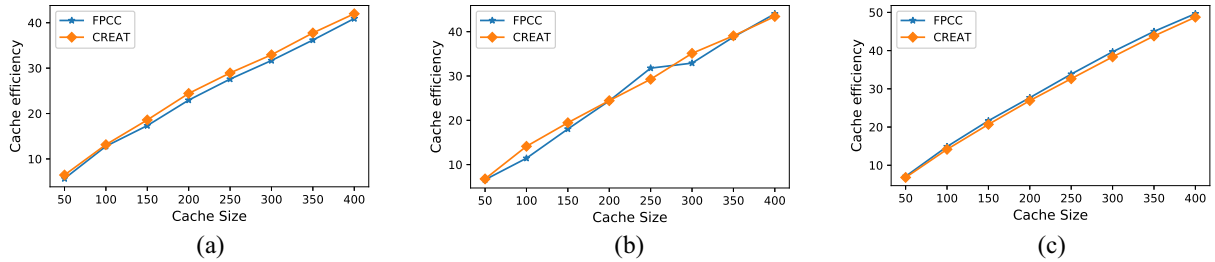


Fig. 10. Cache efficiency versus Cache sizes with different fraction of clients in MovieLens 100k. (a) Cache efficiency versus Cache sizes with 10% of clients in MovieLens 100k. (b) Cache efficiency versus Cache sizes with 20% of clients in MovieLens 100k. (c) Cache efficiency versus Cache sizes with 30% of clients in MovieLens 100k.

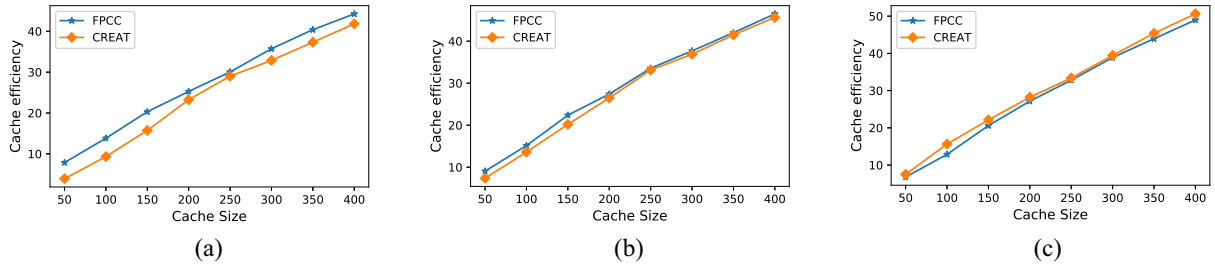


Fig. 11. Cache efficiency versus Cache sizes with different fraction of clients in MovieLens 1M. (a) Cache efficiency versus Cache sizes with 10% of clients in MovieLens 1M. (b) Cache efficiency versus Cache sizes with 20% of clients in MovieLens 1M. (c) Cache efficiency versus Cache sizes with 30% of clients in MovieLens 1M.

new compression algorithm to compress the uploaded gradients, thereby speeding up the training process of FL. It can be seen from the experimental results that the algorithm can significantly improve the caching efficiency of edge nodes. Compared with the FPCC algorithm, CREAT algorithm can greatly reduce the time required to upload data.

REFERENCES

[1] D. Evans, "The Internet of Things: How the next evolution of the Internet is changing everything," CISCO, San Jose, CA, USA, White Paper, 2011.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[3] C. Raphael. (Nov. 2015). *Why Edge Computing is Crucial for The IoT*. [Online]. Available: http://www.rtinsights.com/why-edge-computing-and-analytics-is-crucial-for-the-iot/

[4] Y. Shi and Q. Ling, "An adaptive popularity tracking algorithm for dynamic content caching for radio access networks," in *Proc. 36th Chin. Control Conf. (CCC)*, Dalian, China, 2017, pp. 5690–5694.

[5] R. Casado-Vara, F. de la Prieta, J. Prieto, and J. M. Corchado, "Blockchain framework for IoT data quality via edge computing," in *Proc. 1st Workshop Blockchain Enabled Netw. Sens. Syst.*, 2018, pp. 19–24.

[6] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Security*, 2015, pp. 112–125.

[7] W. Yu *et al.*, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[8] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[9] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 1146–1159, Feb. 2020.

[10] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," 2017. [Online]. Available: arXiv:1704.05021.

[11] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, and M. Xu, "A decentralized and trusted edge computing platform for Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 3910–3922, May 2020.

[12] L. Cui, S. Yang, Z. Chen, Y. Pan, M. Xu, and K. Xu, "An efficient and compacted dag-based blockchain protocol for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4134–4145, Jun. 2020.

[13] K. Zhang, S. Leng, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Artificial intelligence inspired transmission scheduling in cognitive vehicular communications and networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1987–1997, Apr. 2019.

[14] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.

[15] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[16] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[17] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[18] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Sep./Oct. 2019.

[19] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80–87, Jun. 2018.

[20] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

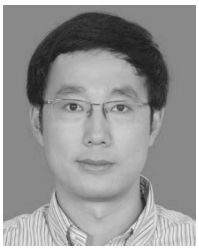
[21] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018.

[22] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[23] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, early access, Apr. 8, 2020, doi: 10.1109/COMST.2020.2986024.

