The Journal of Systems & Software 186 (2022) 111195

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A novel load balancing scheme for mobile edge computing☆,☆☆

Zhenhua Duan [a], Cong Tian [a,*], Nan Zhang [a,*], Mengchu Zhou [b], Bin Yu [a,*], Xiaobing Wang [a], Jiangen Guo [a], Ying Wu [a]

[a] Institute of Computing Theory and Technology, and ISN Lab, Xidian University, Xi'an, 710071, China
[b] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA

ARTICLE INFO

Article history:
Received 3 October 2021
Received in revised form 19 November 2021
Accepted 18 December 2021
Available online 29 December 2021

Keywords:
Mobile edge computing
Cloudlet
MHP2P
Load balance

ABSTRACT

To overcome long propagation delays for data exchange between the remote cloud data center and end devices in Mobile Cloud Computing (MCC), Mobile Edge Computing (MEC) is emerging to push mobile computing, network control and storage to the network edges. A cloudlet in MEC is a mobility-enhanced small-scale cloud, which contains several MEC servers located in close proximity to mobile devices. The main purpose of a cloudlet is to stably provide services to mobile devices with low latency. When a cloudlet offers hundreds kinds of services to millions of mobile devices, it is critical to balance the loads so as to improve performance.

In this paper, we propose a three-layer mobile hybrid hierarchical P2P (MHP2P) model as a cloudlet. MHP2P performs satisfactory service lookup efficiency and system scalability as well as high stability. More importantly, a load balance scheme is provided so that the loads of MHP2P can be well balanced with the increasing of MEC servers and query load. A large number of simulation experiments indicate that the proposed scheme is efficient for enhancing the load balancing performance in MHP2P based MEC systems.

1. Introduction

Recently, mobile devices (such as smart phones, tablet computers etc.) are becoming more useful tools for learning, entertainment, social networking, updating news and businesses. In the beginning, mobile users do not get the same satisfaction compared with desktop due to resource limitations of mobile devices. With the tremendous advancements in wireless communications and networking, Mobile Cloud Computing (MCC) (Fernando et al., 2013; Noor et al., 2018; Bharati et al., 2021) is emerging as a new paradigm of computing in the last decade to improve above scenario. With MCC, the high-rate and highly-reliable air interface allows resource-constrained end-user devices to offload some computations to the remote resourceful cloud. After the evolution of MCC, many cloud computing services such as mobile health care, mobile learning and mobile gaming can be directly accessible from mobile devices (Hameed et al., 2019; Rimale et al., 2016; Cai et al., 2013; Qiao et al., 2020).

With the advent of cloud computing, the back-end server is typically hosted at the cloud data center. Though the use of a cloud data center offers various benefits such as scalability and elasticity (Li et al., 2021), its consolidation and centralization lead to a large separation between a mobile device and its associated data center. Offloading computation to the public cloud may involve long latency and low bandwidth for data exchange between the public clouds and edge device through the Internet (Satyanarayanan et al., 2009; Cho et al., 2016). Thus, MCC is not adequate for a wide-range of emerging mobile applications that are latency-critical.

As a new platform proposed by the *European Telecommunications Standard Institute* (ETSI) in 2014, Mobile Edge Computing (MEC) "provides IT and cloud-computing capabilities within the Radio Access Network (RAN) in close proximity to mobile subscribers" (Anon, 2014). In an MEC platform, the edge responsibility is increased. Computation and service are allowed in cloudlets at the edge, which include several proximal MEC servers. Mobile applications benefit from MEC by offloading their computation-intensive tasks to the MEC servers, aiming to reduce the network latency and bandwidth consumption (Mao et al., 2017; Ren et al., 2019; Zhang et al., 2018).

☆ This research is supported by the National Key Research and Development Program of China (2018AAA0103202); National Natural Science Foundation of China (62172322, 61751207, 61732013); Key Science and Technology Innovation Team of Shaanxi Province(2019TD-001).

☆☆ Editor: Gabriele Bavota.

* Corresponding authors.
E-mail addresses: zhhduan@mail.xidian.edu.cn (Z. Duan), ctian@mail.xidian.edu.cn (C. Tian), nanzhang@xidian.edu.cn (N. Zhang), zhou@njit.edu (M. Zhou), yubin9011@126.com (B. Yu), xbwang@mail.xidian.edu.cn (X. Wang), jgguo@mail.xidian.edu.cn (J. Guo), YingWu9208@126.com (Y. Wu).

https://doi.org/10.1016/j.jss.2021.111195
0164-1212/

There are three basic components in the model of MEC: (1) mobile devices include all types of devices (both mobile phones and IoT devices) connected to the network; (2) MEC servers are typically small-scale data centers deployed in close proximity with end users. MEC servers have the responsibility of traditional network traffic control (both forwarding and filtering) and hosting various mobile edge applications (edge health care, smart tracking etc.) and (3) public cloud is the cloud infrastructure hosted in the Internet.

Since the lower requirements for computation power and storage space, more devices can act as MEC servers. Harvesting the vast amount of the idle computation power and storage space distributed at the network edges can yield sufficient capacities for performing latency-critical tasks at mobile devices. Models of computation tasks, communications, mobile devices and MEC servers are studied in recent years. These state-of-the-art researches involve deployment of MEC systems (Cui et al., 2021), mobility management for MEC (Mehrabi et al., 2019) and security-and-privacy issues in MEC (Roman et al., 2018) etc. However, there are still challenges in a cloudlet including millions of MEC servers and mobile users:

(1) In a geographic area, there may exist a large number of MEC servers which can supply kinds of services for enormous amount of mobile users. For a mobile user, how to find out a suitable MEC server efficiently and then connect it is a concerned issue (Lee et al., 2019);
(2) With the popularity of MEC, more and more network elements at the network edge are willing to act as an MEC server. To ensure stability of a cloudlet, the leaving and joining of MEC servers should be detected in time. For key servers which manage others, recovery approaches should be performed to continue the MEC service (Satria et al., 2017);
(3) Unlike remote cloud data centers, the computation power and storage space in an MEC server is insufficient to serve too many mobile users simultaneously. For a cloudlet with several MEC servers, it is necessary to ensure performance of the service by applying load balancing schemes to managing MEC servers in the cloudlet (Zhang et al., 2020).

In order to construct a cloudlet performing satisfactory server lookup efficiency and system scalability as well as high stability, we integrate a three-layer mobile hybrid hierarchical P2P (MHP2P) network as a cloudlet into MEC systems. It can take advantages of both Distributed Hash Table (DHT) and flooding methods to improve performance (Duan et al., 2017). MHP2P uses Chord (Stoica et al., 2001) as the upper layer, clusters as the middle one and mobile devices as the lower one. Here the whole system is regarded as a Chord ring composed of a set of virtual nodes. Each virtual node corresponds to a cluster which is a group of MEC servers in the close geographical area.

