Li *et al. EURASIP Journal on Wireless Communications and Networking* (2020) 2020:3
https://doi.org/10.1186/s13638-019-1624-9

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A new load balancing strategy by task allocation in edge computing based on intermediary nodes

Guangshun Li[1,2], Yonghui Yao[1], Junhua Wu[1*], Xiaoxiao Liu[3], Xiaofei Sheng[1] and Qingyan Lin[1]

Abstract

The latency of cloud computing is high for the reason that it is far from terminal users. Edge computing can transfer computing from the center to the network edge. However, the problem of load balancing among different edge nodes still needs to be solved. In this paper, we propose a load balancing strategy by task allocation in edge computing based on intermediary nodes. The intermediary node is used to monitor the global information to obtain the real-time attributes of the edge nodes and complete the classification evaluation. First, edge nodes can be classified to three categories (light-load, normal-load, and heavy-load), according to their inherent attributes and real-time attributes. Then, we propose a task assignment model and allocate new tasks to the relatively lightest load node. Experiments show that our method can balance load among edge nodes and reduce the completion time of tasks.

Keywords: Edge computing, Load balancing, Task allocation, State assessment

1 Introduction

The Internet of Things (IoT) can connect a large number of smart devices across regions and it has become part of many advanced application infrastructures. It also generates large amounts of data, which will keep on growing in the coming years[1, 2]. However, the limitations of its devices make it very complicated to solve current paradigms such as big data or deep learning [3]. In the last few years, the integration of the IoT with disruptive technologies such as cloud computing has provided the capabilities needed in the IoT to address these paradigms[4]. The advent of cloud computing technology has provided us with a light-weight resolution for building various complex business applications[5]. However, cloud computing centers are usually located far away from mobile users, and data delays between users and remote clouds might be long and unpredictable. And users accessing the remote cloud can result in high access latency, which can seriously affect network performance[6].

Edge computing is slowly moving cloud computing applications, data and services from centralized nodes to the edge of the network[7]. And it is located between terminal devices and traditional cloud computing data centers for handling low latency and real-time tasks [8]. This service is seen as a cloud close to the end user to provide computing and services with less latency[9].

Although edge computing can greatly reduce latency, the unreasonable assignment of tasks leads to the unbalanced load of each node [10]. And because of the diversity and heterogeneity of edge computing nodes, the general load balancing algorithm can not be directly applied to edge computing. Therefore, edge computing load balancing has become a very important research topic in academia.

There are two main types of load balancing strategies: static and dynamic. Static load balancing algorithms do not consider the previous state of the node, while distributing the load and it works well when nodes have a small variation in the load. So it is not suitable for edge environment. Load balancing strategy by task allocation in edge computing based on intermediary nodes is a dynamic load balancing technique, which considers the previous state of a node while distributing the load[11].

In this paper, we propose a network architecture for edge computing based on the intermediary node to better

*Correspondence: shdwjh@163.com
[1]School of Information Scicence and Engineering, Qufu Normal University, Yantai Road, 276826 Rizhao, China
Full list of author information is available at the end of the article

Springer Open

© The Author(s). 2019 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

obtain the state information of the node. The intermediary node classifies and evaluates the status of the node by using the intrinsic attribute values and the real-time attribute values. And return the node information with the relatively lightest result. Then we propose a task allocation model that the relatively lightest nodes and the task arrival node are used as the target node to allocate new tasks, while the other nodes are not assigned tasks temporarily, so as to achieve dynamic balancing of the system. The main contribution of this paper can be summarized as follows:

1. We studied the load balancing strategy in the edge computing environment and implemented dynamic load balancing through task allocation. We propose an edge computing network architecture based on intermediary nodes. Compared with the traditional architecture, this architecture adds intermediary nodes in the edge computing layer and cloud computing layer to better control the global information of the edge nodes.
2. For the system with unbalanced initial state, we use naive Bayes algorithm to classify the state of nodes. And standardize the original data in order to avoid highlighting the role of higher-value indicators in the comprehensive analysis when the levels between the indicators vary greatly. And we take the nodes with relatively light classification state and the nodes of task arrival as the target nodes to allocate new tasks, while the other nodes are not assigned tasks temporarily, so as to achieve dynamic balancing.
3. A mathematical framework is cast to investigate the load balancing problem between edge nodes. The purpose of load balancing is achieved by the method of task assignment and estimating the task completion time based on the transmission rate between edge nodes, the computation speed, and the current tasks calculation time.

The rest of the paper is organized as follows. Section 2 reviews the related work in edge computing and load balancing. Section 3 describes the load balancing strategy, including the selection of target nodes and the task allocation model. The simulation results and analysis are presented in Section 4. Finally, Section 5 draws a conclusion.

2 Related work

Edge computing optimizes cloud computing systems by performing data processing at the edge of the network closest to the data source using the concept of caching and data compression. Due to the proximity to the end users, low latency, and other advantages, the research on edge computing has attracted great attention with a large quantity of literature. In this section, we review the research progress of edge computing and load balancing.

In the work of He et al. [12], an improved constrained particle swarm optimization algorithm based on software-defined network (SDN) is proposed in the framework of software-defined cloud-fog network. This algorithm improves the performance of the algorithm by using the opposite property of the mutated particles and reducing the inertia weight linearly. Chen et al. [13], proposed a task allocation model to solve the load balancing at the server level. Calculate the completion time of the large aggregation tasks on each server by treating the tasks offloaded by other servers as one large aggregation task. They formulate a load balancing optimization problem for minimizing deadline misses and total runtime for connected car systems in fog computing.

