Computers, Materials & Continua
DOI:10.32604/cmc.2021.018303
Article

Tech Science Press

ECC: Edge Collaborative Caching Strategy for Differentiated Services Load-Balancing

Fang Liu[1,*], Zhenyuan Zhang[2], Zunfu Wang[1] and Yuting Xing[3]

[1]School of Design, Hunan University, Changsha, China
[2]School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China
[3]Department of Computing, Imperial College London, UK
[*]Corresponding Author: Fang Liu. Email: fangl@hnu.edu.cn
Received: 04 March 2021; Accepted: 19 April 2021

Abstract: Due to the explosion of network data traffic and IoT devices, edge servers are overloaded and slow to respond to the massive volume of online requests. A large number of studies have shown that edge caching can solve this problem effectively. This paper proposes a distributed edge collaborative caching mechanism for Internet online request services scenario. It solves the problem of large average access delay caused by unbalanced load of edge servers, meets users' differentiated service demands and improves user experience. In particular, the edge cache node selection algorithm is optimized, and a novel edge cache replacement strategy considering the differentiated user requests is proposed. This mechanism can shorten the response time to a large number of user requests. Experimental results show that, compared with the current advanced online edge caching algorithm, the proposed edge collaborative caching strategy in this paper can reduce the average response delay by 9%. It also increases the user utility by 4.5 times in differentiated service scenarios, and significantly reduces the time complexity of the edge caching algorithm.

Keywords: Edge collaborative caching; differentiated service; cache replacement strategy; load balancing

1 Introduction

With the advent of 5G era, the number of edge network devices has greatly increased, and the amount of data that needs to be processed or cached on the edge is explosively increasing. For example, Intel reported that the data generated by an autonomous vehicle in one day was 4TB in 2016. Cisco predicted that by 2021, the growth rate of mobile data to be processed will far exceed the capacity of data centers. There are currently billions of IoT devices connected to the Space, Air, Ground, and Sea (SAGS) network, and a large amount of data is generated [1]. If all these data were up-loaded to the cloud for processing, it would certainly not be able to meet the low latency requirements of complex applications, such as on-board applications. Also, it is well known that there is a serious mismatch in I/O speed between the CPU and the disk. Adding a

memory cache between the two can solve this problem. Therefore, the edge cache in the network path between cloud and Internet of Things (IoT) devices is a good choice to reduce network latency. Yu et al. [2] proposed an algorithm based on the "IoT-Edge-Cloud" three-layer multi-hop model to evenly distribute computing tasks to network devices to process large amounts of data with huge potential value generated by IoT devices. Moreover, Liu et al. [3] proposed a matrix-based data sampling to alleviate the problems of data redundancy and high energy consumption that artificial intelligence is facing in collecting and processing big data.

Besides the latency issues, with the growth of network users, the traditional cloud-IoT or Cloud-Edge-IoT transmission architecture will cause some problems such as overloading of edge servers, single point failure of transmission links, and redundant transmission (such as hot videos) on resource-limited links.

In view of the above defects, there has been much research on the strategy of caching the most popular content locally according to the popularity of content, and a lot of work follows the Zipf distribution [4]. These methods can solve some of the above problems, but Zwolenski points out that the popularity of content on the Internet is easy to change greatly in a certain period of time, and it is easy to cause great deviation by using popularity [5]. In addition, Zhou et al. [6] proposed a new type of mobile photo selection scheme for congestion detection to reduce data redundancy on the server.

With the proliferation of Internet online requests, service providers will face the challenge of huge bandwidth overhead and the quality of service for users will be difficult to guarantee. Deploying the service close to the user and running the service cache on the edge server can effectively reduce access latency and improve the user's utility.

There is no doubt that collaborative caching strategy can reduce the probability of obtaining services from the original server. However, there are few effective caching schemes for caching the hottest services locally according to the popularity of services. It is worth noting that this is based on services. Taking services as the research object, the heat of services in different nodes is different for a period of time, which will lead to different loads on nodes (i.e., the requested number of the services). It is necessary to keep services load balance to prevent the server from going down. Not only does it reduce the service access delay but also improves the user's experience.

At present, most of the work doesn't consider the difference of user requests, that is, the demand of the Internet for differentiated services. For example, some users are VIPs who can prior to occupying server resources but some are not. Nielsen, a global monitoring and data analysis company, pointed out that 39% of consumers are willing to buy products with better quality but relatively expensive prices, while 15% of consumers are willing to buy products with basic functions but relatively cheap prices, and 1% of consumers are willing to buy low-priced products at the expense of quality [4]. Therefore, providing differentiated services can bring higher commercial benefits to service providers. In addition, most of the existing work does not consider the extra cost of a large number of sudden service requests on edge nodes, such as queuing delay, which will greatly increase the average access delay and lead to a bad experience for users. Caching services or applications which users frequently request to collaborative edge nodes can effectively reduce average access latency and network traffic [7–9].

In the Internet online request service scenario, we will supplement the differentiated Internet services and the average queuing delay of mass emergent requests at edge collaborative nodes, in

order to meet the differentiated service demands of users, reduce the average request access delay and improve user experience.

The characteristics of the edge collaborative caching strategy proposed in this paper are as follows:

- Our research object is different from the traditional file or content caching problem. We study the service caching problem in the Internet scenario where common content popularity model such as Zipf is not applicable to this scenario.
- The congestion of services in the node will damage user experience. When a user requests a service and the service runs on the edge node to answer the request, it needs to occupy node resources, such as CPU and memory resources. So the high concurrency of the service may easily cause other services to wait or even the node to go down, extremely increasing average access time of services.
- The node selection is divided into two stages where service should be placed. Referring to the characteristics of multi-node cooperative architecture (local cache hit delay $<$ neighbor node hit delay $<<$ cloud delay), when the service request frequency is low, it is randomly placed including the neighbor nodes and the local node. When the service request frequency reaches a large threshold, the service prefetching is placed in the local node.
- The same service delay, different user benefits. Differentiated services are common in Internet service scenarios. While considering the average access latency, we also need to consider the user benefits. Based on a real dataset, we classify different user requests into eight levels. Take it into cache replacement stage, optimizing the utility of users.
- The assumption of node blocking in point b and the assumption of differentiated service in point d above have been verified in real Google Trace (it is observed that there is blocking phenomenon and request level classification), and experiments are carried out based on this Trace, which reduces the delay and improves the user benefit.