In our system, an MEC server can be any device which can access network, including workstations, desktop computers and tablet computers. In practice, it can be a hospital, a market or a petrol station offering online service, or idle computers willing to provide computing service. MHP2P is good at efficiency because a search message is transferred within DHT on the top level in a large scale and flooding search is limited to clusters each of which consists of a small number of MEC servers only. MHP2P is stable because MEC servers joining or leaving the system is limited to a cluster. MHP2P has better scalability due to the combination of DHT and flooding approaches. It inherits the good scalability from DHT since clusters are organized into a Chord ring. Meanwhile, inter-cluster and intra-cluster load balancing schemes are provided to solve the problem of load imbalance in the MHP2P upper and middle layers.

The contributions of the paper are summarized as follows:

(1) A novel cloudlet model called MHP2P is presented for MEC systems. It has high stability, scalability and look-up efficiency. More importantly, load balancing schemes are provided so that the loads in MHP2P can be well balanced with the increasing of MEC servers and query load;
(2) Except for theoretical analysis, a large number of experimental simulations are conducted and the results show that the proposed schemes can significantly improve the degree of load balancing even the model size is in millions.

The paper is structured as follows. A motivating example is given in the next section. Section 3 provides our novel cloudlet model. Section 4 describes the details of the inter-cluster and intra-cluster load balancing schemes. A large number of simulation results are shown in Section 5. Finally, conclusion is drawn in Section 6.

2. A motivating example

In our daily life, health communities and city hospitals provide emergency services. An emergency department usually provides longer service time than other departments, even remains open 24 h. With the development of Internet, more and more emergency departments offer audio or video services for medical advice and diagnosis online. Based on the IP address, an ambulance can find the precise geographical location at the time when a patient requires.

When a patient expects to obtain some emergency medical services, he will first open an application installed on his mobile device. Then the application will send the query to the data center and receive information about suitable ones. In the case in which all related information is stored in the remote cloud, the long propagation distance from the end user to the remote cloud center will result in excessively long latency. What is worse, the increasing of query number will lead to the network congestion, in which situation the response takes more time. Suppose a patient needs service supplied by a emergency department, long time latency may make the patient miss the best time for rescue.

From the above scenario, it can clearly be seen that MCC is unsuitable for latency-critical service. Instead, an MEC can be employed to avoid frequently requesting the remote cloud for each query. Further, a cloudlet consisting of several MEC servers in proximity to mobile users is really necessary. Related information about service suppliers is stored in MEC servers. In fact, service suppliers can act as MEC servers since they are constantly online and have enough storage capability. With an ideal environment, sufficient service suppliers can join together in a cloudlet.

With the increasing of MEC servers and service queries, there are three problems that we must take into account:

(1) It is necessary to store information about different kinds of service suppliers in a specific rule which allows one to efficiently look up the required information;
(2) MEC servers should be well managed so that the failure of an MEC server should be detected in time. A cloudlet should maintain stability even when MEC servers frequently join and leave;
(3) To avoid network congestion and the overload of an MEC server, the loads of the whole cloudlet should be well balanced to ensure its performance and fairness.

3. A novel cloudlet model

This section presents the architecture of MHP2P as a novel cloudlet model. MHP2P is a hybrid hierarchical P2P network combining both unstructured and structured P2P networks. The upper

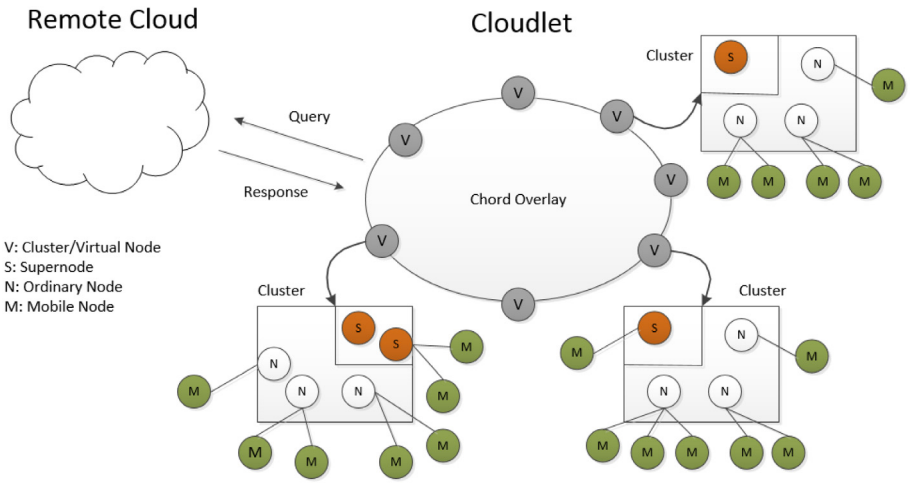


Fig. 1. Framework of MHP2P.

structured layer, which stands for the overall framework, is based on Chord using DHT. The middle unstructured layer, including several MEC servers, is based on a cluster using flooding. The lower layer are mobile devices which request services.

3.1. MHP2P framework

The framework of MHP2P is shown in Fig. 1. It evolves from HP2P architecture presented in Duan et al. (2017). As you can see, a node (N) stands for an MEC server in a cloudlet, which can be workstations, desktop computers, tablet computers etc. An MEC server has two functions: (1) to supply its own service and (2) to store metadata for other service suppliers. A piece of metadata is used to record the necessary information of a service supplier, including the service name, geography location, IP address, business hours, and service description etc. Nodes are classified into two types: Ordinary node (ODN) and Supernodes (S). Supernodes are the MEC servers with high network bandwidth, computation power and shared disk storage capacity. In general, a supernode will perform some specific tasks beyond ordinary ones. A cluster (CL) is a group of MEC servers in a geographical area. The same style of management and routing is followed in a cluster. A virtual node (V) is actually a cluster. However, in the hierarchical networks, a cluster appears at the top level as a virtual node. A mobile device (M) is a service requester, which is willing to connect a service supplier in time to obtain required service.

When a mobile device needs some service, it initiates a request to a boot server. According to the response from the boot server, the mobile device then connects with a suitable MEC server to publish its service query. After the look-up process in MHP2P, the mobile device finally gets a successful response if the service can be found in the cloudlet. Otherwise, the query will be sent to the remote cloud, which is out of scope of this paper.

MEC servers in our model are organized in clusters where messages are forwarded by Gossip Flooding (Voulgaris et al., 2003). In each cluster, some supernodes are elected from ordinary nodes. All clusters are organized in a Chord mechanism. Each cluster is as a virtual node in the Chord ring that has a routing table maintained by its supernodes. Communications among clusters are forwarded by their supernodes.