In the work of Wang et al. [14], a distributed city-wide traffic management system is constructed. And design an offloading algorithm for real-time traffic management in fog-based internet of vehicle (IoV) systems, with the purpose of minimizing the average response time of the traffic management server (TMS) for messages. Ning et al. [15], investigated a joint computation offloading, power allocation, and channel assignment (COPACA) scheme for 5G-enabled traffic management systems, with the purpose of maximizing the achievable sum rate. In the work of Ning et al. [16], in order to satisfy heterogeneous requirements of communication, computation and storage in IoVs, they constructed an energy-efficient scheduling framework for MEC-enabled IoVs to minimize the energy consumption of road side units (RSUs) under task latency constraints. Ning et al. [17] , proposed a deep learning based data transmission scheme by exploring trirelationships among vehicles at the edge of networks (i.e., edge of vehicles) by jointly considering the social and physical characteristics. In the work of Ning et al. [18] a deep reinforcement learning (DRL) method is integrated with vehicular edge computing to solve the computation ofoading problem, where we jointly study the optimization of task scheduling and resource allocation in vehicular networks.

Our team has done a lot of work on edge computing and fog computing. For example, the resource scheduling method for fog computing is studied [19], the data processing delay optimization in mobile edge computing [20], and the resource scheduling in edge computing[21], etc. In this paper, We focus on the load balancing problem of edge computing. In our research, for the system with unbalanced initial state, we propose a load balancing strategy by task allocation in edge computing based on intermediary nodes. First, the state of nodes is classified and evaluated according to their inherent and real-time attributes. Then, according to the classification results, the target nodes are selected. Finally, the new task assignment is completed according to the task assignment model proposed by us.

3 Load balancing strategy

In this paper, we will study the load balancing technology under the edge computing architecture based on intermediary nodes. The network model of this architecture is shown in Fig. 1.

In this architecture, we store the intrinsic attributes of the node to the intermediary node before applying the load balancing strategy. When a new task arrives at a node, this node sends a request signal to the intermediary node. The intermediary node is responsible for forwarding the signal to the edge node and the edge node that received the signal returns its real-time properties values. After receiving the real-time attribute, the intermediary node starts classification of the edge node.

3.1 Selection of target nodes

3.1.1 Selection of load attributes

There are many factors affecting load balancing, including memory, CPU, disk and network bandwidth. However, it is incomplete to judge the load state of nodes by ignoring one or several factors. In our research, we will evaluate the state of nodes by combining the intrinsic and real-time attributes of nodes. Intrinsic and real-time properties are defined as follows [22]:

Definition 1 *Intrinsic attributes. Static properties of the nodes, including physical memory, CPU main frequency multiplied by the number of cores, disk size, and network bandwidth.*

Definition 2 *Real-time attributes. Dynamic attributes of nodes, that is, attribute values monitored by the intermediary node in real time, including memory usage, disk usage, CPU utilization, and bandwidth utilization.*

When the levels between the indicators vary greatly, if we directly use the original index value for analysis it will highlight the role of higher-value indicators in the comprehensive analysis and relatively weaken the role of lower-value indicators [22]. So we perform dimensionless processing of the intrinsic attributes, and each attribute value of node i is represented as:

Definition 3 *The values of load attribute [23]. The combination of intrinsic and real-time attributes of edge node i is used as a criterion for classifying load states of nodes. Expressed as* $L = (L_1, L_2, L_3, L_4)$*, and each attribute value of node i is represented as:*

The property values of memory: $L_1 = \sigma_1 R_i^1 + \sigma_2 \left(1 - R_i^2\right)$, where R_i^1 is the size of physical memory after dimensionless processing, and $R_i^1 = \frac{R_i - \min(R_i)}{\max(R_i) - \min(R_i)}$, R_i represents the memory size of node i. R_i^2 is the memory utilization of node i. And $\sigma_1 + \sigma_2 = 1$.

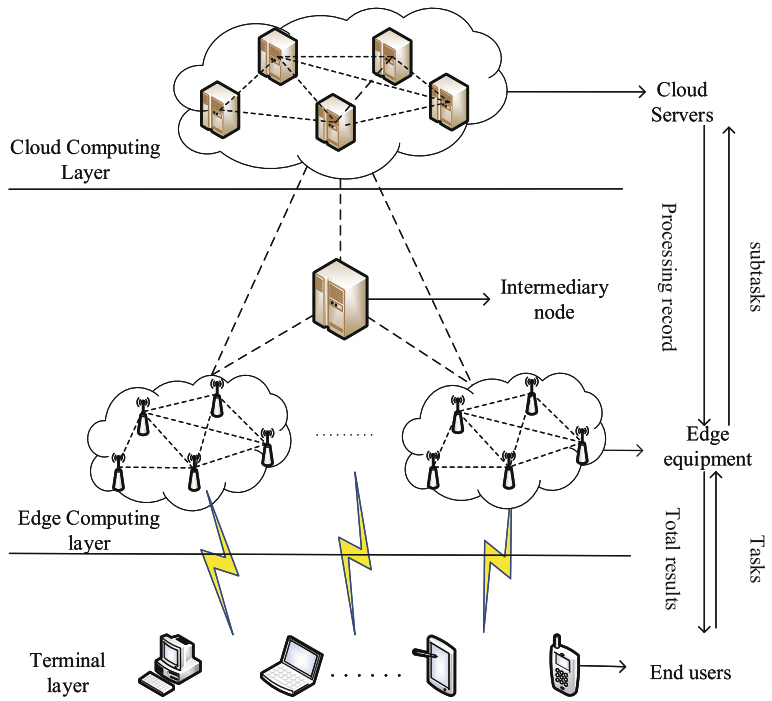

Fig. 1 Network architecture of edge computing based on mediation nodes

The property values of CPU: $L_2 = \varepsilon_1 C_i^1 + \varepsilon_2 \left(1 - C_i^2\right)$, where C_i^1 the product of the CPU main frequency and core number after dimensionless processing, and $C_i^1 = \frac{C_i - \min(C_i)}{\max(C_i) - \min(C_i)}$, C_i represents the product of the CPU main frequency and core number of nodes i. C_i^2 is the CPU utilization of node i. And $\varepsilon_1 + \varepsilon_2 = 1$.

