Future Generation Computer Systems 90 (2019) 39–48

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

An advanced decision model enabling two-way initiative offloading in edge computing

Zhida Yin [a], Haopeng Chen [a,*], Fei Hu [b]

[a] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
[b] China Aeronautical Radio Electronics Research Institute, China

HIGHLIGHTS

- We utilize the inborn hierarchical structure of the Internet.
- We admit that information can be stale and congestion can take place unexpectedly.
- RED algorithm is modified to fit the context of edge computing and integrated.
- We enable two-way initiative offloading to avoid congestion.

ARTICLE INFO

Article history:
Received 25 December 2017
Received in revised form 25 April 2018
Accepted 17 July 2018
Available online 27 July 2018

Keywords:
Edge computing
Offloading problem
Congestion avoidance
RED algorithm integrated

ABSTRACT

Edge computing is envisioned as a promising enabler to leverage computation capacities at the edge and address the issues faced in cloud computing. In this paper, we propose an advanced decision model to solve the computation offloading problem in edge computing. It utilizes the intrinsic hierarchical topology of the Internet and performs online scheduling in a decentralized manner to eliminate anticipated modeling. It also admits the fact that congestion can take place unexpectedly, for the possible reasons such as stale information. Our decision model is enhanced in two approaches to endow both sender and receiver (for one offloading action) with the ability of "dropping" the request, so as to avoid congestion. Random Early Detection(RED) algorithm is incorporated. The results of simulation demonstrate that our decision model can handle workloads well and these two enhancements are effective.

1. Introduction

The maturity of cloud computing has tremendously influenced our lives in various aspects. Famous software as a service(SaaS) instances, such as Google applications, already have daily active users(DAU) on the scale of millions or even billions. Meanwhile, diverse infrastructures aiming at providing scalability, elasticity, and autonomy, such as Amazon EC2 and Microsoft Azure, have been developed to support these services. Our previous work [1] and its enhanced version [2] also offered some suggestions towards this topic.

In recent years, the proliferation of Internet of Things(IoT), cloudlets and mobile phones is gradually making changes. In traditional cloud computing, there are clear boundaries between data consumers and data producers; while nowadays these IoT devices can work as not only data producers but also data consumers [3]. Based on this role transition, the number of data producers surges, as well as the raw data produced. Even though the processing capacity of central servers keep developing, the growing quantity of data and the limited bandwidth of the network is becoming the bottleneck for the cloud-based computing [3].

Moreover, services, especially mobile ones, are becoming more and more sensitive to latency and require immediate response [4]. Offloading workloads to central servers, as we do in traditional cloud computing, will inevitably incur long latency due to network transmission [5] and thus have major impact on performance. Additionally, other considerations, such as privacy protection requirement, are also attracting more and more concerns.

Such trends have pushed the horizon of a new computing paradigm, edge computing. In edge computing, data should be handled in close proximity to where it is produced, namely at the edge. Since this paradigm can save network bandwidth and take full advantage of processing capacity at the edge, it is viewed as a promising enabler. However, there are still several issues to be addressed. Among them, the computation offloading problem, which concerns the trade-off between energy consumption and service latency, has drawn wide attention.

* Corresponding author.
E-mail addresses: darkness_y@sjtu.edu.cn (Z. Yin), chen-hp@sjtu.edu.cn (H. Chen), hu_fei@careri.com (F. Hu).

https://doi.org/10.1016/j.future.2018.07.031
0167-739X/

Recent research efforts focused on proposing some mixed frameworks, which leverage both central servers and edge servers [6,7], to deal with the computation offloading problem. Others like [8] proposed a messaging based modular gateway platform to enable clustering gateways and the abstraction of peripheral communication protocol details. However, most of them inappropriately estimated edge servers as black boxes, which means that once the offloading decision is made, the selected server can, by all means, work as expected and congestions will never take place. In other words, these mixed frameworks make the decision in a static manner. Furthermore, they also make the decision in a flat manner among central servers and edge servers, which neglect the intrinsic hierarchical structure of the network from edge servers to central servers [3]. Although some researchers exploited the hierarchical structure, they did an intentional organization and its workload placement was an offline job [9].

In this paper, we solve the computation offloading problem by organizing edge servers and central servers in its intrinsic hierarchical structure, maintaining task queues on each server respectively and doing online scheduling of workloads. Our basic idea is to endow both sender and receiver (for one offloading action) with the ability of "dropping" the request for the purpose of congestion avoidance. We integrate RED [10], which is a queueing discipline for a network scheduler, into an enhanced decision-making procedure so that the utilization of resources is improved and the whole framework remains lightweight. The results of simulation demonstrate that our method can handle workloads well, even when serving peak loads or skewed loads (caused by the topology or the workloads itself).

The main contributions of this paper can be summarized as follows:

- We conclude the weakness of existing solutions on computation offloading and correspondingly propose a lightweight decision model which utilizes the intrinsic hierarchy of the network, tolerates stale information and enables two-way initiative offloading in an online manner.
- We analyze the opportunity for RED algorithm in details and migrate it into our decision model after appropriate modifications.
- We present a comprehensive evaluation of our decision model from various aspects to validate its effectiveness and robustness.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 introduces the system model. Section 4 details our advanced decision model and its implementation. Section 5 presents the evaluation results. Section 6 draws the conclusion and discusses future work.

2. Related work

Extensive research studies in the computation offloading problem have been done. Academically, the problem can be decomposed into a set of decisions to be made: when the workloads are generated (usually in the form of raw data), the user equipment(UE), such as an IoT device, should decide whether to handle it in local or to offload it to some remote server; once deciding to offload, UE should select one appropriate server among several accessible edge/central servers as the receiver. Furthermore, UE can choose between full offloading and partial offloading, according to concerns about privacy and dependency [11].

In early years, researchers were looking for some adaptive approaches. Xian et al. [12] avoided prediction by introducing a timeout which was initially 2-competitive and finally statistically optimal. Gonzalo et al. [13] presented an adaptable mechanism based on the analysis of execution history of application to perform adaptive offloading.

In recent years, most of the decisions [6,7,14,15] are made based on the trade-off between energy to be consumed and latency to be caused [11]. Xu et al. [6] formulated the decision-making problem as a decentralized computation offloading game and proposed a game theoretic approach to achieve efficient offloading. Zhang et al. [7] designed an energy-efficient computation offloading(EECO) scheme to jointly optimize offloading and radio resource allocation, incorporating the multi-access characteristics of the 5G heterogeneous network. Yang et al. [14] argued that the connection between UE and its accessible remote servers, such as cloudlets, can be intermittent, thus they formulated and solved a Markov decision process(MDP) model to obtain an optimal policy to minimize the computation and offloading costs. Rahimi et al. [15] introduced a hybrid, tiered cloud architecture called MAPCloud, intending to leverage both local and public clouds to increase performance and scalability of mobile applications.