A candidate will substitute for a supernode if the supernode leaves its cluster. Only when no node exists or all supernodes collapse at the same time, will a cluster disappear. Hence, a cluster tends to be stable in a long period since its establishment. The Chord overlay routing information will not be influenced by the frequently joining or leaving of nodes, which will only impact on several neighborhood nodes in a cluster. Therefore, the whole network keeps stable. Further, the nodes with high bandwidth are effectively utilized with the employ of "power" supernodes. Finally, by combining the high search efficiency and scalability of DHT, the whole system performance can be significantly enhanced.

3.2. Architecture of upper layer

The original protocol of Chord supports just one operation: mapping a given key onto a node. In MHP2P, the original Chord is modified to map one dimensional hash space onto a cluster. The key is a service name and the value is IP address where the metadata is stored. Clusters in the modified Chord are identified by a consistent hash function (Karger et al., 1997). When a cluster is established, the IP address of its supernode is hashed to be an m-bit identifier by a hash function. With the same hash function, a service name is hashed to be m-bit ID. Then the relation between service and clusters is established (Duan et al., 2017).

A successor item of the finger table in the upper layer of MHP2P points a supernode table recording a group of supernodes in the same cluster. Given a service name as a key, the Chord overlay is able to find the cluster that is responsible for storing the key's value by a Chord routing mechanism. In a cluster, all supernodes maintain a finger table together. The supernode table of related clusters is updated when a supernode is changed. In a supernode table, it must be assured that the cluster can be valid if at least one supernode item is valid. Therefore, the existence of several supernodes in a supernode table enhances the stability of the routing table.

3.3. Architecture of middle layer

A cluster in MHP2P is used as the middle layer to restrict the flooding extension and enhance the system stability. MEC servers with vicinal geography locations are organized into a cluster. There are two kinds of nodes in a cluster: supernodes and ordinary nodes. Nodes with high capability act as the leaders of clusters, which are called supernodes. Supernodes collaborate to manage the cluster and maintain its finger table. A cluster is established when the first node joins it, and the first node's ID is considered to be the cluster's ID. At the same time, the first node becomes the first supernode in it. The nodes joining it afterward are generally considered to be ordinary nodes. Other supernodes can be selected by the cluster establisher. All supernodes work

together to manage the cluster to transfer the information across different clusters.

In one cluster, the main way for the communication of different nodes is a lightweight flooding in which a node just sends a newly generated message to a set of randomly selected neighbor nodes. These nodes do similarly in the next round, and so do other nodes until the message is spread to the entire cluster or TTL is reduced to zero.

3.4. Architecture of lower layer

The lower layer of our model is composed of mobile devices and the number of devices can reach millions. The application installed on a mobile device sends a message to a boot server first. After that, the boot server will establish a connection between the mobile device and its nearest MEC server. Then a query is sent from the mobile device to the MEC server, which contains the expected service name and IP address of the mobile device. The MEC server will help the mobile device forward the service query to the Chord. The detailed process to look up the service metadata will be given in the next subsection.

After one or more service suppliers are located, the MEC server connecting with the mobile device will first get the query result. The nearest service supplier will be selected by the MEC server and then sent to the mobile user. At last, the mobile user connects with the service supplier directly in peer-to-peer fashion.

3.5. Lookup service of MHP2P

After a query is transmitted from a mobile device to its nearest MEC server, it is sent to the Chord to locate the cluster that stores the metadata, and then queries are spread in the cluster through a lightweight flooding search.

The detailed search process of MHP2P is as follows:

1. A mobile device M sends a query to its nearest node P_i;
2. P_i in CL_i sends the query to S_i in CL_i;
3. S_i finds a successor (a cluster CL_j) through the Chord;
4. S_i forwards the query to successor CL_j;
5. In CL_j, its supernode S_j floods the query;
6. When found, node P_j in CL_j returns a successful response to P_i, go to 7; and otherwise P_i sends the query the remote cloud;
7. After the filtering procedure, P_i sends suitable information to M.

4. Load balancing in our model

The MHP2P upper layer is constructed by Chord which employs protocols based on DHT, thereby leading to a serious load imbalance problem that some clusters initiate flooding queries in high frequency while others are in idle states. In the MHP2P middle layer which uses the Gossip protocols, the load imbalance causes the problem that some nodes maintain too many metadata that consume too much of their processing capacity, while others maintain few metadata only.

To solve the load imbalance problem in MHP2P, this section presents a load balancing scheme, which consists of inter-cluster and intra-cluster load balancing. In the former, we achieve the MHP2P upper layer load balance by means of cluster splitting and cluster moving to adjust loads of nodes on the Chord ring. In the latter, the MHP2P middle layer reaches load balance by making full use of supernodes. Since the supernodes own the whole view of their corresponding cluster, their guidance can be used to enable a node with heavy load to find another with light load in the same cluster easily. For clarity, Table 1 lists the terms and notions used in this section.

Table 1
Terminology.

Notation	Definition
$Cluster_A$	Cluster A
$Length_A$	Length of address space that $Cluster_A$ should manage on the Chord ring
$Load_A$	The load of cluster A
$load_n$	The load of node n
$capacity_n$	The power of node n
$Load_{avr}$	The average cluster load by estimation
$rate_a$	The load rate of node a
$rate_{avr}$	The average load rate of a cluster
α	$\alpha \in (1, 2)$ is used to determine whether a node is with high load rate or low load rate
β	$\beta \in [0, 0.5)$ is used to determine whether a cluster can find a neighbor cluster to transfer or receive some loads
γ	$\gamma \geq 2$ is used to determine whether a cluster can split to reduce its cluster loads
k	k is used to determine the number of clusters from which a cluster should receive the load messages to estimate $Load_{avr}$

4.1. Inter-cluster load balancing

4.1.1. Load definition in inter-cluster load balancing

The load of a node in a network can represent the utilization of bandwidth, the utilization of CPU cycles, the number of items to maintain and the number of messages to process. In the MHP2P architecture, each node completes most of operations by sending and receiving messages such as flooding messages and metadata maintaining messages. Thus we define load as the number of messages a node to process in each period of time. Cluster load is defined in terms of the average number of messages in the cluster for a node to process:

$$Load_A = \frac{1}{Num_A} \sum_{n \in A} load_n \tag{1}$$

where Num_A is the number of nodes in cluster A, and n is a node index.

4.1.2. Average cluster load estimation

Since the MHP2P upper layer is a fully distributed network, we are not able to obtain its average cluster load directly. In its inter-cluster load balancing, a supernode in each cluster selects $k \times \log N$ other clusters randomly to obtain their cluster loads at regular intervals. Finally, a supernode estimates the average cluster load by calculating the average cluster load of $k \times \log N$ clusters as follows:

$$Load_{avr} = \frac{1}{k \cdot \log N} \sum_{m=0}^{k \cdot \log N} Load_m \tag{2}$$

where N represents the number of nodes available in the network. Because the inter-cluster load balancing scheme does not need high precision in an average cluster load estimation, we should not set k to a high value. According to our simulations, if the network is in a low degree of the inter-cluster load balancing, k is set to 4. If it is in a high degree of the inter-cluster load balancing, k is set to 1. In this way, the estimation of the average cluster load is in the interval $[0.85 \cdot Load, 1.15 \cdot Load]$ with high probability, where $Load$ is the average cluster load of the entire network in practice.