The property values of disks: $L_3 = \delta_1 D_i^1 + \delta_2 \left(1 - D_i^2\right)$, where D_i^1 is the size of disks after dimensionless processing, and $D_i^1 = \frac{D_i - \min(D_i)}{\max(D_i) - \min(D_i)}$, D_i represents the disks size of nodes i. D_i^2 is the disks utilization of node i. And $\delta_1 + \delta_2 = 1$.

The property values of bandwidth: $L_4 = \omega_1 B_i^1 + \omega_2 \left(1 - B_i^2\right)$, where B_i^1 is the size of bandwidth after dimensionless processing, and $B_i^1 = \frac{B_i - \min(B_i)}{\max(D_i) - \min(B_i)}$, B_i represents the bandwidth size of nodes i. B_i^2 is the bandwidth utilization of node i. And $\omega_1 + \omega_2 = 1$.

Definition 4 *Sample classification set:* $T = \{T_j | j = 1, 2, 3\}$, T_1 *represents the light-load state,* T_2 *represents the normal-load state, and* T_3 *represents the heavy-load state.*

3.1.2 Classification of node states

In this section, we use the load attribute value as a basis to classify the state of the node by the Naive Bayes algorithm. Based on Bayesian theory, this classification method is a pattern recognition method with known prior probability and conditional probability [24]. According to Bayesian theorem, when each attribute is independent of each other, its classification result is the most accurate. The selected attributes of us are actually independent between each other and thus meet the condition of it. Let the sample space be U and the prior probability of training sample classification T_j be $Pr(T_j)(j = 1, 2, 3)$. Its value is equal to the total number of samples belonging to class T_j divided by the total number of training samples $|U|$. For an unknown sample n_x, the conditional probability that it belongs to the T_j class is $Pr(n_x|T_j)$. According to Bayesian theorem, the posterior probability that it belongs to the T_j class is $Pr(T_j|n_x)$:

$$Pr(T_j|n_x) = \frac{Pr(n_x|T_j)Pr(T_j)}{Pr(n_x)}. \tag{1}$$

Let the load attribute value $L(n_x) = \left(L_1^{n_x}, L_2^{n_x}, L_3^{n_x}, L_4^{n_x}\right)$ of unknown sample n_x , where $L_k^{n_x}$ represents the kth attribute value of sample n_x , because we assume that $L_k^{n_x}(k = 1, 2, 3, 4)$ is independent of each other, the conditional probability of belonging to T_j is as follows:

$$Pr(n_x|T_j) = Pr((L_1^{n_x}, L_2^{n_x}, L_3^{n_x}, L_4^{n_x})|T_j) = \prod_{k=1}^{4} Pr(L_k^{n_x}|T_j). \tag{2}$$

where $Pr(L_k^{n_x}|T_j)$ denotes the probability when the kth load attribute value of sample n_x belong to T_j classification. From (1) and (2), the posterior probability of T_j is obtained as follows:

$$Pr(T_j|n_x) = \frac{Pr(T_j) * \prod_{k=1}^{4} Pr(L_k^{n_x}|T_j)}{Pr(n_x)}. \tag{3}$$

According to the Naive Bayes classification method, the posterior probability multiplied by the prior probability maximum term is the class of the unknown sample n_x , and it is represented by the following formula:

$$\arg\max\{Pr(T_j|n_x)Pr(T_j)\}(j = 1, 2, 3). \tag{4}$$

From the Eqs. (3) and (4), the classification decision function of sample n_x is:

$$\arg\max\{Pr(T_j)^2 * \prod_{k=1}^{4} Pr(L_k^{n_x}|T_j)\}(j = 1, 2, 3). \tag{5}$$

Therefore, the state of the sample can be represented by Eq. (6).

$$T_{V_x} = \begin{cases} \text{light} - \text{load}, & \arg\max\left\{Pr\left(T_j\right)^2 * \prod_{k=1}^{4} Pr\left(L_k^{n_x}|T_j\right)\right\} = 1 \\ \text{normal} - \text{load}, & \arg\max\left\{Pr\left(T_j\right)^2 * \prod_{k=1}^{4} Pr\left(L_k^{n_x}|T_j\right)\right\} = 2 \\ \text{heavy} - \text{load}, & \arg\max\left\{Pr\left(T_j\right)^2 * \prod_{k=1}^{4} Pr\left(L_k^{n_x}|T_j\right)\right\} = 3 \end{cases} \tag{6}$$

According to the Eq. (6), when the predicted result of an unknown node state is 1, it represents that the current state of the node is light load state, and if the predicted result is 2, it indicates that the current state of the node is normal load state, otherwise it is heavy load state.

The intermediary node divides the edge nodes into three categories according to the above method, and returns the information of node with relatively light node status (the smallest classification result). This kind node is designated as the target node.