As mentioned in Section 1, although these strategies considered different characteristics of mobile applications and edge computing, they were making the decision in a static and flat manner. More emphases are laid on scheduling, edge/central servers are inappropriately estimated as black boxes and the possibility of the existence of unexpected congestion is neglected. Meanwhile, they merely estimated "how far is the server" according to the transmission time and thus ignored the intrinsic hierarchical structure of the network [3].

Mike et al. [16] tried to remedy the defects by redirecting tasks based on a flow network and looking for an optimal placement of workloads. However, they assumed that all cloudlets were connected with each other, thus the overall flat structure remained. Liang et al. [9] made their attempt by proposing an analytical model to show that the hierarchical structure has better efficiency serving the peak loads and then developing algorithms to adaptively place workloads among different tiers of servers. However, they did a deliberate organization, instead of utilizing the intrinsic hierarchy. Additionally, these strategies shared a common flaw: the workloads placement became an offline job. Thus anticipated modeling and its resulting overhead are inevitable.

From other aspects, Chen et al. [17] extended the single-UE scenario to the multi-UEs scenario and the overall complexity of the proposed solution is $O(N^6)$, which could be too high for a great number of UEs connected. You et al. [18] studied the energy-efficient resource-management policy for asynchronous computation offloading system, where UEs have heterogeneous input data arrival time instants and computation deadlines, formulated the optimization problem and showed two optimal policy for data partitioning (for offloading and local computing) and time division (for transmissions). The possibility of the existence of unexpected congestion is still neglected.

When facing the issue of probable congestion, Shermila et al. [19] adopted game theory, formulated the problem as a Minority Game (MG), and applied reinforcement learning to solve the game. Through this approach, the issue of possible congestion is considered from the perspective of resource allocation instead of computation offloading [11], and centralized dispatching is required. Other researches such as [20,21] also turned to reinforcement learning and game theory to solve other issues like exploiting harvesting edge computing, quality of services(QoS) to users and cutting monetary cost. Unfortunately, these models take time to converge and the process of warming up will cause overhead.

3. System model

In this section, we will introduce the system model from two aspects: the intrinsic hierarchical structure and the primitive decision model.

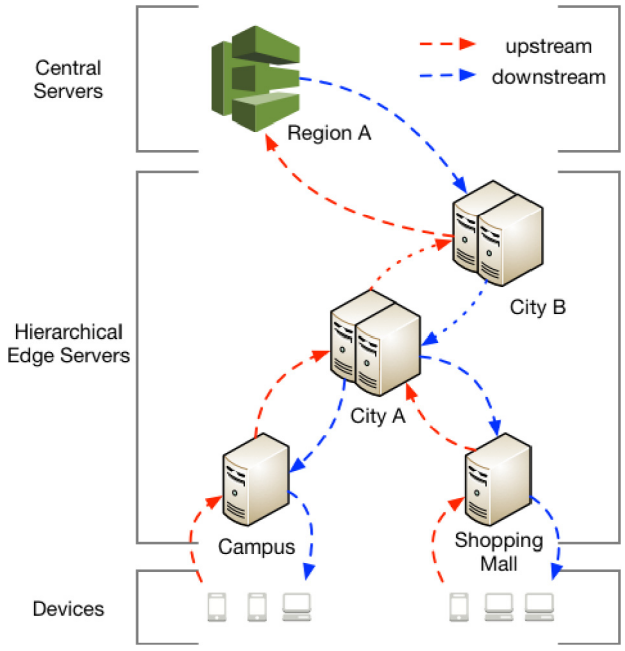


Fig. 1. Edge computing paradigm.

3.1. The hierarchical structure

As described in Section 1, edge computing aims at handling data in the close proximity to where it is produced so as to reduce service latency, utilize all-tiers' resources and save network bandwidth. By some restricted definition, the whole edge system is constrained within a limited geographical area. However, as analyzed in [3,9], the edge system can be extended and generalized. There are two computing streams in edge computing: one works as upstream from edge servers to central servers; the other one works as downstream from central servers to edge servers. "Edge servers" are regarded as any computing and network resources along the streams. Fig. 1 demonstrates such hierarchical structure. Users' devices connect into the Internet via edge servers at the bottom tier and can finally reach central servers after several hops. The connection between UE and edge server at the bottom tier can be either wired or wireless, while the connection between servers at two adjacent tiers is usually through the Internet backbone. It is worth noting that any node in the system, including both users' devices and servers, can somehow play a role in both consuming data and producing data, resulting from the development of IoT.

Based on such hierarchical structure, each request that is sent to the central server will actually go through several edge servers (including the producer itself) which are equipped with computing resources before it can finally reach the central server. Each edge server that the request goes through can be viewed as one hop the request takes. Then, aforementioned "handling data in the close proximity" can refer to handling data within as few hops as possible. Meanwhile, the computation offloading problem can be transformed into decentralized decision-making problems on each server separately: once a request arrives, whoever produced it, the current server should decide whether to handle it in local (if possible) or continue to transmit it to its "next-hop" server. There exists a greedy context with tradeoffs in practice. Abstract models can be extracted.

Since each server can make their own decision independently in real time, the decentralized decision-making problems analyzed above can eliminate preprocessing of workloads and do online scheduling of these requests. In the meantime, since the intrinsic hierarchical structure is exploited, the decision will no longer be made in a flat manner. Furthermore, as proved in [9], the hierarchical structure has better efficiency serving the peak loads.

3.2. The primitive decision model

For ease of exposition, we first consider the primitive decision model, which is a common decision model used in the computation offloading problem [6,15]. It can be applied when one server is providing service for only one arriving request. This server can decide whether to handle this request in local or offload it to the next-hop server without the influence of other requests, which implies that there will never be congestion on related servers. The existence of congestion will be discussed in Section 4.

As described in Section 1, we assume that the requests are computationally intensive and delay sensitive. They can be computed either locally on the current server or remotely on the next-hop server. Each request can be formulated as $I_i \triangleq \{B_i, D_i\}$. Here B_i denotes the size of computation input data (e.g., the input parameters and programming codes) involved in this request and D_i denotes the total number of CPU cycles required to accomplish this request. This information can be obtained through similar methods as used in [22]. Meanwhile, we consider a set of servers $\mathcal{N} = \{1, 2, \ldots, N\}$. They locate at separate tiers throughout the entire hierarchical structure. Each server is equipped with the different capacity of computation which can be quantified as F_i in terms of CPU cycles per second. Overhead in terms of both energy and time consumption should be taken into consideration.