4.1.3. Principle of inter-cluster load balancing

Since the number of messages in each cluster cannot be controlled directly, we achieve the adjustment of the number of messages by controlling the number of metadata in each cluster

indirectly. In MHP2P, a supernode caching mechanism can avoid hot issues, such that the number of metadata in each cluster is proportional to the number of messages on the whole. In other words, the more metadata a cluster has the more messages it processes. This is because if a cluster has more metadata, it should have a higher probability to be retrieved, and also cause more messages to maintain metadata.

In the upper layer Chord ring, each supernode of clusters is required to send its load balance request to its neighbors from time to time. Let $Load_{avr}$ denote the average load of all clusters in the Chord ring and L_r denote the requesting cluster's load at present. If $L_r < Load_{avr}$, the requesting cluster is a light load cluster; if $L_r = Load_{avr}$, it is a moderate cluster; if $L_r > Load_{avr}$, it is a heavy load cluster; and further if $L_r \geq \gamma \cdot Load_{avr}$ with a threshold value γ, it is a very heavy cluster. To balance loads among clusters, the basic idea is: (a) to move some loads of a heavy load cluster to a light load one; and (b) split a cluster with very heavy load into two whenever a supernode is requesting.

(a) Moving strategy

To move loads reasonably between clusters, we set a threshold value V_m such that a light cluster r can acquire some loads from a heavy one x if $L_r < V_m \cdot L_x$ or a heavy one x can release some loads to a light neighbor cluster r if $V_m \cdot L_x \geq \cdot L_r$. Roughly speaking, the distance from a cluster to its predecessor cluster in a counterclockwise direction can be treated as its load. Therefore, if a light load cluster intends to acquire extra load from a heavy successor cluster, it needs to move in a clockwise direction closer to its successor cluster while if a heavy load cluster plans to release some loads to its light predecessor cluster, it needs to move in a counterclockwise direction closer to its predecessor cluster so as to change the distribution of metadata between a cluster and its successor cluster.

In order to balance loads precisely, on one hand, we need to know the percentage of loads of a cluster to be released; on the other hand, we need to estimate the exact distance a cluster will move clockwise or counterclockwise. For instance, if a cluster is requested to have equal load with its successor cluster, the load of the heavier cluster can decrease β ($\beta \in [0, 0.5)$) times of its load by moving clockwise or counterclockwise through a small amplitude. This is an effective load transfer between neighbor clusters. If β is 0.2, a load transfer can make a heavier cluster decrease 20 percentage of its load. In MHP2P, if two adjacent clusters are $Cluster_A$ and $Cluster_B$, and their loads $Load_A$ and $Load_B$ satisfy:

$$Load_B \geq \frac{Load_A}{1 - 2 \cdot \beta} \tag{3}$$

$Cluster_A$ can obtain some metadata to decrease β times of $Load_B$. In this case, we set $V_m = 1\text{–}2 \cdot \beta$. Since the supernode does not know every ID of the metadata in its cluster, we cannot determine the exact length a cluster should move on the Chord ring with low cost. Thus, we assume that IDs of metadata are evenly distributed in each address space of the cluster. In this way, we can estimate the *Length* a cluster should move by

$$Length = \frac{(Load_B - Load_A) \times Length_B}{2 \times Load_B} \tag{4}$$

To fully clarify the load transfer between two adjacent clusters, we have two cases, where $Cluster_B$ is assumed to be the successor of $Cluster_A$:

Case1: $Load_B \geq \frac{Load_A}{1-2\cdot\beta}$, $Cluster_A$ moves clockwise through

$$\frac{(Load_B - Load_A) \times Length_B}{2 \times Load_B}$$

on the Chord ring, while $Cluster_B$ transfers the corresponding Chord ring regions of metadata to $Cluster_A$.

Case2: $Load_A \geq \frac{Load_B}{1-2\cdot\beta}$, $Cluster_A$ moves counterclockwise through

$$\frac{(Load_A - Load_B) \times Length_A}{2 \times Load_A}$$

on the Chord ring, while $Cluster_B$ transfers the corresponding Chord ring regions of metadata to $Cluster_A$.

(b) Splitting strategy

If load L_r of a requesting cluster satisfies $L_r \geq \gamma \cdot Load_{avr}$ with a threshold value γ ($\gamma \geq 2$), its load is too heavy. In this case, a splitting strategy is employed (Duan et al., 2017). To do so, a new cluster node called y is created in the counterclockwise direction with a suitable distance from the cluster r, such that some of L_r can be moved to the new cluster y. As for how to split a cluster, it is out of scope of this paper. Please refer to Duan et al. (2017) for details.

The basic idea of the inter-cluster load balancing is to split clusters and balance the load among neighbor clusters (the successor and predecessor of a cluster). In this way, we reduce the load imbalance among clusters according to the individual topology of the MHP2P upper layer network.

4.1.4. Analysis of inter-cluster load balancing

In the inter-cluster load balancing mechanism of MHP2P, the upper layer network can achieve load balance within a narrow range of the Chord address space by cluster moving and splitting. New nodes continually joining the clusters with heavy load will lead to a high probability to split the latter. As the inter-cluster load balancing proceeds, the number of clusters increases through the splitting in the heavier load area of the upper layer Chord address space. Thus the load balancing in the whole MHP2P upper layer network is achieved.

As the supernode of each cluster needs to collect the loads of other $k \cdot \log N$ clusters to estimate the average load of all clusters, it needs to send $k \cdot \log N$ additional messages to obtain load information of these $k \cdot \log N$ clusters in every cycle. Note that only one supernode is in charging a cluster at any time though there may be a few supernode candidates within a cluster.

4.2. Intra-cluster load balancing

Intra-cluster load balancing is intended to solve the problem of load imbalance in the MHP2P middle layer network by the supernode in each cluster to schedule the loads in terms of computing power of its nodes.

4.2.1. Load rate definition in intra-cluster load balancing

According to our simulations, the load of each node mainly comes from flooding query messages, metadata maintenance and network topology maintenance. The load rate of a node can be defined as:

$$rate_a = \frac{load_a}{capacity_a} \tag{5}$$

However, for each node in the same cluster, various numbers of messages are mainly from metadata maintenance. The number of metadata maintenance messages for a node is determined by the number of metadata items this node should maintain. In MHP2P, the mechanism of the topology maintenance can guarantee that each node in the cluster has roughly the same number of neighbors so as to keep the balanced distribution of nodes in the flooding network. Moreover, we use an improved flooding strategy based on RBFS (Random breadth-first search) which can increase the randomness of the process in the flooding. These two mechanisms can keep the flooding query messages and network topology maintenance messages roughly the same for each node in the same cluster. Hence, the load of each node can be adjusted by numbers of metadata items.