3.2 Task allocation model

When one or more tasks arrive at node n_i simultaneously, these tasks are merged into an aggregated task U. And according to the information of the target node, the aggregated task is decomposed into several subtasks $u_j = \alpha_j U$ which are handled by different target nodes and the node which tasks arrived.

(1) The transmission time of task

The transmission time is the size of the subtasks divided by the data transmission rate from edge node n_i to target node n_j or cloud server d, that is:

$$t_1 = \frac{\alpha_j U}{B_{n_i,n_j}}. \tag{7}$$

$$t_2 = \frac{\alpha_d U}{B_{n_i,d}}. \tag{8}$$

Among them, B_{n_i,n_j} is the data transfer rate of the edge node n_i to the target node n_j, and $B_{n_i,n_j} = \infty$ when $n_i = n_j$[11]. $B_{n_i,d}$ is the data transfer rate of the edge node n_i to the cloud server d.

(2) The computation time of subtasks

$$t_3 = \frac{\alpha_j U}{f_{n_j}} + \tau_{n_i,n_j}\frac{N_{n_j}}{f_{n_j}}. \tag{9}$$

$$t_4 = \frac{\alpha_d U}{f_d} + \tau_{n_i,d}\frac{N_d}{f_d}. \tag{10}$$

Where, $\frac{\alpha_j U}{f_{n_j}}$ is the calculation time of the subtask at the target node n_j, and f_{n_j} is the computation speed of the target node n_j. $\tau_{n_i,n_j}\frac{N_{n_j}}{f_{n_j}}$ is the current tasks calculation time of the target node n_j. And $\tau_{n_i,n_j} = \lceil \alpha_j \rceil$, it denotes whether there is a task assignment relationship between the edge nodes n_i and n_j. $\tau_{n_i,n_j} = 1$ denote that the relationship exists; $\tau_{n_i,n_j} = 0$ indicates that the relationship does not exist [6]. N_{n_j} is the current task size of the target node n_j. Similarly, we can get an explanation of the computation time t_4 of subtasks in cloud servers.

(3) The transmission time of the computing result. In most cases, the computing result is a small packet such as a control signal; thus, the transmission time of the computing result can be ignored [25].

- The completion time of subtasks between edge nodes

$$T_1(\alpha_j) = \max(t_1 + t_3) = \max\left(\frac{\alpha_j U}{B_{n_i,n_j}} + \frac{\alpha_j U}{f_{n_j}} + \tau_{n_i,n_j}\frac{N_{n_j}}{f_{n_j}}\right). \tag{11}$$

- The completion time of subtasks on cloud server

$$T_2(\alpha_d) = (t_2 + t_4) = \frac{\alpha_d U}{B_{n_i,d}} + \frac{\alpha_d U}{f_d} + \tau_{n_i,d}\frac{N_d}{f_d}. \tag{12}$$

- Total completion time of aggregate task U

$$T(\alpha_j, \alpha_d) = \max\left\{\frac{\alpha_j U}{B_{n_i,n_j}} + \frac{\alpha_j U}{f_{n_j}} + \tau_{n_i,n_j}\frac{N_{n_j}}{f_{n_j}}, \frac{\alpha_d U}{B_{n_i,d}} + \frac{\alpha_d U}{f_d} + \tau_{n_i,d}\frac{N_d}{f_d}\right\}. \tag{13}$$

In order to minimize the completion time, it is necessary to determine the optimal $\{\alpha_j, \alpha_d\}$ set. In summary, the problem is modeled as follows:

$$\min\left\{\max\left\{\frac{\alpha_j U}{B_{n_i,n_j}} + \frac{\alpha_j U}{f_{n_j}} + \tau_{n_i,n_j}\frac{N_{n_j}}{f_{n_j}}, \frac{\alpha_d U}{B_{n_i,d}} + \frac{\alpha_d U}{f_d} + \tau_{n_i,d}\frac{N_d}{f_d}\right\}\right\} \tag{14}$$

$$s.t. \sum_{j=1}^{k}(\alpha_j + \alpha_d) = 1$$

In this task model, the subtask assigned to each edge node satisfies $u_j = \alpha_j U$. Therefore, the proportion of tasks allocated to target node and cloud can form a $k+1$ dimensional vector $\alpha = (\alpha_1, \alpha_2, ..., \alpha_k, \alpha_{k+1})^{\hat{o}}$ [12]. Assuming that the edge node n_1 receives the current task, the total completion time T can be described as

$$T(\alpha) = \max\left(\frac{\alpha_1 U}{B_{n_1,n_1}} + \frac{\alpha_1 U}{f_{n_1}} + \tau_{n_1,n_1}\frac{N_{n_1}}{f_{n_1}}, \ldots, \frac{\alpha_k U}{B_{n_1,n_k}} + \frac{\alpha_k U}{f_{n_k}} + \tau_{n_1,n_k}\frac{N_{n_k}}{f_{n_k}}, \frac{\alpha_d U}{B_{n_1,d}} + \frac{\alpha_d U}{f_d} + \tau_{n_1,d}\frac{N_d}{f_d}\right)$$