Our key contributions of this paper are listed as follows:

- We specifically describe the online request application scenario of distributed edge cache, analyzes the Google data set, and finds that introducing relaying mechanism in nodes with relatively balanced must break the load balance. In particular, some cache algorithms without considering load balancing have too much queueing delay in some hot nodes, which leads to too much average access delay and too little benefit for users.
- By optimizing the average access time of online service requests and differentiated services for user requests, we propose a collaborative edge caching algorithm with differentiated services and load balancing. Compared with the classical cache replacement algorithm and the advanced online edge cache algorithm, analyzes the effectiveness of our proposal.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 introduce our system modeling and problem formulation. We present effective algorithms in Section 4. Experiments and performance evaluations are in Section 5. Finally, Section 6 concludes this paper and discusses the future work.

2 Related Works

Among the traditional online caching algorithms, the most widely used is LRU [10]. With low spatial complexity, it performs well in cache hit ratio evaluation because online requests often have the characteristics of "locality principle." On the edge cooperative cache scenario, LRU which is naturally modified also performs well in hit ratio evaluation [9]. Edge caching

technology has developed rapidly and it can be traced back to content distribution network technology [11]. In recent years, many excellent works were proposed on edge caching research. According to the use of tools or the field of studies, it can be divided into three kinds: D2D (Device-to-Device) communication aided edge caching [12,13], Game theory aided edge caching which reduces operator cost or increases profits [14], and edge collaborative caching which reduces the service response/access time [8,9].

D2D communication aided edge caching. Golrezaei et al. [11] proposed an architecture for caching popular video content based on the edge caching net-work assisted by D2D, and proved that D2D communication can effectively improve system throughput. On the D2D aided edge network, Wang et al. proposed an effective hierarchical collaborative caching algorithm for unloading network traffic, which takes the social behavior and preferences of mobile users and cache content size into account [12]. Besides, it is also a popular method to establish game models for edge cache network and it takes system cost and benefit as the optimization goal. Li et al. [13] on the edge of the small commercial network, proposed that the competition relation between NEP (network equipment providers) and VSP (video service provider) can be built as a Stackelberg Game model. By describing the cache equipment rental and deployment strategy, it optimizes the benefit of the NEP and VSP.

Game theory aided edge caching. Cao et al. [14] conducted auction modeling for the content delivery relationships among SP (service providers), users and content providers on edge cache network, and reduced the cost of SP and maximized the revenue of SP by using Myerson optimal auction model. Wu et al. [15] devised a distributed game-theoretical mechanism with resources sharing among network service providers with the objective to minimize the social cost of all network service providers, by introducing a novel cost sharing model and a coalition formation game.

Edge collaborative caching. Tan et al. [8] studied online service caching in the multiple edge node collaboration scenario, and proposed an asymptotic optimal cache replacement algorithm with the goal of optimizing network traffic and other costs. Ma et al. pointed out that due to the heterogeneity of edge resource capacity and the inconsistency of edge storage computing capacity, it is difficult to make full use of edge resource storage and computing capacity in the absence of collaboration between edge nodes. To solve this problem, they considered edge collaborative caching based on Gibbs sampling and Water falling algorithm, reducing the service outsourcing traffic and response time [4]. Hao et al. [7] proposed an edge intelligent algorithm in the heterogeneous Internet of Things architecture to jointly optimize the wireless communication cost, the collaborative cache and the computing unloading cost in the edge cloud, so as to minimize the total delay of the system. Wu et al. [16] proposed Edge-oriented Collaborative Caching (ECC) in information centric networking (ICN). In ECC, edge devices (such as edge server, micro data center, etc.) cache file contents while routers only maintain file cache indexes which are used to redirect subsequent requests towards the cached file content. Ren et al. [17] proposed a hybrid collaborative caching (Hy-CoCa) design that places cache in nodes, node groups and nodes in the network according to content popularity to further reduce delay and energy consumption.

There have been some researches on edge service caching algorithms, but the difference of service requirements has not been considered. In addition, most studies have ignored the unbalanced load of a large number of user requests on the edge server, which may cause congestion on edge servers, or even server downtime.

3 System Modeling and Problem Formulation

From the collaborative model of edge caching, it can be divided into cloud-edge, edge-edge and edge-IoT collaboration. The system proposed in this paper is based on the mode of cloud-edge and edge-edge collaboration to study the cooperative caching strategy. In this model, we study the cooperative caching strategy applied by Internet service in edge nodes. It is reasonable to assume that the cloud center has configured with all Internet service applications. Due to the limited storage capacity of edge nodes, the installation configuration can only be performed in the node after downloading/acquiring source files (or application installation packages) from the cloud center. Usually, due to the limited capacity of edge nodes, after installing new service applications, edge nodes will discard their source files (or application installation packages).

In the system architecture in Fig. 1, when an Internet user issues a request for application services which were deployed in the cloud or edge nodes, edge nodes will respond to the request with four different actions according to their cache hits. Denote the user request as $r := (f, s, p)$, the requested service as f, the edge node/server as s, and the priority of the request as p.