Table 2
Simulation environment.

Environment parameters	Default values
Networksize	2^{16}
Ratio of node arrival:departure	1%:1% 1%:3% 3%:1%
Number of metadata for each node	10
Node capacity distribution	Pareto: shape 2
Simulation cycles	50

Table 3
Algorithm parameters.

Algorithm parameters	Default values
α	1.4
β	25%
γ	2.0
k	4 in first 10 cycles, 2 in 11 to 20 cycles, 1 in last 30 cycles

4.2.2. Principle of intra-cluster load balancing

To balance the loads among nodes within a cluster, the supernode maintains a hash table H and a sorted link list L. Each record in L contains load rate, number of metadata to be removed and time stamp of a heavy node A ($rate_A > \alpha \cdot rate_{avr}$) in descending order while each record in H contains node ID, and a pointer pointing to a relative record in L. Whenever a node requires to release some load (($rate_A - rate_{avr}$)·$capacity_A$), the supernode makes a record in H and its relative record in L. When a light load node B ($rate_B < (2-\alpha) \cdot rate_{avr}$) requests to acquire some load (($rate_{avr} - rate_B$) · $capacity_B$), the supernode checks list L and allocates sufficient metadata from suitable heavy nodes to it, and then updates H and L.

4.2.3. Analysis of intra-cluster load balancing

In the intra-cluster load balancing scheme, we make full use of the feature that there are several supernode candidates in each cluster. Because a supernode has a better view of the whole cluster, with its help, a node with low load rate can find the node with high load rate efficiently.

In order to keep the load information for a node with high load rate, a supernode should manage a hashtable and sorted link list. The size of the list is determined by the number of the nodes with high load rate. However, it does not cost too heavy load for the supernode, because the total number of nodes in a cluster is small. As the network evolves, the load is more and more evenly distributed, and the number of nodes with high load rate decreases quickly.

5. Simulation results for load balancing

In this section, we analyze the very significant performance improvements that are owing to the proposed algorithms by simulations. For fairness, a third-party simulation engine PeerSim (Anon, 2009) is used. PeerSim provides an engine and two simulation models: cycle and event-driven models. Table 2 lists parameters of the simulation environment for our algorithms, and the values we set unless otherwise specified.

We run each trial of the simulation for 50 cycles. According to statics of Kademlia (Steiner et al., 2007), the maximum number of nodes is three times the minimum number of nodes. Thus we set the proportion of nodes arrival to departure 1%:1%, 1%:3%, 3%:1% such that the number of nodes is kept the same, decreasing to roughly one-third, or increasing to 3 times, in 50 cycles. Finally, in order to indicate the reliability of the simulation results, each selected data point in our plots represents the average simulation result over 5 trials. Under 2^{20} nodes simulation scale, each trial takes about 10 h. Tables 2 and 3 list the parameters of our simulations, and the values we set unless otherwise specified.

5.1. Simulation of the inter-cluster load balancing algorithm

In simulations, we mainly calculate the ratio of the load of the cluster with the heaviest load and average cluster load in order to determine whether the inter-cluster load balancing algorithm can solve the problem that some clusters have too heavy load. In order to determine whether the algorithm has effect on improving the load balance of the whole MHP2P upper layer network, we calculate the relative standard deviation of the cluster load.

5.1.1. Simulations under different network sizes

Since the number of clusters and size of each cluster are both determined by the size of the whole network, we do simulations under different network sizes to determine whether the inter-cluster load balancing algorithm fits for MHP2P under different numbers of clusters and sizes of clusters. The network size is from 2^{14} to 2^{20}. Thus, the largest network size is more than one million nodes.

Fig. 2(a) shows that the inter-cluster load balancing algorithm can always keep the ratio of the load of the heaviest load cluster to average cluster load below 2 under different network sizes. By contrast, the ratio without the inter-cluster load balancing algorithm in MHP2P ranges from 3 to 7. In Fig. 2(b), the standard deviation of the cluster load with the inter-cluster load balancing algorithm in MHP2P is much smaller than that without it. In conclusion, the inter-cluster load balancing algorithm can make loads in a balanced distribution among clusters under different network sizes.

5.1.2. Simulations under different number of metadata items

As mentioned earlier, the metadata maintenance message is also an important part of the total messages of an MHP2P network. When the network has a different number of metadata items, the metadata maintenance messages are in different proportions of the total messages. Thus we do this simulation to determine whether the inter-cluster load balancing algorithm has effect on the MHP2P network with different proportions of the metadata maintenance messages. The average number of metadata for each node is 5, 10, 15, 20 and 25.

Figs. 2(c) and 2(d) show that the inter-cluster load balancing algorithm can not only sharply decrease the load of the heaviest load cluster but also improve the degree of the load balance of the whole MHP2P network under different numbers of metadata items. In fact, with 2^{19} nodes scale, the average load and heaviest load of a cluster is respectively 423 and 2811 items without using the algorithm while 362 and 718 with the algorithm.

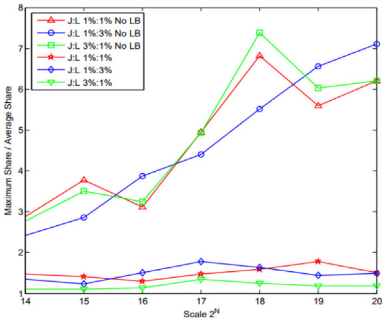
5.1.3. Simulations under different node request rates

Since different node request rates lead to different proportions of flooding messages in total messages of an MHP2P network, this simulation is used to indicate if the algorithm has effect on the MHP2P network under different numbers of flooding messages. In the simulation, we vary the node request rate between 0.0125 and 0.05.

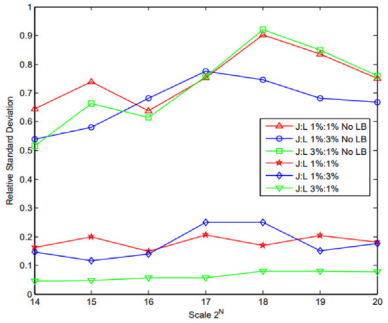
Figs. 2(e) and 2(f) show the ratio of the load of the heaviest load cluster to average cluster load keeps below 1.5 and the degree of the load balance of the whole MHP2P network is greatly improved with the inter-cluster load balancing algorithm.