Therefore, the mapping of the computing task is solved in the case where the aggregation task U is known, that is, the proportion of tasks assigned to each target node, the solution of vector α. In order to avoid overloading a node after being assigned a large number of tasks, we make the subtask u less than or equal to the average load of the system, that is $u_j \le \frac{\left(U + \sum_{j=1}^{k} N_j\right)}{m}$. Assuming that the total number of edge nodes is m, the problem can be reduced to the following optimization problems:

$$\begin{aligned} \alpha &= \arg\min_{U \in I}\{T(\alpha)\} \\ s.t.\ & 0 \le \alpha_j \le 1 \\ & 0 \le \alpha_j \le 1 \\ & \alpha_j U \le \frac{\left(U + \sum_{j=1}^{k} N_j\right)}{m} \\ & \sum_{j=1}^{k}(\alpha_j + \alpha_d) = 1 \end{aligned} \tag{15}$$

The search space I for the optimization problem is:
$I \triangleq \prod_{j=1}^{k}[\alpha_{jmin}, \alpha_{jmax}] = \prod_{j=1}^{k}[0, 1]$

Particle swarm optimization (PSO) has the advantages of easy description and understanding, strong search ability and simple programming. Therefore, for the above optimization problem, we choose PSO algorithm for intelligent optimization. Due to the limitation of population diversity, the PSO algorithm appears premature convergence, so we adopt the Modified Particle Swarm Optimization (MPSO) Algorithm which introduces the reverse flight of mutation particles [26]. This algorithm can effectively avoid falling into local optimum in the iterative process.

When solving optimization problems, particles in the swarm $\{X_i^L\}_{i=1}^N$ move in search space I to find the best position X, i.e., α. N is the size of the particle swarm and the number of iterations is L. The position and velocity vectors of the ith particle in the evolutionary L generation are expressed as follows:

$$\begin{cases} X_i^L = \left[x_{i1}^L, x_{i2}^L, \ldots, x_{ik+1}^L\right] \\ v_i^L = \left[v_{i1}^L, v_{i2}^L, \ldots, v_{ik+1}^L\right] \end{cases} \tag{16}$$

where, $v_i^L \in M, M \triangleq \prod_{i=1}^{k+1} [-v_{i\max}, v_{i\max}], v_{i\max} = \frac{1}{2}(\alpha_{j\max}, -\alpha_{j\max})$

This is an optimization problem with constraints. Therefore, the penalty function method is used to deal with the constraints [27], and the fitness function is defined as follows:

$$F(X) = \begin{cases} f(X) & X \in F \\ f(X) + r \sum_{i=1}^{2k+3} f_i(X) + \varphi(X, L) & X \in I - F \end{cases} \quad (17)$$

Where, r represents the penalty factor, $f_i(X)$ denotes the constraint violation measure of the infeasible particles on the jth constraint. Moreover, $\varphi(X, L)$ denotes additional heuristics value for infeasible particles in the Lth generation of the algorithm[27]. $f_i(X)$ expressed by Eq. (19).

$$f_i = \begin{cases} \max(0, -X(D)), & 1 \le i \le k+1 \\ \max\left(0, X(D) - \frac{U + \sum_{j=1}^{m} N_j}{U * m}\right), & k+2 \le i \le 2k+2 \\ \left|\sum_{D=1}^{k+1} X(D) - 1\right|, & i = 2k+3 \end{cases} \quad (18)$$

$\varphi(X, L)$ expressed by Eq. (20).

$$\varphi(X, L) = P(L) - \min_{X \in I - F} r \sum_{i=1}^{2k+3} f_i(X) \quad (19)$$

$$P(L) = \max\left(P(L-1), \min_{X \in F} f(X)\right) \quad (20)$$

$f(X)$ represents the fitness value of the Lth generation feasible particle. $P(L)$ records the feasible particles with the maximum fitness value obtained by the evolution of the algorithm to the Lth generation. And the value is dynamically updated according to Eq. (21) during the execution of the algorithm. When the algorithm is executed, the update formulas of particle velocity and position are follows as:

$$v_i^{L+1} = \omega v_i^L + c_1 \text{rand}() \left(P_i^L - X_i^L\right) + c_2 \text{rand}() \left(g^L - X_i^L\right) \quad (21)$$

$$X_i^{L+1} = X_i^L + v_i^{L+1} \quad (22)$$

In the Eq. (22), ω is inertia weight. rand() is a random number evenly distributed in the interval $[0, 1]$; c_1 and c_2 are two accelerating factors. Defining the individual historical optimal position P_i^L of the ith particle is the position with the best fitness value experienced by the ith particle; The global historical optimal location g^L is the location with the best adaptive value experienced by all particles in the particle swarm during the evolution.

In addition, the updated formula of inertia weight is as follows:

$$\omega = \omega_{\min} - \frac{\omega_{\min} - \omega_{\max}}{L_{\max}} L \quad (23)$$

In order to avoid falling into the local optimal risk, we introduce the reverse of the flight of mutation particles [26], the position and velocity formulas are updated as follows:

$$v_i^{L+1} = -v_i^L \quad (24)$$

$$X_i^{L+1} = X_i^L - v_i^{L+1} \quad (25)$$

The basic parameters of the MPSO algorithm are the population size N is equal to 50, the maximum number of iterations $L_{\max}$ is 1000, the acceleration factors c_1 and c_2 are equal to 1.0, and $\omega \in [0.4, 0.9]$; $\omega_{\min} = 0.4$; $\omega_{\max} = 0.9$.

4 Results and discussion

4.1 Experiment setup

In this experiment, we represent the task arrival node as n_1. When $n_j \neq n_1$, the data transmission rate B_{n_1,n_j} from node n_1 to target node n_j is randomly selected from the integer between 80 and 100 Mbps. Otherwise $B_{n_1,n_j} = \infty$. Data transmission rates $B_{n_1,d}$ between edge nodes and cloud nodes are randomly selected between 20 and 30 Mbps integers[28]. The details are shown in Table 1.