[24] X. Yao, C. Huang, and L. Sun, "Two-stream federated learning: Reduce the communication costs," in *Proc. IEEE Vis. Commun. Image Process. (VCIP)*, Taichung, Taiwan, 2018, pp. 1–4.

[25] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.
[26] Q. Lu, X. Xu, Y. Liu, I. Weber, L. Zhu, and W. Zhang, "uBaaS: A unified blockchain as a service platform," *Future Gener. Comput. Syst.*, vol. 101, pp. 564–575, Dec. 2019.
[27] I. Weber, Q. Lu, A. B. Tran, A. Deshmukh, M. Gorski, and M. Strazds, "A platform architecture for multi-tenant blockchain-based systems," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Hamburg, Germany, 2019, pp. 101–110.
[28] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Japan, Rep. -01, 2019.
[29] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project, Zug, Switzerland, Yellow Paper, 2014.
[30] K. Zhang, Y. Zhu, S. Maharjan, and Y. Zhang, "Edge intelligence and blockchain empowered 5G beyond for the industrial Internet of Things," *IEEE Netw.*, vol. 33, no. 5, pp. 12–19, Sep./Oct. 2019.
[31] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in Internet of Vehicles," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4298–4311, Apr. 2020.
[32] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 706–719.
[33] M. Mukherjee *et al.*, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.
[34] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Oper. Syst. Principles*, 2017, pp. 51–68.
[35] Z. Yu *et al.*, "Federated learning based proactive content caching in edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, UAE, 2018, pp. 1–6.
[36] F. M. Harper and J. A. Konstan, "The Movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, 2015.
[37] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 2, pp. 1024–1036, Feb. 2017.
[38] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, 2019, pp. 1–7.



Laizhong Cui (Senior Member, IEEE) received the B.S. degree from Jilin University, Changchun, China, in 2007, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2012.

He is currently a Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He led more than ten scientific research projects, including National Key Research and Development Plan of China, the National Natural Science Foundation of China, the Guangdong Natural Science Foundation of China, and the Shenzhen Basic Research Plan. He has published more than 70 papers, including the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the *ACM Transactions on Internet Technology*, the IEEE TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, and IEEE NETWORK. His research interests include future Internet architecture and protocols, edge computing, multimedia systems and applications, blockchain, Internet of Things, cloud and big data computing, software-defined network, social network, computational intelligence, and machine learning.

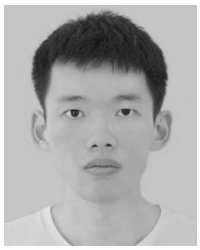
Prof. Cui serves as an Associate Editor or a member of Editorial Board for several international journals, including the *International Journal of Machine Learning and Cybernetics*, the *International Journal of Bio-Inspired Computation*, *Ad Hoc and Sensor Wireless Networks*, and the *Journal of Central South University*. He is a Senior Member of CCF.

Xiaoxin Su received the B.Sc. degree from Shenzhen University, Shenzhen, China, in 2020, where he is currently pursuing the M.Sc. degree.

His research interests include federated learning and blockchain.

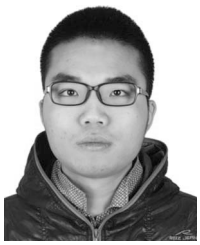
Zhongxing Ming received the B.Eng. degree from the College of Software Engineering, Jilin University, Changchun, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015.

He is currently an Associate Research Fellow with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include future Internet architecture, Internet of Things, blockchain, and edge computing.



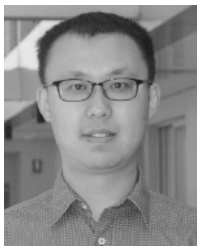
Ziteng Chen received the B.Sc. degree from Shenzhen University, Shenzhen, China, in 2018, where he is currently pursuing the M.Sc. degree.

His research interests include software-defined networking and blockchain.



Shu Yang received the B.Sc. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree from Tsinghua University, Beijing, in 2014.

He is currently an Associate Researcher with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include network architecture, edge computing, and high-performance router.



Yipeng Zhou (Member, IEEE) received the bachelor's degree in computer science from the University of Science and Technology of China, Hefei, China, in 2006, and the M.Phil. degree supervised by Prof. D. M. Chiu and Prof. J. C. S. Lui, in 2008, and the Ph.D. degree supervised by Prof. D. M. Chiu from the Information Engineering Department, Chinese University of Hong Kong (CUHK), Hong Kong, in 2012.

He is a Lecturer of computer science with the Department of Computing, Macquarie University, Sydney, NSW, Australia. From August 2016 to February 2018, he was a Research Fellow with the Institute for Telecommunications Research, University of South Australia, Adelaide, SA, Australia. From September 2013 to September 2016, he was a Lecturer with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He was a Postdoctoral Fellow with the Institute of Network Coding, CUHK, from August 2012 to August 2013.

Dr. Zhou is a recipient of the ARC Discovery Early Career Research Award in 2018.

Wei Xiao received the bachelor's degree in 2004, and the Doctoral degree in computer science and technology from Tsinghua University, Beijing, China, in 2014.

He has published several articles in IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, and ACM Multimedia Conference. His research interests include computer vision and multimodal information fusion.