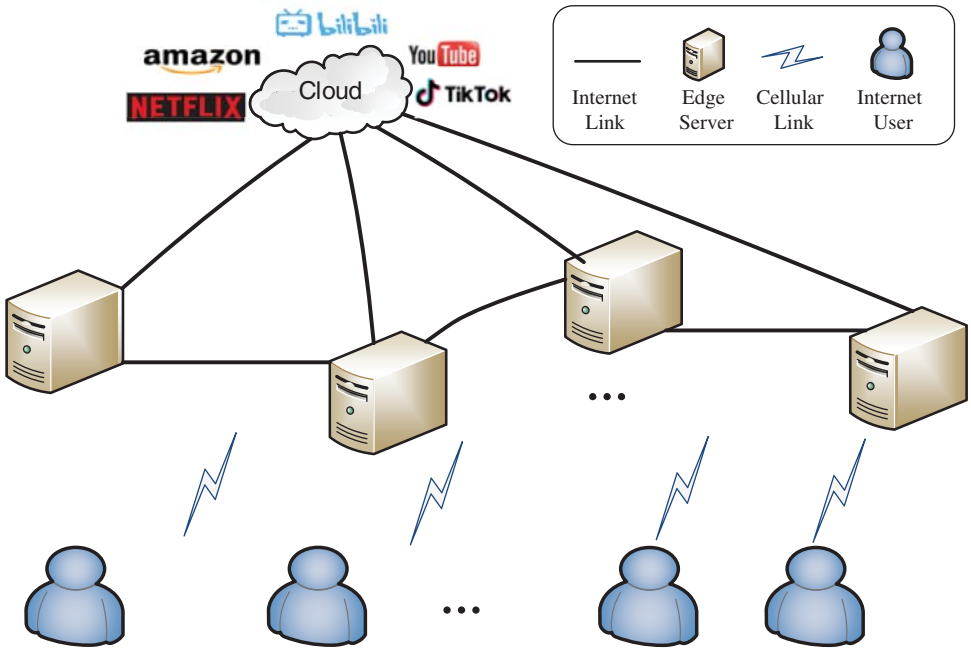


Figure 1: An example for edge collaborative caching system

First, if the local edge node s, such as the base station, router and other devices with storage and networking capacity, has deployed the service, the user request $r := (f, s, p)$ locally hit. The access delay of the local hit request is recorded as t_l, generally speaking, t_l is small.

Second, if the local edge node s is not hit, and the neighbor node s' has deployed services f, s will relay the request $r := (f, s, p)$ to s'. The access delay of the relaying hit request r is denoted as t_r, which is usually small.

Third, if none of the edge nodes are hit, the local node s will send request $r := (f, s, p)$ to the cloud data center (i.e., bypassing). The user request r is not hit and the request access delay is denoted as t_b which is large usually.

Fourth, if the local node or neighbor node does not hit multiple times, the edge node s and s' will download the service application source file or application installation package from the cloud and configure it to s or s'. This action is denoted as fetch, and the time cost/delay is t_f which is usually large.

It is reasonable to assume that all service applications (i.e., files, for simple) are accessible in the cloud center, and that the edge nodes have limited cache space, so only some files can be cached. Suppose there are m edge nodes in the cache system, denoted as the node set S, and the cache space of the node s_i is k_i file slots ($i = 1, \ldots, m$), that is, the total capacity of the cache system is $\sum_{i=1}^{m} k_i$. Assume that all the file sets are F each file takes up a slot in the file. The user request is denoted as $r := (f, s) \in \mathrm{F} \times \mathrm{S}$, where (f, s) represents the request r access f from the edge node s. If cached, the response will be quick, otherwise it will be relayed to a neighbor node s' or bypassed to the cloud for a response.

As mentioned above, the high concurrency of the service may easily cause request congestion. When a request is blocked in a node, there is a queueing delay. Define the queueing ratio for the user's request, called the mean blocking rate $p_{queueing}$, as shown in Eq. (1).

$$p_{queueing} = \frac{n_{queueing}}{n_{request}} \tag{1}$$

where $n_{queueing}$ represents the number of requests blocking the queue, and $n_{request}$ represents the number of requests.

Define the total queueing delay of the request in the node as shown in Eq. (2).

$$T_{queueing} = \sum_{i=1}^{K} i * t_{avgQ} \tag{2}$$

where K is the number of queueing tasks and t_{avgQ} is the average queuing delay, which is usually set to 100 milliseconds (ms).

Cache hit ratio hr is an essential performance indicator for cache system evaluation, and its definition is shown in Eq. (3).

$$hr = \frac{h_{local} + h_{relay}}{n_{request}} \tag{3}$$

where h_{local} represents the number of local hits, h_{relay} represents the number of relay hits, and $n_{request}$ represents the number of requests.

Furthermore, average access delay t_{avg} is also an essential performance indicator for cache system evaluation, and its definition is shown in Eq. (4).

$$t_{avg} = t_l * hr_{local} + (t_l + t_r) * hr_{relay} + (t_l + t_r + t_b) * p_{bypass} + (t_l + t_r + t_b + t_f) * p_{fetch} + t_q * p_{queueing} \tag{4}$$

Among them, $hr_{local} = \frac{h_{local}}{n_{request}}$, $hr_{relay} = \frac{h_{relay}}{n_{request}}$. p_{bypass} refers to the proportion of the local nodes to select the user request bypass to the cloud to serve, among which the number of bypass is n_{bypass}, $p_{fetch} = \frac{n_{fetch}}{n_{request}}$ refers to the proportion of the number of installing and configurating services to total requests, t_l represents the access delay from user to the local node, t_r represents

the access delay from user to the neighbor node, t_b represents the access delay from user to the cloud, and t_f represents the time it takes to configure the service/application installation.