3.2.1. Communication model

We first introduce the communication model, which is closely related to time spent on transmission. As analyzed in Section 3.1, the connections in the system can be classified into two types: wired connection and wireless connection. The former one can probably be found between servers locating at two adjacent tiers; the latter one can be found between users' devices and edge servers locating at the bottom tier.

To these wired connections, they are usually provided with relatively stable bandwidth for offloading which can be denoted as $Wired_i$.

To these wireless connections, the occupancy of different channels should be further considered. We use $d_i \in \{0, 1\}$ to describe the decision server i has made: 0 for handling this request in local, 1 for offloading this request to its next-hop server. Then the offloading data rate can be computed according to [23] as

$$Wireless_i = W \log_2\left(1 + \frac{P_i H_i}{\omega_i + \sum_{m \in \mathcal{N} \setminus \{i\}: d_i = 1} P_m H_m}\right) \tag{1}$$

where W is the channel bandwidth, P_i is the transmission power H_i is the channel gained by server i, and ω_i is the background interference power.

Thus, the communication cost can be concluded as

$$W_i = \begin{cases} Wired_i & \text{under wired connection} \\ Wireless_i & \text{under wireless connection} \end{cases} \tag{2}$$

3.2.2. Handling in local

We then discuss the overhead caused by local handling the request. With this choice, the current server decides to handle the request in local.

The computation execution time mainly contributes to the overhead of time consumption, which can be estimated according to

$$T_i = \frac{D_i}{F_i} \tag{3}$$

Then the overhead of energy consumption can be estimated according to

$$E_i = \nu_i D_i \tag{4}$$

where ν_i is the coefficient of energy consumption per CPU cycle. According to some realistic measurements [24], $10^{-11}(F_i)^2$ can be used to estimate ν_i.

Based on Eq. (3) (4), we can compute the overhead caused by local handling as

$$Overhead_{i,local} = \gamma_t \frac{D_i}{F_i} + \gamma_e 10^{-11}(F_i)^2 D_i \tag{5}$$

where γ_t, γ_e ($\gamma_t + \gamma_e = 1$ and $0 \leq \gamma_t, \gamma_e \leq 1$) are the weights of time and energy consumption respectively. Different weights can be configured on separate servers and these weights can be adjusted at runtime, so that flexibility is provided. Devices and server at lower tiers are usually more sensitive to energy consumption since they may be using battery.

3.2.3. Offloading request

We then discuss the overhead caused by offloading the request. With this choice, the current server decides to offload the request to its next-hop server.

The overhead of time consumption consists of transmission time and potential computation execution time. It can be estimated according to

$$T_i = \frac{B_i}{W_i} + \frac{D_i}{F_j} \tag{6}$$

Here the first half of the expression indicates the transmission time spent on offloading the request to its next-hop server; while the second half of the expression indicates the potential computation execution time spent on handling the request on that server. Even though this request might be further offloaded to other servers, the situation will not be worse since further offloading cannot happen unless another decision is made. So we use potential computation execution time to do the current judgment directly.

Then the overhead of energy consumption can be estimated according to

$$E_i = P_i \frac{B_i}{W_i} \tag{7}$$

where P_i is the coefficient of energy consumption.

Based on Eq. (6) (7), we can compute the overhead caused by offloading as

$$Overhead_{i,offload} = \gamma_t\left(\frac{B_i}{W_i} + \frac{D_i}{F_j}\right) + \gamma_e P_i \frac{B_i}{W_i} \tag{8}$$

where γ_t, γ_e ($\gamma_t + \gamma_e = 1$ and $0 \leq \gamma_t, \gamma_e \leq 1$) are the weights of time and energy consumption respectively. We can also apply different weights to different servers to provide flexibility.

Similar to many studies such as [9,6,15], we also neglect the time overhead for sending back the outcomes, because of the fact that the size of outcomes is generally very small compared with the input data.

3.2.4. The decision procedure

Due to the assumption of the primitive decision model, which is that one server is providing service for only one arriving request, we can directly make the decision according to the comparison between two choices' overhead. If the overhead of local handling is lower, then the current server will choose it, and vice versa, which can be formulated as

$$d_i = \begin{cases} 0 & Overhead_{i,upload} \geq Overhead_{i,local} \\ 1 & \text{o.w.} \end{cases} \tag{9}$$

Here 0 indicates local handling and 1 indicates offloading.

Since congestion will never take place under our assumption, the primitive decision model can always work as expected.

4. The advanced decision model

In this section, we develop an advanced decision model based on the hierarchical structure and the primitive decision model. We abandon the aforementioned assumption and admit the probable existence of congestion.

4.1. The enhanced decision model

According to the analysis in Section 3.1, when one request is being sent, it will actually go through several servers and form a path from its original server to the central server. One request will generate one path; a great number of requests will generate a great number of distinct paths. Due to the hierarchical structure, these paths may intersect, overlap, or converge. Then edge servers which locate at the joint points may be overloaded, form task queues, or even get congested because of the influx of requests. For example, as in Fig. 1, the edge server in the shopping mall will get congested if all the devices below decide to offload their requests to it. Because servers at higher tiers are more likely to be the joint points of more paths, congestion can also happen at higher tiers regardless of the fact that these servers are usually more powerful. Some traditional solutions assume the edge servers to be black boxes so that they can handle all requests as expected (looks like our primitive decision model). Unfortunately, congestion can take place unexpectedly in reality. Other solutions apply some global placement strategies to schedule these requests. But, as a result, the scheduling becomes an offline job and anticipated modeling becomes inevitable.

One straightforward solution to avoid congestion is to additionally take the status of the next-hop server into consideration when evaluating the overhead of time consumption for offloading, so that the sender can make initiative decision. Namely, Eq. (6) should be modified as

$$T_i = \frac{B_i}{W_i} + \frac{D_i + \sum D_j}{F_j} \tag{10}$$

where $\sum D_j$ is the total amount of CPU cycles required to complete all enqueued requests on the next-hop server.

By this little enhancement, we can avert pushing requests to a next-hop server which is already overloaded.

4.2. The opportunity for RED

Although the enhanced decision model can effectively prevent servers from pushing requests to other overloaded servers, it is not enough. In the example given in Section 4.1, the edge server in the shopping mall can still get congested even if it was totally idle when the devices below were independently making their decisions. This is because the status information these devices fetched can be stale and congestion can take place right after the decision is made.

We further study the congestion on edge servers during offloading and find out that it shares some common characteristics with the congestion on a router at the network layer in the Open Systems Interconnection (OSI) model [25]. The request which is going to be offloaded is just like a datagram but much larger in size. We assume that there is only one worker thread on the server. When the server is serving one request (e.g., making decision, executing computation, or offloading request), other requests have to be enqueued. If the queue becomes too long, the requests at the tail of the queue have to wait for a long time before it could be served, even if it will later be offloaded at that time. Such situation will lead to a significant increase in latency. If things go worse, that is the server no longer has enough space to buffer more requests, the server will be obliged to refuse following requests. Such situation

will result in congestion. Both of these two situations will greatly vitiate user experience. To be more straightforward, for some of the requests, especially those at the tail of the queue, waiting in the queue can be a waste of time.