4.2 Simulation results and analysis

4.2.1 Effect of number of target nodes on completion time

In this part, we set the total number of normal-load and heavy-load nodes to be 4, the total number of nodes m to be 10, 12, and 14, respectively, and compared the completion time. As shown in Fig. 2, When the task is small, the difference in completion time is not significant. When the task is large, because $B_{n_1,n_j} = \infty$ when $n_j = n_1$, the task assignment to itself will exceed the average task amount of the system, but we limit the size of the subtasks, so the difference of task completion time is significantly increased.

Table 1 Simulation parameters

Parameter name	Parameter value
Aggregated task U	0.1 ∼ 1 Gb.
Data transmission rate between edge nodes $B_{n_1,n_j}, n_1 \neq n_j$	[85, 100] Mbps
Data transmission rate between edge node and cloud service $B_{n_1,d}$	[20, 30] Mbps
Computing speed of edge nodes f_{n_j}	[0.5, 2] Gbps
Computing speed of cloud service f_d	10 Gbps

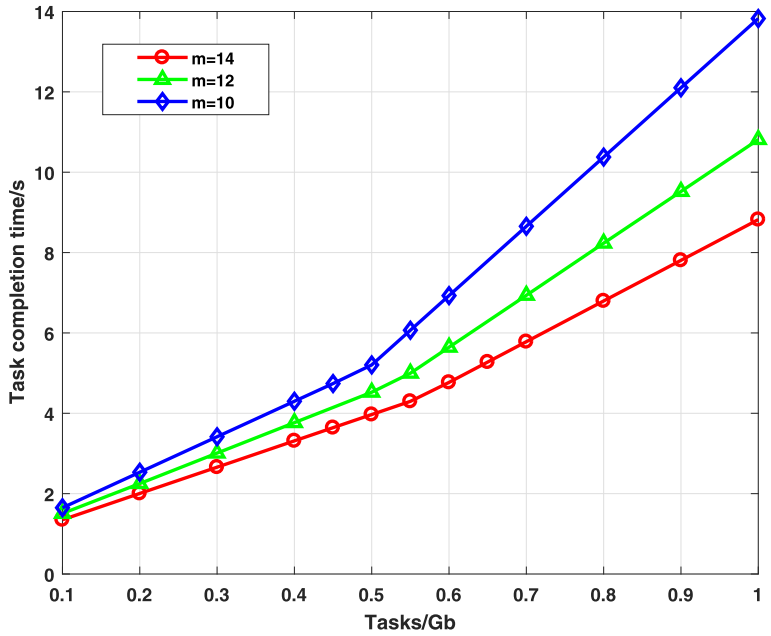

Fig. 2 The effect of the number of target nodes on the completion time

4.2.2 *Impact of system architecture on completion time*

Without loss of generality, in the environment of $M = 12$, we analyzed the impact of cloud server on task completion time. The results are shown in Fig. 3,

E-CC denotes the participation of cloud servers, and EC denotes the absence of cloud servers. When there are a few tasks, the impact on task completion time is small. However, when the amount of tasks is large, due to the fast computing speed of cloud computing, the task completion time of the system with cloud servers is obviously better than that of the system without cloud servers.

4.2.3 *The effect of total number of nodes on task distribution*

In this part, we analyze the distribution of tasks at the edge nodes when the number of nodes is different. And we evaluate the distribution of tasks by load distribution