In addition, we observe the load balancing of nodes with the edge node load variance *va*, which is defined as Eq. (5) and represents the stability of node response delay. From the perspective of users, the greater the variance of node load, the greater the delay difference of service response request perceived by users (i.e., the delay of user request response is sometimes large and sometimes small).

$$va = \sum_{i=1}^{m} (AVG(hc) - hc(s_i))^2 \tag{5}$$

where $AVG(hc)$ is the function that finds the average number of loads on all nodes.

As mentioned above, differentiated services are common in Internet service application scenarios. In order to meet the needs of differentiated services and optimize user benefits, we analyzed the relationship between the priority and frequency of user requests in the Google dataset and found that they are not inversely proportional, as shown in Fig. 2. Considering the priority and frequency of user requests, the definition of service level of user requests is shown in Eq. (6).

$$u_{level} = frequence * priority \tag{6}$$

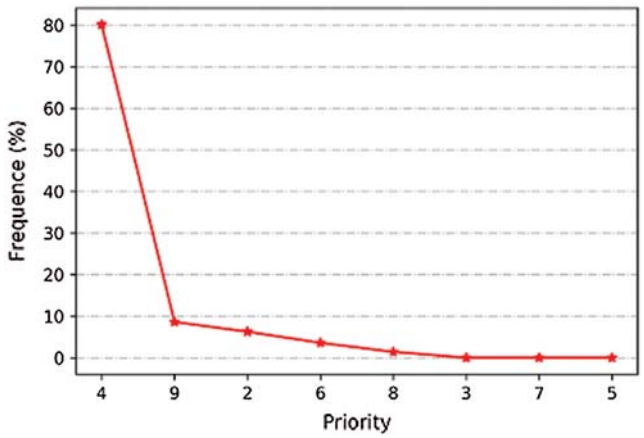


Figure 2: Priority and frequency of user request in google dataset

When the user request $r := (f, s, p)$ get the response in the cache system, as shown in Fig. 1, according to the service level and the response action obtained, the user utility $u_{request}$ is intuitively defined as shown in Eq. (7).

$$u_{request} = \begin{cases} u_{level} * 100, & \text{if local response;} \\ u_{level} * 10, & \text{if relay to neighbor;} \\ u_{level} * 1, & \text{if bypass to cloud;} \\ -u_{level} * 1000, & \text{if fetch requested file to local.} \end{cases} \tag{7}$$

Since different service requests have different user utility, in order to improve sum of user utility, we can replace the services with lower cumulative utility first. If the cumulative utility is the same, then consider the least recently used (LRU) strategy to replace the least recently used service.

4 Distributed Edge Collaborative Caching Mechanism

Similar to the memory cache in a computer, edge caching process can be roughly divided into three stages, data prefetch, node selection and cache replacement stage. In particular, we optimize an algorithm for node selection stage based on the hot load node of probability choice node (i.e., Select-the-node-with-probability), and boost a differentiated service cache replacement algorithm for cache replacement stage based on the recent minimum user benefit respectively (i.e., LUULRU-Fetch).

Algorithm 1: Select-node-with-probability (r)

Input: load count $hc(s_i)$ of edge node s_i, user request $r := (f, s)$
Output: selected node $s'' = nodeIndex_{selected}$
1: $nodeIndex_{selected} = s$
2: if edge node s have a slot q, and f in q then
3: terminated.
4: else
5: denote max load count in all edge nodes as $hc_{max} = MAX(hc)$
6: initialize random interval size $random_{size} = 0$
7: for $i = 0, \ldots, m$ then
8: $tmp = hc_{max} - hc(s_i)$
9: $random_{size} += tmp$
10: $p_i += tmp$
11: $random_{num} = RANDOM(0, random_{size})$
12: for $i = 0, \ldots, m$ then
13: if $random_{num} \leq p_i$ then
14: $nodeIndex_{selected} = i$
15: return $nodeIndex_{selected}$

Above, lines 5 to 11 in algorithm 1 show how to obtain random numbers for the node selection stage. Where, line 5 of the algorithm represents the maximum number of loads $MAX(hc)$ within the node load count table hc, and $RANDOM(0, random_{size})$ of line 11 represents a random value within the interval $RANDOM(0, random_{size})$.

Differentiated services are common in Internet scenarios. For example, compared with normal users, VIPs can get superior service quality and better user experience. In order to meet the needs of differentiated services, we put forward differentiated service strategies.

Algorithm 2: Least user utility-least recently used cache replacement algorithm (LUULRU-Fetch)

Input: user request $r := (f, s)$, user utility u_j of requested file j, selected node s'' in selecting node stage.

Output: the file $f_{selected}$ selected and replaced.

1: if node s'' have a slot q, and file f in q, then
2: terminated.
3: else
4: if the cache capacity of node s'' is full, then
5: if the least-user-utility file in s'' is greater than 1 then
6: Find the least recently used file from the above list, and denote it as $f_{selected}$
7: else if the least-user-utility file in s'' is only 1 then
8: denote it as $f_{selected}$
9: replace $f_{selected}$ from s''
10: cache f to s''.

Edge collaborative caching strategy is proposed as follows (overall strategy, integrating differentiated service strategy and load balancing strategy). Combined with the node load balancing strategy and differentiated service strategy proposed above in the three stages of the cache replacement process (data prefetching, node selection and cache replacement). The specific algorithm is shown in Algorithm 3.