5.1.4. Simulations under different settings of parameters β and γ

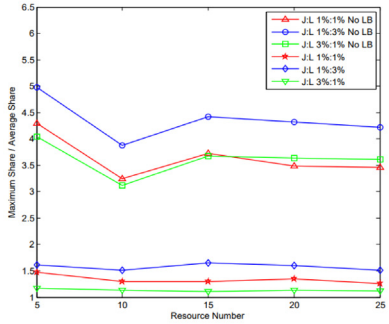
In the inter-cluster load balancing algorithm, β is used to determine whether a cluster can find a neighbor cluster to transfer some loads from it, while γ is used to determine whether a cluster with heavy load can be split into two sub-clusters. Therefore, their different settings may have different effects on the result of the inter-cluster load balancing algorithm. In this



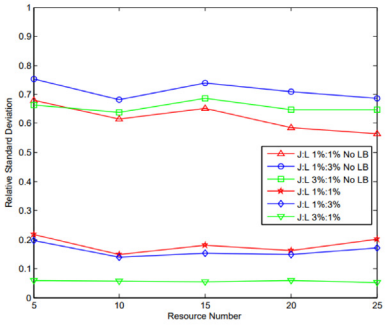
(a) The ratio of the heaviest cluster load and average cluster load under different network sizes.



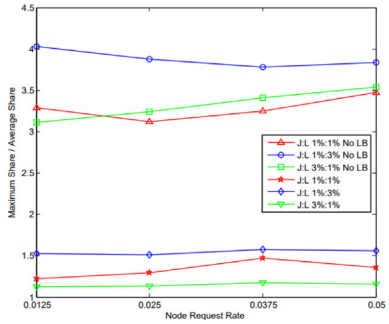
(b) The standard deviation of cluster loads under different network sizes.



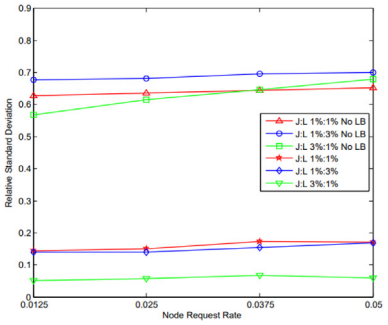
(c) The ratio of the heaviest cluster load to average cluster load under different numbers of metadata items.



(d) The standard deviation of the cluster load number of metadata items under different numbers of metadata items.



(e) The ratio of the heaviest cluster load to average cluster load under different node request rates.



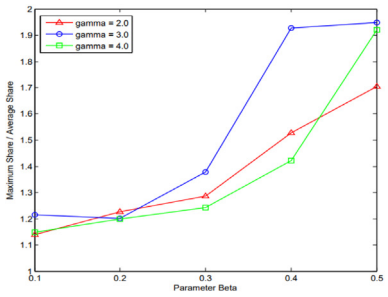
(f) The standard deviation of the cluster load numbers of metadata items under different node request rates.

Fig. 2. Simulations for the inter-cluster load balancing.

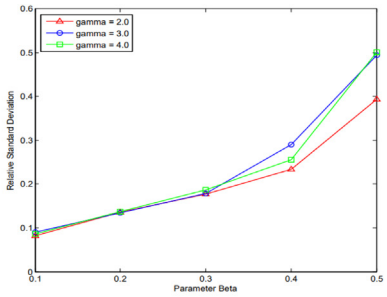
case, we do simulations about different settings of parameters β and γ in order to optimize their values. In simulations, when β is set to 0.5, it means the cluster will not move to promote the load balance.

Figs. 3(a)–3(f) show that the degrees of the cluster load balancing and metadata movement are mainly determined by β, while γ will take effect, when the value of β is so large that the cluster cannot find a neighbor cluster to transfer its load. According to the results in Figs. 3(a)–3(c), when the proportion of nodes arrival to departure is set 1%:1%, meaning that MHP2P size is not subject to quick changes, β can be valued within the interval [0.2,0.3], while γ can be valued as 2.0. Compared with the value of β in [0.2,0.3], that in [0.1,0.2] improves the small amplitude of the cluster load balancing by costing a large number of extra metadata movement, and the value of β in [0.3,0.5] sharply decreases the degree of the cluster load balancing. The value of γ is set to 2.0 which makes the cluster with heavy load split timely without bringing any extra metadata movements on the whole. The results in Figs. 3(d)–3(f) indicates β can be valued within the interval [0.2,0.4], while γ can be valued as 2.0 when MHP2P size is decreasing quickly.

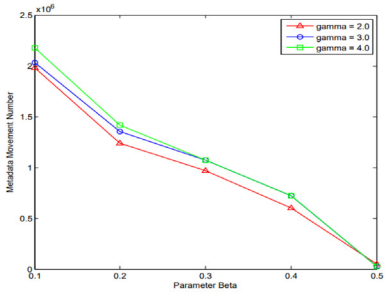
In Figs. 4(a)–4(c), when the network size is growing quickly, the ratio of the load of the heaviest load cluster to average cluster load and the standard deviation of the cluster load do not have obvious relationship with the value of β in [0.1,0.4]. In this case, the process of nodes joining heavier clusters can promote the better load balancing of the clusters. Hence, β can be set in the interval [0.4,0.5), and γ can be set as 2.0.



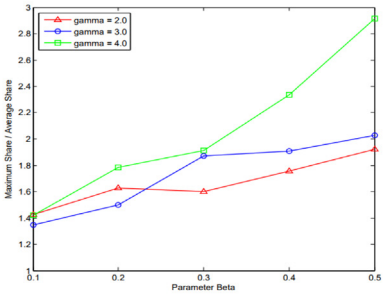
(a) The relation among β, γ and the ratio of the heaviest cluster load and average cluster load when the network size is not subject to quick changes.



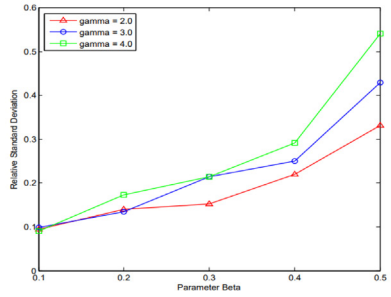
(b) The relation among β, γ and the standard deviation of the cluster load when the network size is not subject to quick changes.



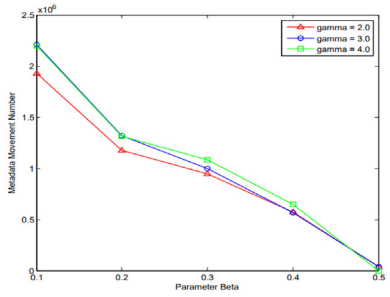
(c) The relation among β, γ and the number of metadata movement when the network size is not subject to quick changes.



(d) The relation among β, γ and the ratio of the heaviest cluster load and average cluster load when the network size is decreasing quickly.



(e) The relation among β, γ and the standard deviation of the cluster load when the network size is decreasing quickly.



(f) The relation among β, γ and the number of metadata movement when the network size is decreasing quickly.

Fig. 3. Relation among parameters β, γ and the effectiveness of the algorithm for the inter-cluster load balancing (I).