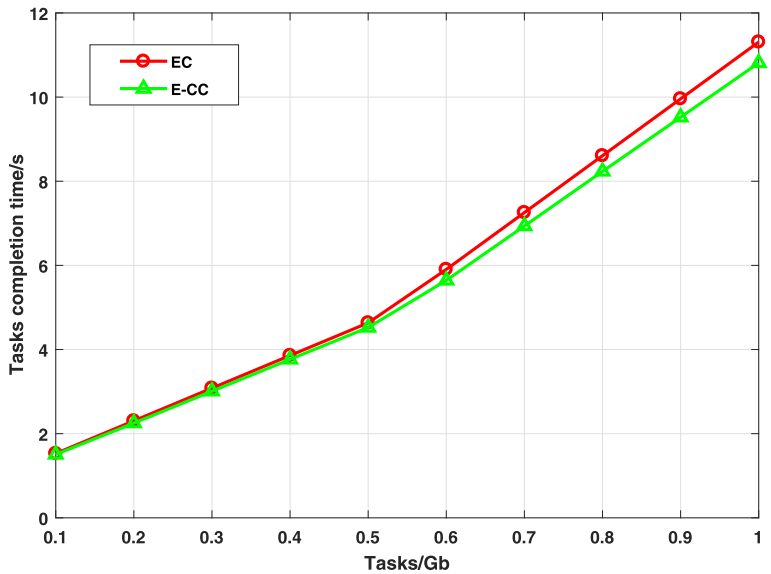

Fig. 3 Impact of cloud server on completion time

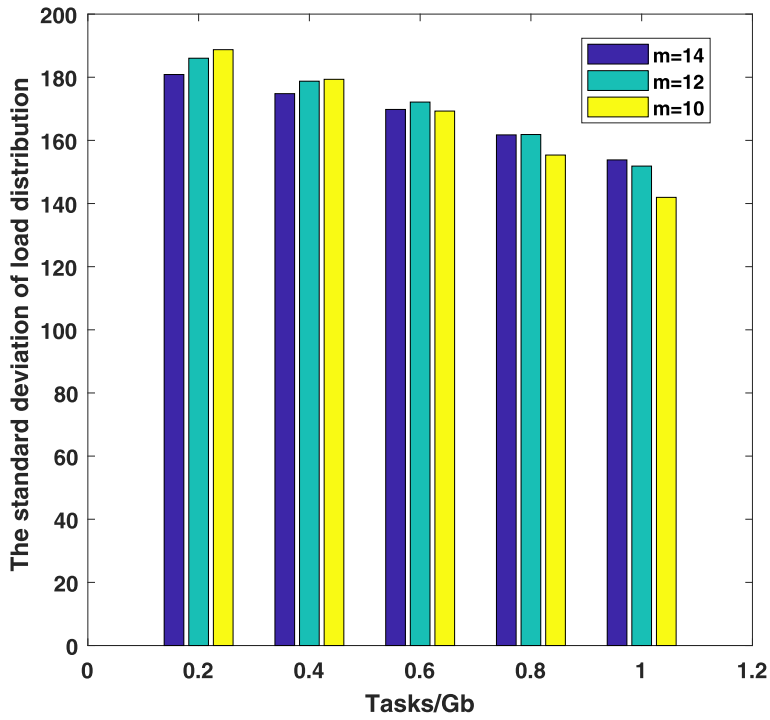

Fig. 4 The effect of total number of nodes on task distribution

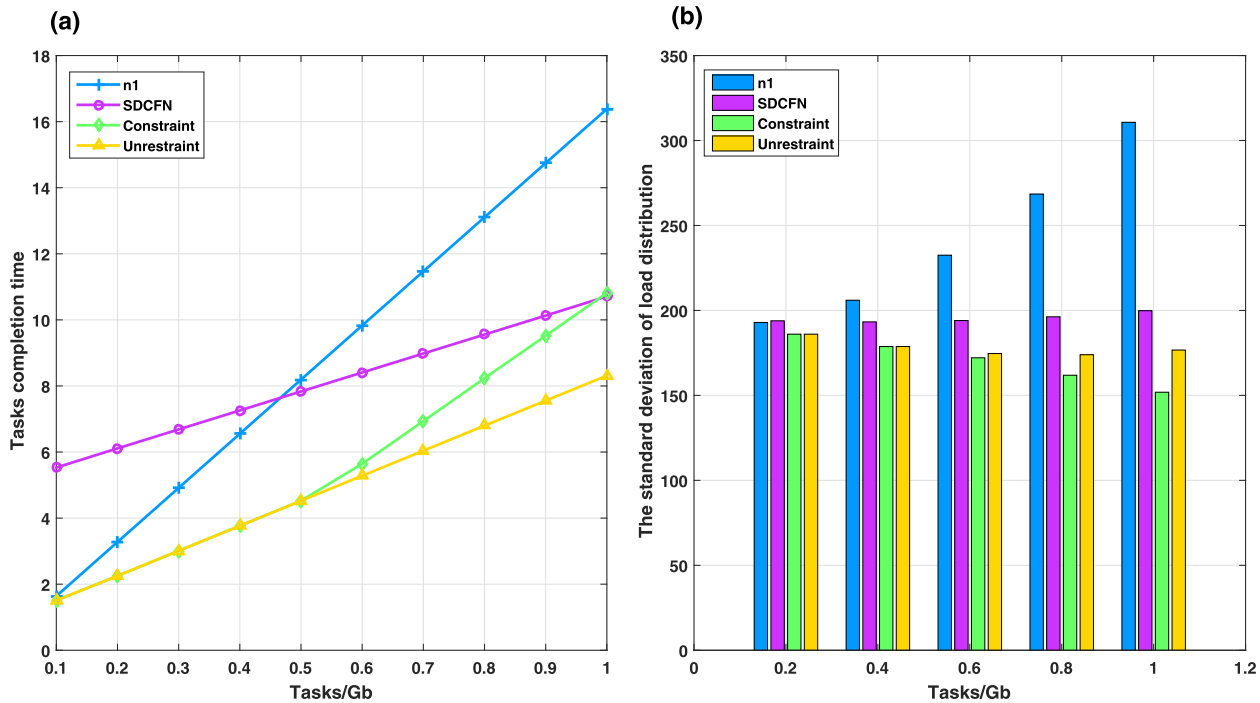

Fig. 5 a Effects of different strategies on task completion time. **b** Effects of different strategies on standard deviation of load distribution

standard deviation (SD)[29], $SD = \sqrt{\frac{\sum_{j=1}^{m}(\mathrm{Load}_{n_j} - \mathrm{Load}_{\mathrm{avg}})_2}{m}}$, where $\mathrm{Load}_{n_j} = N_j + u_j$, $\mathrm{Load}_{\mathrm{avg}} = \frac{\sum_{j=1}^{m} N_j + u_j}{m}$.

The smaller the standard deviation, the more balanced the task distribution. The result is shown in Fig. 4.

In the case that the heavy-load and normal-load nodes are not assigned new tasks for the time being, the standard deviation of load distribution decreases with the increase of new tasks. That is, with the increase of new tasks, the system gradually tends to a relatively balanced state.

When the new task is small, the more nodes there are, the smaller the average task is, and the more balanced the load distribution is. When the new task is large, due to the number of target nodes is small, it will allocate more tasks and get closer to the average load. Therefore, the more uniform the load distribution, the smaller the load distribution standard deviation.

4.2.4 *Comparison of task completion time and load distribution under different strategies*

We compare the tasks completion time and the load distribution of our strategy with single node and SDCFN [12]. As shown in Fig. 5.