Algorithm 3: Edge collaborative caching strategy

Input: $\lambda = \frac{t_f}{t_r},\ \mu = \eta\lambda \geq \frac{t_f}{t_r},\ hc(s_i) = 0,\ U = 0,\ r := (f, s, p),\ S_1(f),\ S_2(r)$

Output: cache hit ratio hr, average access delay t, user utility U, node load variance va

1: for each request $r := (f, s, p)$ do
2: add f to file request queue $S_1(f)$, add r to request queue $S_2(r)$
3: if s have a slot q, and f in q then
4: $hc(s) += 1,\ U += getUtility(f, p, \text{'}local\text{'}),\ s$ responses r and its delay is t_l
5: else if neighbor s' have a slot q and f in q then
6: $hc(s') += 1,\ U += getUtility(f, p, \text{'}relay\text{'}),\ s$ responses r and its delay is $t_l + t_r$
7: else $U += getUtility(f, p, \text{'}bypass\text{'})$, cloud responses r and its delay is $t_l + t_r + t_b$
8: /* The following is the cache replacement process, including data prefetching, node selection, and cache replacement */
9: if $|S_1(f)| = \lambda$ then
10: Call Select-node-with-probability(r) algorithm
11: Call LUULRU-Fetch(r) algorithm
12: Empty $S_1(f)$
13: if $|S_2(r)| = \mu$ then
14: Call LUULRU-Fetch(r) algorithm
15: Empty $S_2(r)$

In the input section of Algorithm 3, $\lambda = \frac{t_f}{t_r},\ \mu = \eta\lambda \geq \frac{t_f}{t_r}$, where η is the smallest available integer. $S_1(f)$ and $S_2(r)$ mean request queues for logging f and r respectively, initializing empty.

Denote service load count in node s_i as $hc(s_i)$ and initialize 0, user utility $U = 0$, p in $r := (f, s, p)$ mean its priority.

Lines 1 to 7 of algorithm 3 describe the response action of edge node or neighbor node, or cloud service/response when the user request r access f. In line 4, $hc(s)$ represents the number of requests/loads processed in the node s, $getUtility(f, p, \text{'}local\text{'})$ represents Eq. (6) to calculate the user utility of f when the user request priority is p and the local server is served at the local node s. $getUtility(f, p, \text{'}relay\text{'})$ of line 6 and $getUtility(f, p, \text{'}bypass\text{'})$ of line 7 and so on. Relay means that the user's request is forwarded by the local node and responded by the neighboring node. Bypass means that user requests are bypassed by local nodes and responded by the cloud.

Lines 9 to 15 of Algorithm 3 describe the cache replacement (also known as cache update) process. Lines 9 to 12 describe that after prefetching f, Algorithm 1 (Select-node-with-probability (r)) is used for node selection, and finally Algorithm 2 (LUULRU-Fetch) is used for cache replacement.

Lines 13 to 15 of Algorithm 3, similar to lines 9 to 12, describe the process after prefetching f, selecting the current node, and finally using algorithm 2 (LUULRU-Fetch) for cache replacement.

5 Performance Evaluation

Based on the Task Event Table [18] in the real Data set Google Cluster Data 2011-2, we conducted a large number of experimental tests and performance analysis compared with the baselines. Due to probabilistic selection existing in some baselines and the proposed algorithm, we conducted 10 times and analyzed the results of the proposed algorithm improvements. Specifically, the baselines are the advanced Camul [8] and the classic LRUwithRelay and LRUwithoutRelay algorithms. The baselines are used to test and analyze important performance indicators such as hit ratio, average access delay and so on.

5.1 Experimental Setup

Considering the system shown in Fig. 1, we set some important parameters and explain them as follows. According to the experimental test results of Maheshwari et al. on edge cloud system in 2018, we set $t_l = 1$, $t_r = 10$, $t_b = 100$, and the unit of time was simply set as milliseconds (ms) [19]. Refer to Camul [8], the ratio of operation cost of FETCH and bypass is 10, so set $t_f = 1000$ *ms*. Due to the huge difference between the two poles of queueing delay in Google Cluster Data 2011-2, which is 1us $\sim 10^{13}$us, analysis of its "queue delay-ranking" shows that it has a strong long tail effect, so we adopted the approximate average value of the long tail and set it as $t_p = 100$ *ms*. In the actual scenario, queueing delay is closely related to machine I/O capability, data/request arrival rate, etc., and there is no universal value. Therefore, it is feasible to use the approximate value in the long tail for algorithm verification [18–23]. The selected experimental platform is shown in Tab. 1.

5.2 Additional Overhead Analysis of Algorithms

Suppose there are n user requests, m nodes in the system, and each node has e slots. Compared with the traditional LRU algorithm, the extra space overhead of each algorithm is shown in Tab. 2.

In short, the space overhead of each algorithm is: LRUwithoutRelay $<$ LRUwithRelay $<$ Proposal $<$ Camul.

Table 1: Experimental platform

Item	Experimental platform configuration
CPU	Intel(R) Core(TM) i7-6700 CPU@3.40 GHz 3.41 GHz
Memory	Kingston DDR4 2666 16 GB
Operation system	Windows 10 professional
Simulation environment	Spyder 3.3.6 (Anaconda3)
Language	Python 3.7

Table 2: The extra space overhead of each algorithm compared to LRU

Extra overhead	LRUwithRelay	Camul	Proposal
Neighbor node file record cost	$O(m*e)$	$O(m*e)$	$O(m*e)$
Request queue		$O(n)$	$O(n)$
Slot state table cost		$O(3m*e)$	
Node load information table cost			$O(m)$
File-user utility table			$O(m*e)$

5.3 Experimental Results

5.3.1 Impact of Cache Number

Observe the experimental results, as shown in Fig. 3, and explore the impact of the number of cache nodes on the algorithm performance: (a) with the increase in the number of nodes, similar to other algorithms with relaying mechanism, the hit ratio of proposal is almost unchanged; (b) As the number of nodes increases, the number of queueing tasks in each node decreases, the queueing delay decreases, and the hit ratio slightly increases, so the average access delay decreases; (c) As the number of nodes increases, the number of local hits decreases and the number of relaying hits increases for the algorithm with relaying mechanism. According to user utility Eq. (7), it can be seen that the user utility decreases. In addition, for LRUwithoutRelay algorithm, since nodes do not have relaying mechanism and fetch delay are relatively more, its user utility is usually low. (d) Except for LRUwithoutRelay which does not have the relaying mechanism and remains the load balance of original traces, which leads to the lowest load variance, the proposals are all better than the other two algorithms. (e) In terms of the algorithm's time overhead, Proposal is obviously superior to Camul, and even slightly superior to LRUwithRelay algorithm when the number of nodes is large.