According to the above simulation results, we can conclude that the inter-cluster balancing algorithm can not only decrease the cluster load with heavy load drastically, but also improve the degree of the cluster load balancing in the whole MHP2P upper layer network.

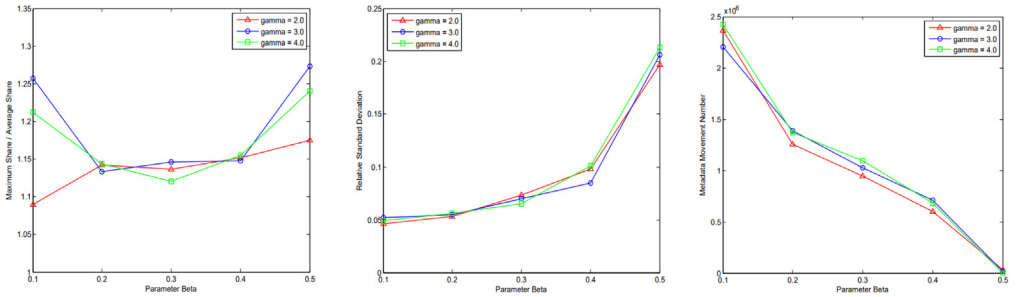
5.2. Simulation of intra-cluster load balancing algorithm

In the simulation of the intra-cluster load balancing algorithm, we mainly calculate the ratio of the highest load rate to average load rate as well as the relative standard deviation of the load rate. The defined ratio is used to analyze whether the algorithm can relieve the node with the high load rate while the relative standard deviation of the load rate is used to test whether the algorithm can improve the degree of the load balancing in a cluster. Since there are many clusters in the MHP2P network, the results of the ratio and relative standard deviation are the average values of all clusters.

5.2.1. Simulations under different network sizes

In this simulation, different network sizes from 2^{14} to 2^{20} are used to determine whether the algorithm is effective.

In Fig. 5(a), an MHP2P network with the intra-cluster load balancing algorithm can keep the load rate of a node with the highest load rate about twice of the average load rate, while the value of MHP2P network without the intra-cluster load balancing algorithm can be more than 3 times in the worst case scenario. Fig. 5(b) shows that the MHP2P network with the algorithm can

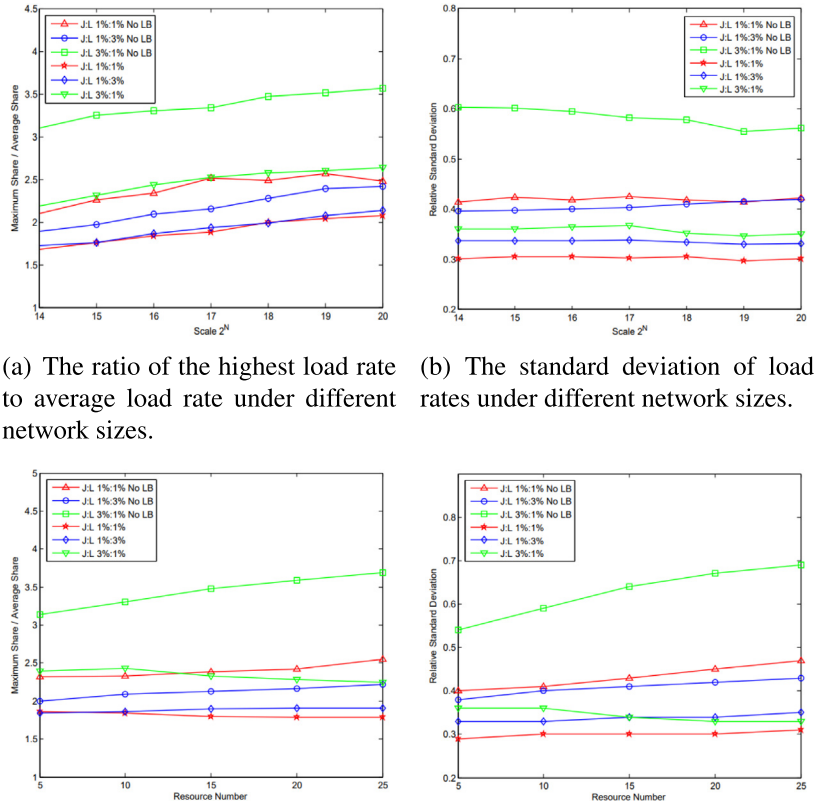


(a) The relation among β, γ and the ratio of the heaviest cluster load and average cluster load when the network size is increasing quickly.

(b) The relation among β, γ and the standard deviation of the cluster load when the network size is increasing quickly.

(c) The relation among β, γ and the number of metadata movement when the network size is increasing quickly.

Fig. 4. Relation among parameters β, γ and the effectiveness of the algorithm for the inter-cluster load balancing (II).



(a) The ratio of the highest load rate to average load rate under different network sizes.

(b) The standard deviation of load rates under different network sizes.

(c) The ratio of the highest load rate and average load rate under different numbers of metadata items.

(d) The standard deviation of load rates under different numbers of metadata items.

Fig. 5. Simulations for the intra-cluster load balancing under different network sizes and under different numbers of metadata.

also improve the degree of load balancing among the nodes in the same cluster. The largest network size in our simulation is in millions in order to indicate the scalability of the intra-cluster load balancing algorithm.

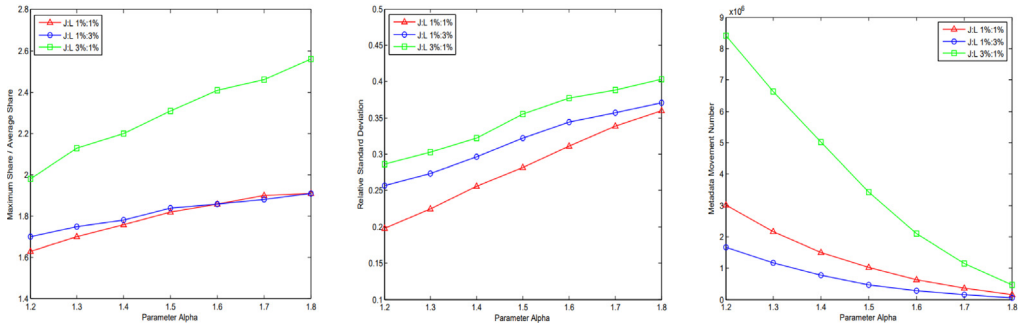
5.2.2. Simulations under different numbers of metadata items

In a cluster, different numbers of metadata items cause different load imbalance problems. Hence, we simulate the intra-cluster load balancing algorithm under different numbers of metadata items to indicate if the algorithm has any effect on the MHP2P network. In this simulation, the average number of metadata items varies from 5 to 25.

Figs. 5(c) and 5(d) show that the intra-cluster load balancing algorithm can work well under different numbers of metadata items. The algorithm can keep the ratio of the highest load rate to average load rate below about 2.5, and the relative standard deviation is much lower than the result without it.