The deeper reason lies in the fact that even though the enhanced decision model enables the current server, which works as a sender, to make initiative decision according to the receiver's status, it also forces the next-hop server, which works as a receiver, to passively accept the request. Unfortunately, neither the primitive decision model nor the enhanced one can make the receiver be conscious of whether it is already overloaded, because it will not touch one request until the request finally becomes the head of the queue (maybe after a long time waiting). In other words, when the server is accepting requests, it makes no differences whether there is a queue or not. Thus, the server may accept requests beyond its ability.

The constraint is the case that, in order to maintain the scheduling to be an online job, each server should only make its independent decision, along with the ineluctable possibility that the status information can be stale due to the transmission time. The idea of qjump [26] inspired us, "Queues don't matter if you can jump them". Now that waiting in the queue can be a complete waste of time for some requests, let them jump the queue directly and actively! As analyzed before, the congestion on edge servers during offloading shares common characteristics with the congestion on a router at the network layer, then RED becomes a promising candidate.

Random Early Detection (RED) [10], also known as Random Early Discard or Random Early Drop, is a famous queueing discipline for a network scheduler suited for congestion avoidance. Only enqueued datagrams are going to be processed. RED monitors the queue size by a weighted moving average model and drops datagrams based on statistical probabilities. When the buffer is empty, it will accept all incoming datagrams. As the queue grows, it will drop incoming datagrams according to a growing probability. When the queue is finally full, the probability of dropping will reach 1 and all incoming datagrams are dropped.

Within the context of offloading, "dropping the request" indicates offloading it to its next-hop server, while "processing the request" indicates making the decision upon it. Dropping the request when the current server is already overloaded can help the request to jump the queue directly. This also enables the server, which works as the receiver, to selectively accept the request according to its ability. It is true that such active dropping may selfishly cast the burden on the next-hop server. Nevertheless, servers at higher tier are usually more powerful. When both the current server and the next-hop server are under the burden, the latter one should bear more responsibility. It does not breach the original intention of edge computing.

It is also noteworthy that Jacobson et al. argued that RED has two bugs [27]. Some efforts, such as [28,29], have been done to improve the original algorithm. However, within the context of offloading, enabling requests to jump the queue is the key idea. Those improvements are not necessary. Instead, we did some other modifications upon it, so that it could fit the context better:

Firstly, the evaluation of "the queue size" is altered into calculating the expected time to finish processing all enqueued requests. There are three considerations:

- The original version of "queue size" will make the decision of dropping sensitive to the amount of required computation of each request. For instance, the current server is executing one request that requires a great amount of computation and the execution will last for a while. Then if another request arrives, a better choice is to jump the queue if possible. But since there is only one enqueued request, "queue size" is literally very small and such jumping is not triggered. As a contrast, the altered version of "queue size" can figure out such situation and a proper decision can be achieved.
- When making the decision of dropping, the receiver can hardly foretell what it will do upon these enqueued requests. Some of them might be offloaded later, while others might be executed in local. Despite such diversity, it is effective and infallible for RED to assume that all these requests will be executed in local, in which case "queue size" is evaluated as in the worst case. It shares the philosophy adopted during the evaluation of the potential computation execution time in Section 3.2.3.
- The new version of "queue size" can be set independently on different server, according to its Service Level Agreement(SLA).

Secondly, min_{th} and max_{th} should be correspondingly modified. In order to handle requests with both large and small amount of required computation, we use

$$\begin{cases} min_{modified} & = min\{1.0, t_{exe,avg}\} * min_{original} \\ max_{modified} & = max\{1.0, t_{exe,avg}\} * max_{original} \end{cases} \tag{11}$$

Here $t_{exe,avg}$ is the average execution time of these enqueued requests. For the lower bound, when enqueued requests require relatively large amount of computation, the threshold represents the absolute execution time; when enqueued requests require relatively small amount of computation, the threshold represents the expected execution time for $min_{original}$ requests. For the upper bound, we choose to extend the range for calculating the growing probability as wide as possible, so that the current server will not totally "deny" requests too early and the resource can be fully utilized.

Thirdly, some other key parameters should also be modified, such as w_q and max_p. [10] provided some suggestions on setting these parameters.

Obviously, the offloading done by the enhanced decision model lays more emphasis on the status of the next-hop server, while the offloading done by RED lays more emphasis on the status of the current server. These two enhancements complement each other well.

4.3. The integrated result

Based on the enhancements introduced in Sections 4.1 and 4.2, the status of both sender and receiver are taken into account so as to avoid congestion during offloading. As analyzed before, the enhanced decision model is in charge of the requests which are enqueued, while RED aims at dropping requests when the current server is overloaded. Thus, these two enhancements do not interfere with each other and can be applied at different points respectively.

We integrate them together so that they can cooperate harmoniously:

1. We implement a RED Filter based on the RED algorithm and our modifications described above. It works as the entrance of each server. The RED filter will collect the queue information(*info*) from the task queue maintained by the current server and use a weighted moving average model to evaluate the possibility of offloading. Requests to be "dropped" will be offloaded before it is enqueued.
2. Each enqueued request will be processed when it becomes the head of the task queue. Another filter named Overhead Filter will be there to decide whether the request should be handled in local or offloaded to its next-hop server. Overhead Filter will collect the queue information(*info*) from the task queue maintained by the next-hop server and adopt the advanced decision model.

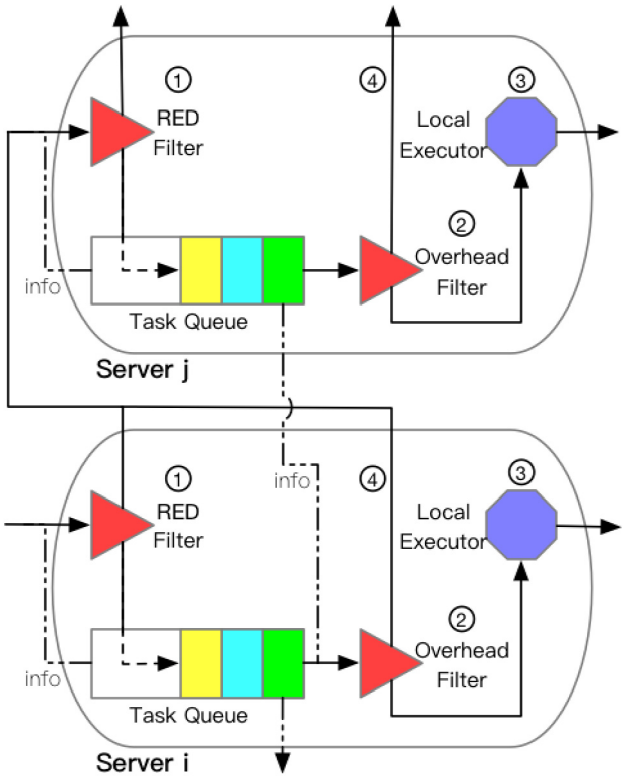


Fig. 2. The advanced decision model.