5.3.2 Impact of Cache Size

Based on the experimental results, as shown in Fig. 4, we can analyze the impact of cache node capacity on algorithm performance: (a) the hit ratio of each algorithm increases with the increase of cache size, and the proposal is very close to Camul. (b) With the increase of cache capacity, the average access delay of LRUwithoutRelay decreases. The benefits brought by relaying mechanism make the other three algorithms not significantly changed. (c) Since user utility of LRUwithoutRelay is far less than 0 in small capacity, it is not shown here. It can be seen from Fig. 4c that proposal is superior to other algorithms. (d) With the exception of LRUwithoutRelay, the proposal is superior to other algorithms in terms of the load variance of performance

indicators. (e) Since the classical LRUwithoutRelay algorithm is the simplest, it has the lowest time cost. LRUwithRelay is similar to the proposal, while Camul needs to record the status of the slot in the node, so its time cost increases with the increase of cache capacity.

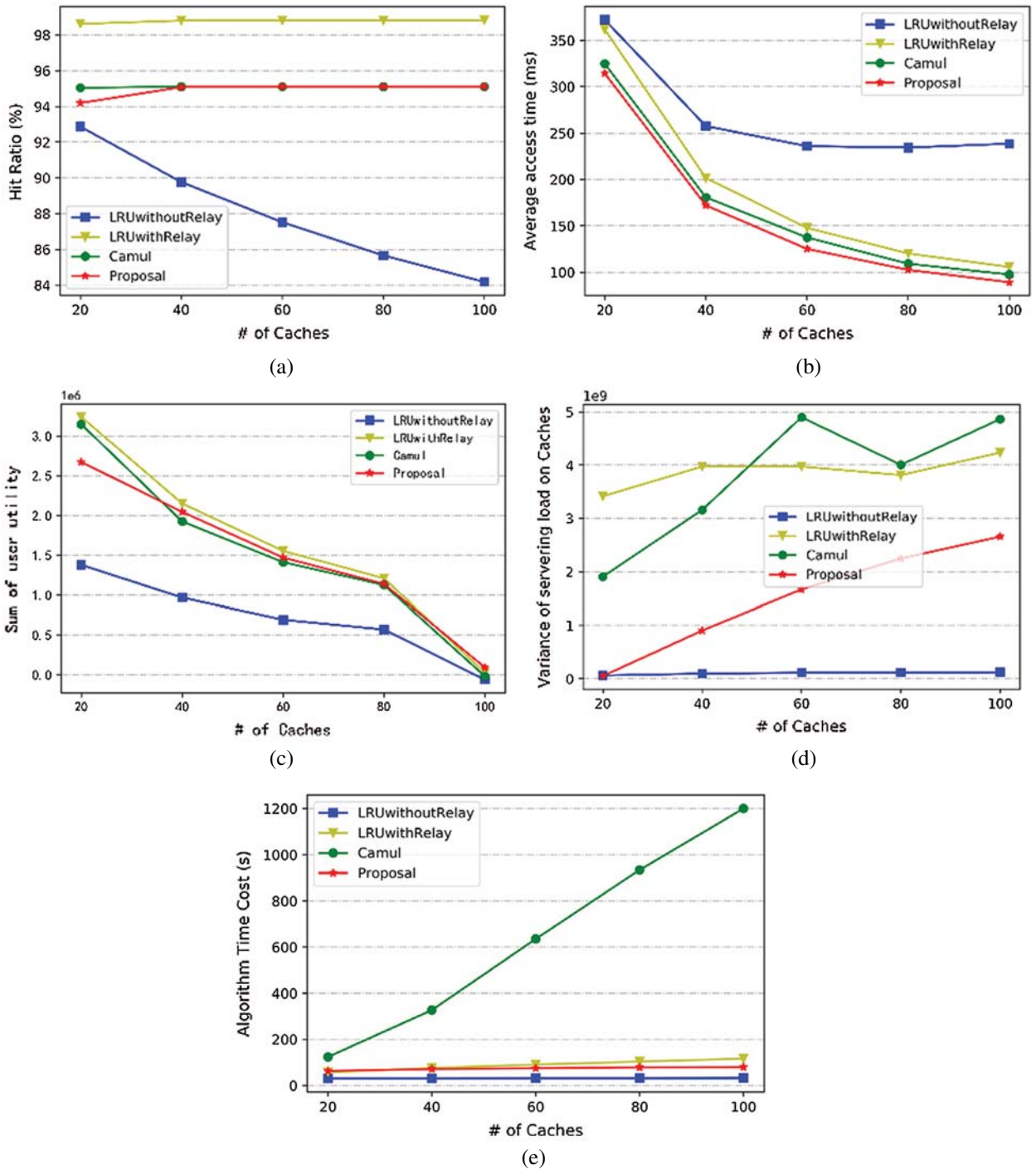


Figure 3: Impact of cache number

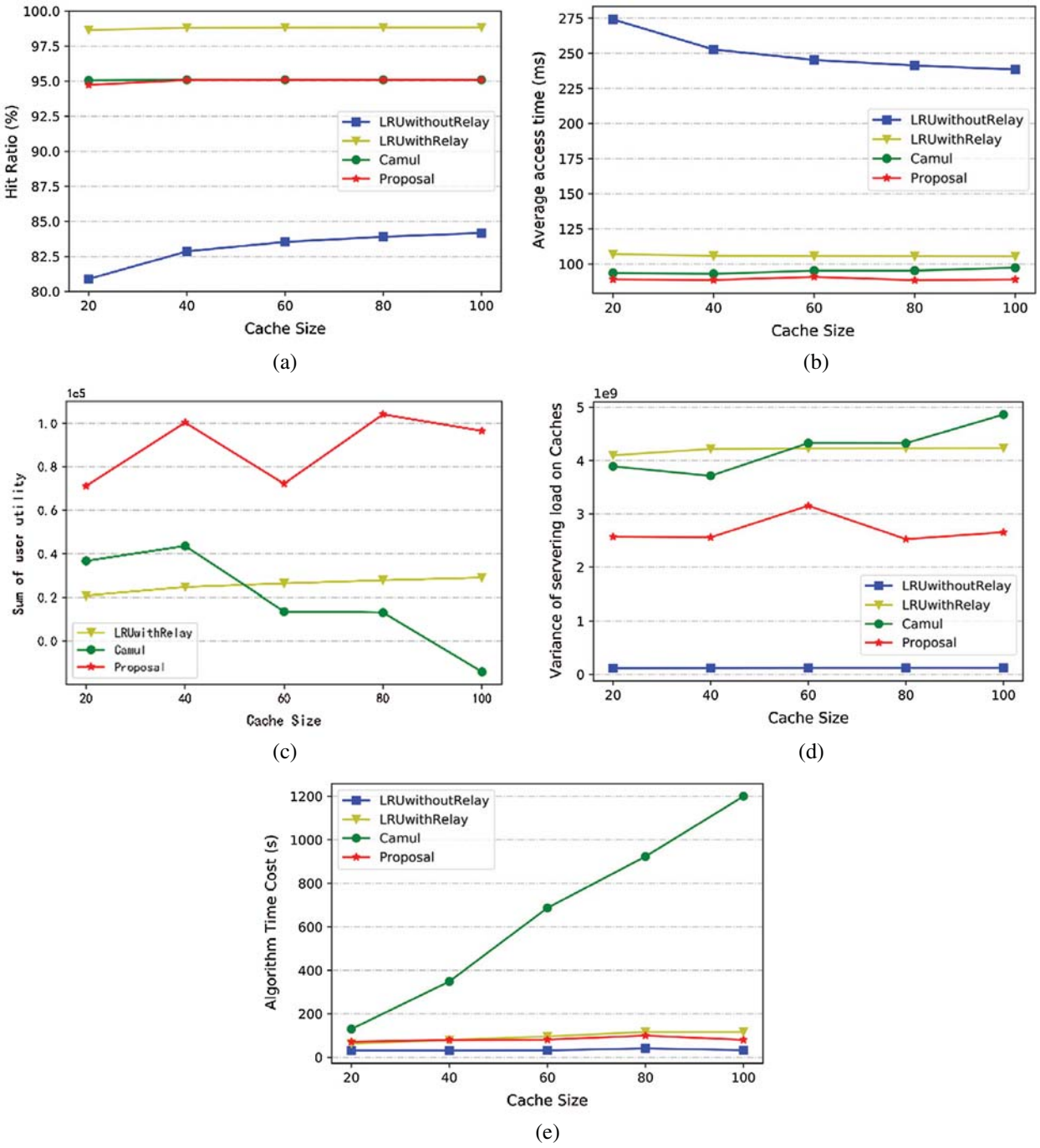
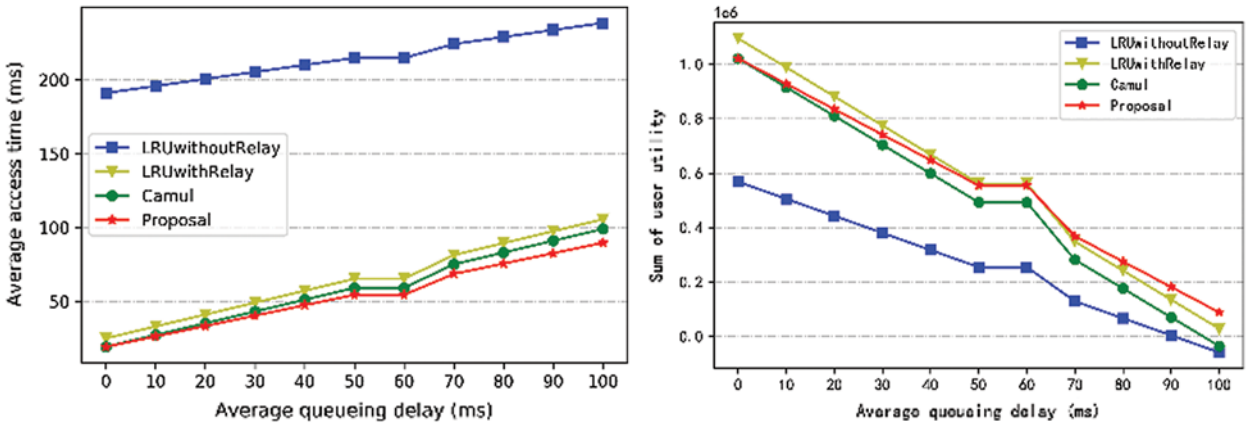


Figure 4: Impact of cache size

5.3.3 Impact of Average Queueing Delay

As shown in Fig. 5, the experiment shows the effect of average queuing delay on the performance of the algorithm: (a) it can be found that with the increase of average queueing delay,

the average access delay of all algorithms increases gradually. The more balanced the algorithm is, the slower the increase rate is. (b) With the increase of average queueing delay, the user utility of each algorithm decreases gradually. In addition, when the average queueing delay is within a reasonable range of 60 ms∼100 ms, the user utility of the proposal is optimal.