5.2.3. Simulations under different settings of parameter α

Since the value of α determines the proportion of the nodes with high load rate and the proportion of those with low load



(a) The relation between α and the ratio of the highest load rate to the average load rate.

(b) The relation between α and the standard deviation of the load rate.

(c) The relation between α and the number of metadata movements.

Fig. 6. Relation between parameter α and the effectiveness of the algorithm for the intra-cluster load balancing.

rate, α has a big effect on the result of the intra-cluster load balancing algorithm. If it is too high, the degree of the load balancing in a cluster is low; while if it is too low, the metadata movement is too much. Hence, we do this simulation to optimize the setting of α. In the simulation, α varies from 1.2 to 1.8.

Figs. 6(a) and 6(b) show that the ratio of nodes with the highest load rate to the average load rate and standard deviation of load rate increase as α increases. In Fig. 6(c), metadata movements decrease as α increases and this deceleration slows. Further, when the network size is increasing, metadata movement is much more than that when the network size is not increasing quickly. This is because the growth of the network size leads to the growth of metadata, and a node also needs to assign some metadata when it joins a cluster.

Fig. 6(b) shows that when α is 1.2, the relative standard deviation is still greater than 0.2. Hence, if α is set to 1.2, it may cause the problem of too many nodes with high load rate. In addition, this also increases the burden on the supernode of a cluster. α should hence be set in interval [1.4,1.7], and can be selected according to the requirement for the degree of the intra-cluster balance.

The result of the simulation shows that the intra-cluster load balancing algorithm can also make the node with high load rate decrease its load rate by transferring some metadata to some nodes with low load rate. It improves the degree of the intra-cluster load balancing greatly.

6. Conclusion

This paper presents a three-layer mobile hybrid hierarchical P2P model called MHP2P as a cloudlet in MEC systems. An MEC server in the cloudlet can be any device willing to offer service. MHP2P uses a Chord ring as the upper layer, clusters as the middle one and mobile devices as the lower one. DHT and flooding methods employed in our model make MHP2P have high stability, scalability and efficiency. More importantly, inter-cluster and intra-cluster load balancing schemes are provided to solve the problem of load imbalance in MHP2P. A large number of experimental simulations are conducted and the results show that the proposed schemes can significantly improve the degree of load balancing even when the model size is in millions. In the future, we will further investigate more mobile end-user based services such as gaming and VR as well as security problems with MHP2P model.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Anon, 2009. PeerSim, http://sourceforge.net/projects/peersim/.
Anon, 2014. ETSI, Mobile-edge computing introductory technical white paper, White Paper, Mobile-edge Computing Industry Initiative. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edgecomputing-introductorytechnicalwhitepaperv1.
Bharati, N., Das, S., Gourisaria, M.K., 2021. A review on mobile cloud computing. In: Intelligent and Cloud Computing. Springer, pp. 209–218.
Cai, W., Leung, V.C., Chen, M., 2013. Next generation mobile cloud gaming. In: 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering. IEEE, pp. 551–560.
Cho, Y., Paek, Y., Ahmed, E., Ko, K., 2016. A survey and design of a scalable mobile edge cloud platform for the smart IoT devices and it's applications. In: Advances in Computer Science and Ubiquitous Computing. Springer, pp. 694–698.
Cui, M., Fei, Y., Liu, Y., 2021. A survey on secure deployment of mobile services in edge computing. Secur. Commun. Netw. 2021, 1–8.
Duan, Z., Tian, C., Zhou, M., Wang, X., Zhang, N., Du, H., Wang, L., 2017. Two-layer hybrid peer-to-peer networks. Peer Peer Netw. Appl. 10, 1304–1322.
Fernando, N., Loke, S.W., Rahayu, W., 2013. Mobile cloud computing: A survey. Future Gener. Comput. Syst. 29 (1), 84–106.
Hameed, S.A., Nirabi, A., Habaebi, M.H., Haddad, A., 2019. Application of mobile cloud computing in emergency health care. Bull. Electr. Eng. Inf. 8 (3), 1088–1095.
Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D., 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. ACM, pp. 654–663.
Lee, S., Lee, S., Shin, M.-K., 2019. Low cost MEC server placement and association in 5G networks. In: 2019 International Conference on Information and Communication Technology Convergence, ICTC. IEEE, pp. 879–882.
Li, H., Li, D., Wong, W.E., Zeng, D., Zhao, M., 2021. Kubernetes virtual warehouse placement based on reinforcement learning. Int. J. Perform. Eng. 17 (7), 579–588.
Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: The communication perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.
Mehrabi, M., Salah, H., Fitzek, F.H., 2019. A survey on mobility management for MEC-enabled systems. In: 2019 IEEE 2nd 5G World Forum, 5GWF. IEEE, pp. 259–263.
Noor, T.H., Zeadally, S., Alfazi, A., Sheng, Q.Z., 2018. Mobile cloud computing: Challenges and future research directions. J. Netw. Comput. Appl. 115, 70–85.
Qiao, X., Luo, L., Yang, J., Hu, Z., 2020. Intelligent recommendation method of sous-vide cooking dishes correlation analysis based on association rules mining. Int. J. Perform. Eng. 16 (9), 1443–1450.
Ren, J., Zhang, D., He, S., Zhang, Y., Li, T., 2019. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Comput. Surv. 52 (6), 1–36.
Rimale, Z., Benlahmar, E., Tragha, A., El Guemmat, K., 2016. Survey on the use of the mobile learning based on mobile cloud computing.. Int. J. Interact. Mob. Technol. 10 (3), 35–41.
Roman, R., Lopez, J., Mambo, M., 2018. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Gener. Comput. Syst. 78, 680–698.
Satria, D., Park, D., Jo, M., 2017. Recovery for overloaded mobile edge computing. Future Gener. Comput. Syst. 70, 138–147.

Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N., 2009. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. 8 (4), 14–23.
Steiner, M., En-Najjary, T., Biersack, E.W., 2007. A global view of kad. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement. ACM, pp. 117–122.
Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H., 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '01, pp. 149–160.
Voulgaris, S., Jelasity, M., Van Steen, M., 2003. A robust and scalable peer-to-peer gossiping protocol. In: International Workshop on Agents and P2P Computing. Springer, pp. 47–58.
Zhang, W.-Z., Elgendy, I.A., Hammad, M., Iliyasu, A.M., Du, X., Guizani, M., Abd El-Latif, A.A., 2020. Secure and optimized load balancing for multitier IoT and edge-cloud computing systems. IEEE Internet Things J. 8 (10), 8119–8132.
Zhang, K., Leng, S., He, Y., Maharjan, S., Zhang, Y., 2018. Mobile edge computing and networking for green and low-latency internet of things. IEEE Commun. Mag. 56 (5), 39–45.