3. If the request is scheduled to be handled in local, it will be forwarded to the Local Executor. The executor will execute the computation and then send the result back.
4. If the request is to be offloaded, it will be transmitted to the entrance of its next-hop server. The request will be further judged by the RED Filter there, together with other requests, including the one that is also offloaded to that server as well as the one that is produced by that server itself.

Fig. 2 demonstrates the whole procedure described above. We integrate RED algorithm into the enhanced decision model and finally get the advanced decision model. Following the procedures, each request will be ultimately offloaded to a suitable server and executed, according to time consumption, energy consumption and status of the servers along the path of offloading. As the figure illustrated, the advanced decision model also succeeds in abstracting a uniform decision model, which is adoptable for servers at any tier. Despite the complicated topology of the network in real life, the scalability of the whole system is maintained since the decisions are made in a decentralized and independent manner. Meanwhile, scheduling is kept to be an online job, so that anticipated modeling and its resulting overhead can be averted. Furthermore, the whole system remains virtually stateless, which obeys "Keep it simple, stupid"(KISS) principle[1] and improves the robustness. Routing protocols can also be helpful to paths' reestablishment when some servers crash.

5. Evaluation

In this section, we implement the advanced decision model and evaluate the performance by conducting simulation experiments.

[1] https://en.wikipedia.org/wiki/KISS_principle.

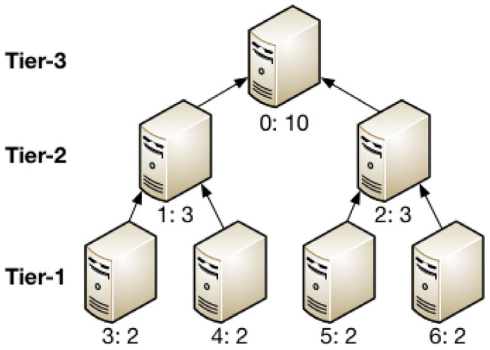


Fig. 3. Simple hierarchical topology.

We first apply the advanced decision model upon a strictly symmetrical hierarchy topology and compare its performance with other decision models. Then, we evaluate our model with skewed test cases, including skewed requests and skewed topology.

5.1. Simulation settings

As demonstrated in Fig. 3, we create a strictly symmetrical hierarchy topology. This simple structure can ease the analysis of results. Meanwhile, according to the traced-based simulation conducted in [9], such 3-tier edge cloud is also recommended, compared with two other simple hierarchical structures. The numbers labeled beneath each server indicates its index number(id) and available computation correspondingly. We provide much more computation to the server at tier-3 by design, because the central server is usually much more powerful than edge server. The ratio of γ_t to γ_e, which are weights for time consumption and energy consumption respectively, is set to $Comp_{available}$: 1, because powerful servers usually locate at high tier and seldom worry about the lack of energy. The bandwidths of these connections are set to 100 Mbps.

One task generator is responsible for pushing requests to the servers at tier-1 periodically. The number of requests per period, which indicates the workload rate, is determined following a Poisson distribution. The required computation of each request, which indicates the request size, is further determined following a Normal distribution.

Without loss of generality, we select the average completion time of requests as the metric of performance as [9] did. The smaller average completion time indicates the larger amount of computation capacity provided by the whole system. Meanwhile, we also record the number of requests that each server has handled for auxiliary analyses.

We validate the performance of edge computing by comparing the advanced decision model with pure cloud computing and pure local handling. In pure cloud computing, each request is offloaded to the central server, as we do in daily life; while in pure local handling, the producer of the data will handle the request in local by itself. We also validate the effectiveness of the aforementioned enhancements by comparing the integrated version with two incomplete versions. In following description, the 1st enhancement indicates the one introduced in Section 4.1; the 2nd enhancement indicates the one introduced in Section 4.2. To sum up, we evaluate the advanced decision model with four controlled experiments.

5.2. Simulation results

5.2.1. The impact of workload rate and request size

Two sets of experiments are conducted. Firstly, we fix $\lambda = 8$ for the Poisson distribution and change μ for the Normal distribution from 1 to 9 (σ^2 is fixed as 1). Fig. 4 demonstrates the

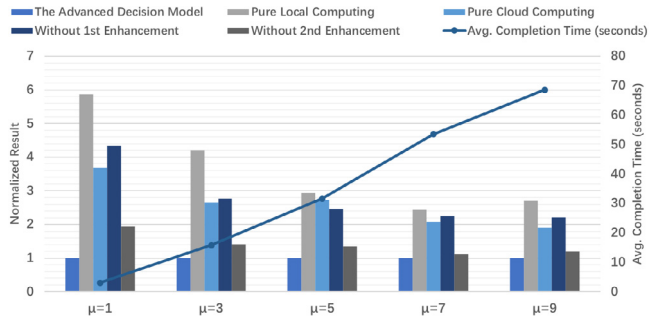


Fig. 4. Normalized average completion time over different request sizes.

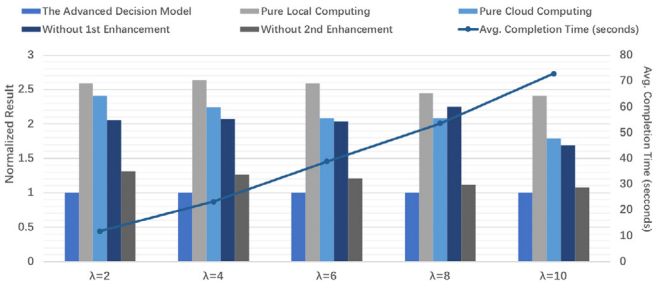


Fig. 5. Normalized average completion time over different workload rate.

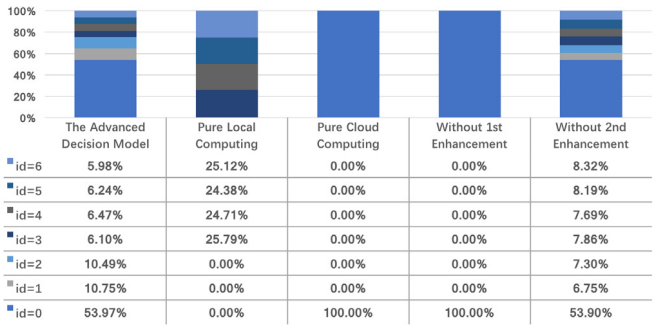


Fig. 6. Percentage of requests handled by each server.