(a) (b)

Figure 5: Impact of average queueing delay

In addition, as can be seen from Figs. 3a and 4a, the proposal is not superior to LRUwithoutRelay on an important indicator of cache hit ratio. This is because LRUwithRelay with relaying mechanism treats all nodes as a whole, and its content repetition proportion between nodes is 0, while the content repetition between proposal and Camul is greater than 0. Besides, LRUwithoutRelay regards each node as independent and its content repetition between nodes is the highest.

6 Conclusions and Future Work

This paper describes the online request scenario of edge collaborative caching, analyzes the Google data set and finds that the introduction of relaying mechanism in nodes with relatively balanced load must increase the variance of node service load. However, some cache algorithms without considering load balancing have large queueing delay in some nodes, which leads to large average access delay and low utility for users.

By optimizing the average access delay of online service requests and differentiated services scenes, we proposed an edge collaborative caching algorithm based on differentiated services and load balancing. Based on differentiation of Internet service scenarios and the introduction of relaying mechanisms, we considered serious request queuing delays from the node load imbalance, compared to the classic cache replacement algorithm and the current advanced online edge caching algorithm. The experimental results show that our proposal not only guarantees the load balancing server process the request, but also reduces the average user requests access latency, improving the utility of users.

By caching the requested content, multiple nodes can provide services for users to speed up transfer. In the future, we will study the influence of single point transmission and coordinated multiple points (CoMP) transmission on edge cache, such as energy consumption and

computational complexity, so as to find a compromise between cooperative multi-point transmission and single point transmission.

Funding Statement: This work is supported by the National Natural Science Foundation of China (62072465) and the Key-Area Research and Development Program of Guang Dong Province (2019B010107001).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Huang, Z. Zeng, K. Ota, M. Dong and N. Xiong, "An intelligent collaboration trust interconnections system for mobile information control in ubiquitous 5G networks," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 1–1, 2020.

[2] M. Yu, A. Liu and N. Xiong, "An intelligent game based offloading scheme for maximizing benefits of IoT-edge-cloud ecosystems," *IEEE Internet of Things Journal*, vol. 99, no. 1, pp. 1–1, 2020.

[3] X. Liu, H. Song and A. Liu, "Intelligent UAVs trajectory optimization from space-time for data collection in social networks," *IEEE Transactions on Network Science and Engineering*, vol. 99, no. 1, pp. 1–1, 2020.

[4] X. Ma, A. Zhou, S. Zhang and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," arXiv preprint arXiv, 2020.

[5] M. Zwolenski and L. Weatherill, "The digital universe: Rich data and the increasing value of the internet of things," *Journal of Telecommunications and the Digital Economy*, vol. 2, no. 3, pp. 47–57, 2014.

[6] T. Zhou, B. Xiao, Z. Cai and M. Xu, "A utility model for photo selection in mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 48–62, 2021.

[7] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad *et al.*, "Smart-edge-cocaco: Ai-enabled smart edge with joint computation, caching, and communication in heterogeneous IoT," *IEEE Network*, vol. 33, no. 2, pp. 58–64, 2019.

[8] H. Tan, S. Jiang, H. C. Z. Han, L. Liu and Q. Zhao, "Camul: Online caching on multiple caches with relaying and bypassing," in *IEEE INFOCOM 2019 - IEEE Conf. on Computer Communications*, Paris, France, pp. 244–252, 2019.

[9] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[10] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Communications of the ACM*, vol. 49, no. 1, pp. 101–106, 2006.

[11] N. Golrezaei, A. G. Dimakis and A. F. Molisch, "Scaling behavior for device-to-device communications with distributed caching," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 4286–4298, 2014.

[12] X. Li, X. Wang, P. Wan, Z. Han and V. C. Leung, "Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1768–1785, 2018.

[13] J. Li, H. Chen, Y. Chen, Z. Lin, B. Vucetic *et al.*, "Pricing and resource allocation via game theory for a small-cell video caching system," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2115–2129, 2016.

[14] X. Cao, J. Zhang and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *IEEE 38th Int. Conf. on Distributed Computing Systems*, Vienna, Austria, pp. 388–399, 2018.

[15] G. Wu, J. Ren and F. Xia, "A game theoretic approach for interuser interference reduction in body sensor networks," *International Journal of Distributed Sensor Networks*, vol. 7, no. 1, pp. 1–1, 2011.

[16] H. Wu and M. Xu, "Intra-router routing mechanism for ForCES architecture," *Journal-Tsinghua University*, vol. 48, no. 1, pp. 124–124, 2008.

[17] D. Ren, X. Gui and K. Zhang, "Hybrid collaborative caching in mobile edge networks: An analytical approach," *Computer Networks*, vol. 158, no. 1, pp. 1–16, 2019.

[18] C. Reiss and J. Wilkes, "Google cluster-usage traces: Format schema," 2020. [Online]. Available: https://drive.google.com/file/d/0B5g07TgRDg9Z0lsSTEtTWtpOW8/view.

[19] S. Maheshwari, "Joint optimization of application specific routing in an anycast network," arXiv preprint arXiv, 2018.

[20] E. Modiano, J. E. Wieselthier and A. Ephremides, "A simple analysis of average queueing delay in tree networks," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 660–664, 1996.

[21] D. C. Lee, "Effects of leaky bucket parameters on the average queueing delay: Worst case analysis," *Proc. of INFOCOM'94 Conf. on Computer Communications*, Toronto, Ontario, Canada, pp. 482–489, 1994.

[22] A. Amin, X. Liu, I. Khan, P. Uthansaku and M. Forsat, "A robust resource allocation scheme for device-to-device communications based on q-learning," *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1487–1505, 2020.

[23] A. Wulamu, Z. Sun, Y. Xie, C. Xu and J. Sang, "An improved end-to-end memory network for QA tasks," *Computers, Materials & Continua*, vol. 60, no. 3, pp. 1283–1297, 2019.