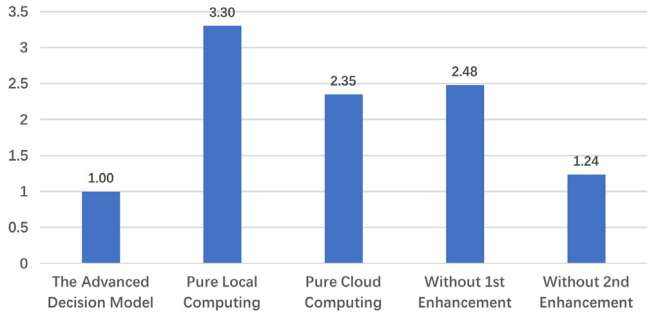


Fig. 7. Normalized overall performance.

average completion time of the advanced decision model and the normalized results for the comparison. Secondly, we fix $\mu = 7$ for the Normal distribution (σ^2 is fixed as 1) and change λ for the Poisson distribution from 2 to 10. Fig. 5 demonstrates the average completion time of the advanced decision model and the normalized results for the comparison.

As the figures show, the average completion time of the advanced decision model grows linearly with the increase of either workload rate or request size. This indicates that no excessive congestion happens. Otherwise, the average completion time will surge in a more offensive way.

Obviously, according to the normalized results, the advanced decision model always performs the best among these five models during these two sets of experiments. Although the normalized average completion time of pure local computing with $\mu = 1$ in Fig. 4 seems to be abnormally larger than others, it is explainable because the record of the number of requests each server has handled shows that skewed requests were generated at that time. So the advanced decision model holds a relatively stable advantage over other models.

5.2.2. The overall performance

Then we do the comparison over the whole solution space, which means we change μ for the Normal distribution from 1 to 9 (σ^2 is fixed as 1), change λ for the Poisson distribution from 2 to 10 and do the comparison over each combination.

Fig. 6 demonstrates the percentage of requests handled by each server (labeled by its index). As expected, the pure local computing schedules all the requests to the servers at the bottom tier; the pure cloud computing schedules all the requests to the central server. The model without 1st enhancement also schedules all the requests to the central server because of the relationship of different servers' computation capacity and related connection's bandwidth. Since we do not consider the change in available energy on each server, this model will do uniform scheduling as long as the settings of the system are determined at the very beginning, regardless of the condition of these servers in runtime. The model without 2nd enhancement does similar scheduling as the advanced decision model. The subtle difference lies in the servers at the second tier, namely the servers with $id = 1,2$. The advanced decision model schedules more requests to these servers. It stems from the fact that some of the requests can jump the queues on those servers at the bottom tier according to the RED discipline. Such jumping does not take place frequently from the second tier to the central server because the servers at the second tier are more powerful and do not get overloaded as the servers at the bottom tier do.

Fig. 7 demonstrates the normalized results of the average completion time. The pure local computing performs worst because it never utilizes the computation resources from other servers and is easy to become overloaded. The model without 1st enhancement performs even worse than the pure cloud computing under our setting because of the overhead of making decisions. The fact that the model without 2nd enhancement performs worse than the advanced decision model validates the analysis mentioned before and proves the effectiveness of jumping (more than 20% improvement). Obviously, the advanced decision model performs the best over the whole solution space.

We use Eq. (12) to approximately estimate the average number of enqueued requests $\bar{n}$ under the advanced decision model. Fig. 8 demonstrates the results under different models over different workload. Different values of μ has little effect on the length of queue, which indicates that our modification of "queue size" mentioned in Section 4.2 works; while different values of λ has a significant effect, which is reasonable since λ determines the workload rate.

$$\frac{\sum \frac{n_i(n_i+1)}{2} \cdot \frac{size_{avg}}{computation_i}}{\sum n_i} = ExeTime_{avg} \tag{12}$$

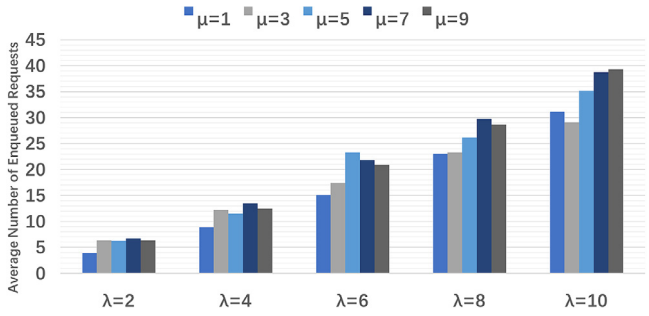


Fig. 8. The average number of enqueued requests under different models over different workload rate.

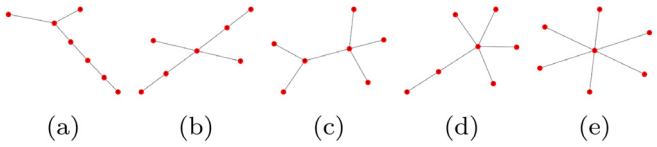


Fig. 9. The average number of enqueued requests under different models over different workload rate.

5.3. Evaluation on different impacts

5.3.1. Impact of skewed test cases

Besides homogeneous test cases as mentioned above, we also evaluate our model with skewed test cases, including skewed requests and skewed topology.

When generating skewed workloads, we reuse the symmetrical hierarchy topology and apply different weights for different servers at the task generator. To be more concrete, we generate 1:2:3:4 tasks respectively on the servers with $id = 3, 4, 5, 6$, so that workloads are skewed at all 3 tiers. For ease of comparison, the provision of computation resources on different servers keeps unchanged.

When generating skewed topology, we adopt Barabasi–Albert Model [30] to generate random scale-free networks according to a preferential attachment mechanism, as [16,31] did. Since generated topology is no longer symmetrical, skewed workloads can take place. For ease of comparison, we still generate the network with 7 nodes and the total amount of computation resources keeps unchanged. As Fig. 9 demonstrates, we totally generate five random networks and tag them as BA-1, BA-2, BA-3, BA-4 and BA-5 respectively. Since the task generator generates more tasks if there are more servers at the bottom tier, the total number of tasks will fluctuate.

In order to mitigate the influence of such fluctuation, the comparative object is set as the average completion time over the average amount of computation generated. The comparison is done over the whole solution space and the settings discussed in Section 5.2 work as the base. Fig. 10 demonstrates the normalized results. The differences are all less than 10%, thus neither skewed requests nor skewed topology makes notable impacts. Our advanced decision model is robust with skewed workloads. Fig. 11 showed the percentage of requests solved by or generated on each server.

5.3.2. Impact of other factors

We also compare the performances with different types of workload. Fig. 12 demonstrates the normalized results. The results of different types of workload are similar to each other. So these five models are not sensitive to the type of workload. The advanced decision model performs the best.

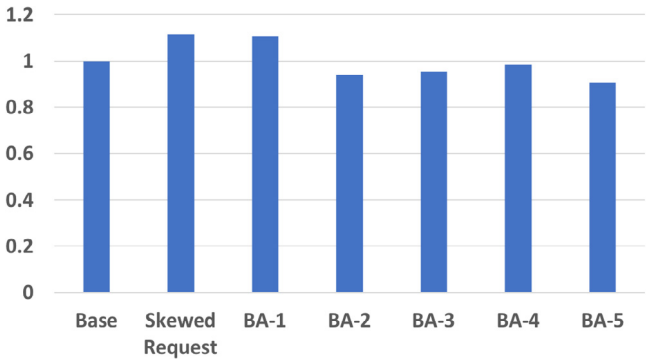
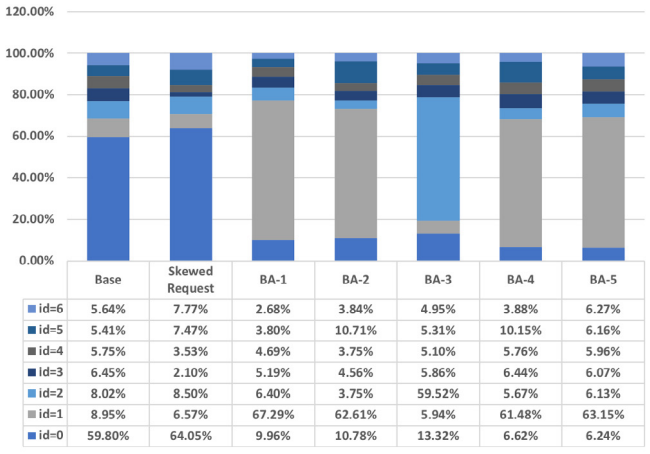
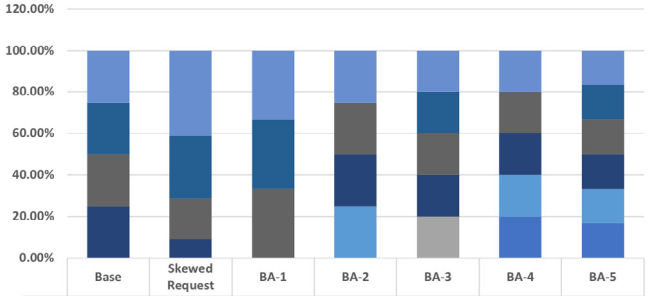


Fig. 10. Normalized Avg. completion time over Avg. amount of computation generated.



	Base	Skewed Request	BA-1	BA-2	BA-3	BA-4	BA-5
id=6	5.64%	7.77%	2.68%	3.84%	4.95%	3.88%	6.27%
id=5	5.41%	7.47%	3.80%	10.71%	5.31%	10.15%	6.16%
id=4	5.75%	3.53%	4.69%	3.75%	5.10%	5.76%	5.96%
id=3	6.45%	2.10%	5.19%	4.56%	5.86%	6.44%	6.07%
id=2	8.02%	8.50%	6.40%	3.75%	59.52%	5.67%	6.13%
id=1	8.95%	6.57%	67.29%	62.61%	5.94%	61.48%	63.15%
id=0	59.80%	64.05%	9.96%	10.78%	13.32%	6.62%	6.24%

(a)



	Base	Skewed Request	BA-1	BA-2	BA-3	BA-4	BA-5
id=6	25.00%	41.13%	33.33%	25.00%	20.00%	20.00%	16.67%
id=5	25.00%	30.28%	33.33%	0.00%	20.00%	0.00%	16.67%
id=4	25.00%	19.72%	33.33%	25.00%	20.00%	20.00%	16.67%
id=3	25.00%	8.87%	0.00%	25.00%	20.00%	20.00%	16.67%
id=2	0.00%	0.00%	0.00%	25.00%	0.00%	20.00%	16.67%
id=1	0.00%	0.00%	0.00%	0.00%	20.00%	0.00%	0.00%
id=0	0.00%	0.00%	0.00%	0.00%	0.00%	20.00%	16.67%

(b)

Fig. 11. Percentage of Requests: (11a) Solved by Each Server, (11b) Generated on Each Server.

We compare the performance with different thresholds for RED. Fig. 13 demonstrates the relative performance of RED with $min_{th}, max_{th} = 5.0, 40.0$ to RED with $min_{th}, max_{th} = 5.0, 20.0$. The narrow range of thresholds works better when the workload is light because it can drop the request with higher probability; the wide range of thresholds works better when the workload is heavy because it can tolerate longer queue before it denies more

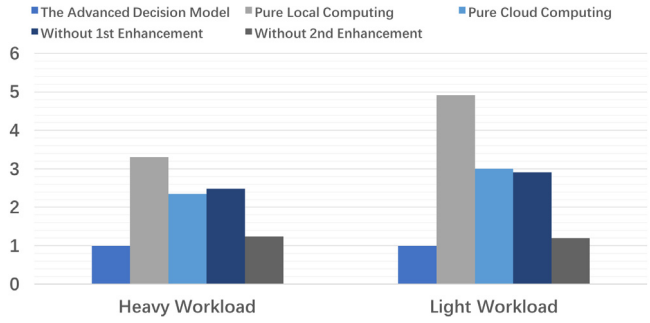


Fig. 12. Normalized performance with different types of workload.

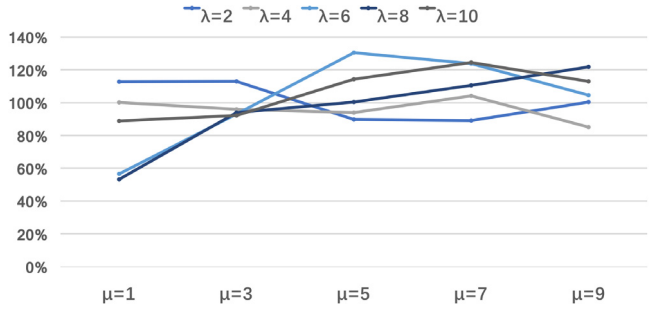


Fig. 13. The effect of different thresholds for RED.

requests. The overall difference between these two setting is only 0.2%. So the performance of the advanced decision model is robust with different thresholds. Nevertheless, different thresholds can fit different tendency of workloads.

6. Conclusion and future work

In this paper, we propose an advanced decision model to solve the computation offloading problem in edge computing. The decision model works on the intrinsic hierarchy and does online scheduling for the incoming requests. It admits the fact that the information can be stale and congestion can take place unexpectedly, even after the decision is made. So it intends to avoid congestion with two enhancements to enable two-way initiative offloading. The results of simulation demonstrate that our method can handle workloads well and these two enhancements are effective.

There is another application scenario where the central server may ask edge servers for help. The advanced decision model can support it instinctively, but further study is required to make progress.

Acknowledgment

This paper is supported by National Natural Science Foundation of China under granted number 61472242.

References

[1] J. Sun, H. Chen, Z. Yin, Aers: An autonomic and elastic resource scheduling framework for cloud applications, in: Services Computing (SCC), 2016 IEEE International Conference on, IEEE, 2016, pp. 66–73.
[2] Z. Yin, H. Chen, J. Sun, F. Hu, Eaers: An enhanced version of autonomic and elastic resource scheduling framework for cloud applications, in: Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on, IEEE, 2017, pp. 512–519.
[3] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[4] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, M. Satyanarayanan, Towards wearable cognitive assistance, in: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, ACM, 2014, pp. 68–81.
[5] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009).
[6] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2015) 974–983.
[7] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, Y. Zhang, Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks, IEEE Access 4 (2016) 5896–5907.
[8] N. Verba, K.-M. Chao, A. James, D. Goldsmith, X. Fei, S.D. Stan, Platform as a service gateway for the fog of things, Adv. Eng. Inf. 33 (2017) 243–257.
[9] L. Tong, Y. Li, W. Gao, A hierarchical edge cloud architecture for mobile computing, in: Computer Communications, IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on, IEEE, 2016, pp. 1–9.
[10] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Trans. Netw. (ToN) 1 (4) (1993) 397–413.
[11] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1628–1656.
[12] C. Xian, Y.-H. Lu, Z. Li, Adaptive computation offloading for energy conservation on battery-powered systems, in: Parallel and Distributed Systems, 2007 International Conference on, Vol. 2, IEEE, 2007, pp. 1–8.
[13] G. Huerta-Canepa, D. Lee, An adaptable application offloading scheme based on application behavior, in: Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on, IEEE, 2008, pp. 387–392.
[14] Y. Zhang, D. Niyato, P. Wang, Offloading in mobile cloudlet systems with intermittent connectivity, IEEE Trans. Mob. Comput. 14 (12) (2015) 2516–2529.
[15] M.R. Rahimi, N. Venkatasubramanian, S. Mehrotra, A.V. Vasilakos, Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture, in: Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, IEEE, 2012, pp. 83–90.
[16] M. Jia, W. Liang, Z. Xu, M. Huang, Cloudlet load balancing in wireless metropolitan area networks, in: Computer Communications, IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on, IEEE, 2016, pp. 1–9.
[17] M.H. Chen, B. Liang, M. Dong, Joint offloading decision and resource allocation for multi-user multi-task mobile cloud, in: Communications (ICC), 2016 IEEE International Conference on, IEEE, 2016, pp. 1–6.
[18] C. You, Y. Zeng, R. Zhang, K. Huang, Asynchronous mobile-edge computation offloading: Energy-efficient resource management, arXiv preprint arXiv:1801.03668, 2018.
[19] S. Ranadheera, S. Maghsudi, E. Hossain, Mobile edge computation offloading using game theory and reinforcement learning, arXiv preprint arXiv:1711.09012, 2017.
[20] J. Xu, L. Chen, S. Ren, Online learning for offloading and autoscaling in energy harvesting mobile edge computing, IEEE Trans. Cogn. Commun. Netw. 3 (3) (2017) 361–373.
[21] J. Zhang, W. Xia, Y. Zhang, Q. Zou, B. Huang, F. Yan, L. Shen, Joint offloading and resource allocation optimization for mobile edge computing, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–6.
[22] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, Perform. Eval. Rev. SIGMETRICS 40 (4) (2013) 23–32.
[23] T.S. Rappaport, et al., Wireless Communications: Principles and Practice, Vol. 2, prentice hall PTR New Jersey, 1996.
[24] Y. Wen, W. Zhang, H. Luo, Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones, in: INFOCOM, 2012 Proceedings IEEE, IEEE, 2012, pp. 2716–2720.
[25] H. Zimmermann, Osi reference model–the iso model of architecture for open systems interconnection, IEEE Trans. Commun. 28 (4) (1980) 425–432.
[26] M.P. Grosvenor, M. Schwarzkopf, I. Gog, R.N. Watson, A.W. Moore, S. Hand, J. Crowcroft, Queues don't matter when you can jump them!, in: NSDI, 2015, pp. 1–14.
[27] V. Jacobson, K. Nichols, K. Poduri, et al., RED in a different light, Unpublished draft available at http://www.cnaf.infn.it/~ferrari/papers/ispn/redlight 9, 1999, 30.
[28] T.H. Kim, K.H. Lee, Refined adaptive red in tcp/ip networks, in: SICE-ICASE, 2006. International Joint Conference, IEEE, 2006, pp. 3722–3725.
[29] S. Floyd, R. Gummadi, S. Shenker, et al., Adaptive red: An algorithm for increasing the robustness of red's active queue management (2001).
[30] R. Albert, H. Jeong, A.L. Barabsi, Internet: diameter of the world-wide web, Nature 401 (6) (1999) 130–131.
[31] M. Jia, J. Cao, W. Liang, Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks, IEEE Trans. Cloud Comput. PP (99) (2017) 1–1.

Zhida Yin received B.S. degree in software engineering from Shanghai Jiao Tong University, Shanghai, China. He is currently pursuing the Master degree in software engineering at Shanghai Jiao Tong University, Shanghai, China. His current research interests include resources scheduling and management in cloud computing and edge computing.

Haopeng Chen received Ph.D. degree from the Department of Computer Science and Engineering, Northwestern Polytechnical University, in 2001. He has been in School of Software, Shanghai Jiao Tong University since 2004 after he finished his two-year postdoctoral research job in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He became an associate professor in 2008. In 2010, he studied and researched in Georgia Institute of Technology as a visiting scholar. His research group focuses on Distributed Computing and Software Engineering. They have kept researching on cloud computing and big data for years.

Fei Hu, he received his Ph.D. from Department of Computer Science and Engineering, Northwestern Polytechnical University in 1998. As an assistant professor, Associate Professor, and Full professor, He has worked in Department of Computer Science and Engineering, Northwestern Polytechnical University from 1993 to 2006. Then, he has worked in School of Software at Shanghai Jiao Tong University from 2006 to 2017. Now, he is the vice chief of key laboratory of China Aeronautical Radio Electronics Research Institute. His research fields are computer real-time system, software testing & quality control, small unmanned aerial vehicle. Recently, his group start some research projects on big data.