In Fig. 5, $n1$ denotes that the U is independently completed by node $n1$, *Constraint* denotes the results of our load balancing add the $\alpha_j U \leq \frac{\left(U + \sum_{j=1}^{k} N_j\right)}{m}$ constraint. *Unrestraint* denotes the results of our load balancing without constraint. From the Fig. 5a, we can find that the completion time of $n1$ are larger than our strategy. And due to *SDCFN* strategy does not consider the completion time of the current tasks of the nodes, the completion time is longer when $U < 0.97G$. When $U > 0.97G$, due to the small number of nodes available for our strategy, its completion time is relatively long. When the task is small, whether to add this constraint has little influence on completion time. When the task is large, because the transmission rate of $n1$ is ∞, the task it undertakes will be constrained after adding the constraint, so its completion time is longer. From the Fig. 5b, we can find that the load distribution standard deviation of n1 and *SDCFN* strategy are larger than our strategy. When the task is small, whether to add this constraint has little influence on load distribution standard deviation. When the task is large, the standard deviation of load distribution standard deviation decreases with the increase of tasks after added the constraint. If we do not add this constraint, since the transmission rate of $n1$ is ∞, the task assigned to itself by $n1$ will exceed the average load, thus causing the standard deviation of the load distribution of the system to increase.

5 Conclusion

In this paper, we propose an edge computing network architecture based on the intermediary node. This architecture not only can obtain the state information of the node better , but also can reduce the pressure of edge nodes. On this basis, a task allocation strategy is proposed to balance the load and reduce the task completion time. In this model, the light-load node and the task arrival node are used as the target node to allocate new tasks, while the other nodes are not assigned tasks temporarily, so as to achieve dynamic balancing. Experiments show that this strategy can not only balance the load between nodes, but also reduce the completion time of tasks. When the task is small, our strategy is significantly better than other methods. Finally, we give two alternative strategies. The first is, for tasks with high task completion time requirements, we can adopt the strategy of unconstrained to minimize the completion time. The second is, for the tasks which requirements for completion time are not too high, we can adopt a constrained strategy to better balance the load between nodes and improve quality of service.

Abbreviations

COPACA: Computation offloading, power allocation, and channel assignment; DRL: Deep reinforcement learning; IoT: The Internet of Things; IoV: Internet of Vehicles; MPSO: Modified particle swarm optimization algorithm; PSO: Particle swarm optimization algorithm; RSUs: Road side units; SDCFN: Software-defined cloud/fog network; SDN: Software-defined network; TMS: Traffic management server

Authors' contributions

YY, XS, and QL are the principal contributors in terms of simulation modelling, writing, and the generation/interpretation of numerical results. In a supervising role, GL , JW, and XL formulated the research problem and contributed to the simulation modelling and the discussion of results. All authors read and approved the final manuscript.

Funding

This work is supported by the National Natural Science Foundation of China (61672321, 61771289, and 61832012), Shandong province key research and development plan (2019GGX101050), Shandong provincial Graduate Education Innovation Program (SDYY14052 and SDYY15049), Qufu Normal University Science and Technology Project (xkj201525), Shandong province agricultural machinery equipment research and development innovation project (2018YZ002), and High-Level Teachers in Beijing Municipal Universities in the Period of the 13th Five-Year Plan (CIT&TCD 201704069).

Availability of data and materials

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]School of Information Scicence and Engineering, Qufu Normal University, Yantai Road, 276826 Rizhao, China. [2]Department of Computer, Hong Kong Polytechnic University, 999077 Hong Kong, China. [3]Smart City College, Beijing Union University, 100101 Beijing, China.

Received: 25 September 2019 Accepted: 18 December 2019
Published online: 02 January 2020

References

1. S. Wan, Y. Zhao, T. Wang, Z. Gu, Q. H. Abbasi, K. K. R. Choo, Multi-dimensional data indexing and range query processing via voronoi diagram for internet of things. Future Gener. Comput. Syst. **91**, 382–391 (2019)

2. A. Zaslavsky, C. Perera, D. Georgakopoulos, Sensing as a service and big data (2013). arXiv preprint arXiv:1301.0159
3. J. Yu, H. Jiang, G. Wang, Q. Guo, Clustering-based energy-efficient broadcast tree in wireless networks. Int. J. Comput. Commun. Control. **7**(4), 265–270 (2014)
4. C. Martin, M. Diaz, B. Munoz, in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. An edge computing architecture in the Internet of Things, (2018), pp. 99–102. https://doi.org/10.1109/ISORC.2018.00021
5. L. Qi, J. Yu, Z. Zhou, An invocation cost optimization method for web services in cloud environment. Sci. Program. **2017**(11), 1–9 (2017)
6. M. A. Nadeem, M. A. Saeed, in *Sixth International Conference on Innovative Computing Technology*. Fog computing: an emerging paradigm, (2016). https://doi.org/10.1109/intech.2016.7845043
7. S. Singh, in *2017 International Conference on Big Data, IoT and Data Science (BID)*. Optimize cloud computations using edge computing, (2017), pp. 49–53. https://doi.org/10.1109/BID.2017.8336572
8. X. Xu, Y. Xue, L. Qi, Y. Yuan, X. Zhang, T. Umer, S. Wan, An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles. Future Gener. Comput. Syst. **96**, 89–100 (2019)
9. T. Zhu, T. Shi, J. Li, Z. Cai, X. Zhou, Task scheduling in deadline-aware mobile edge computing systems. IEEE Internet Things J., 1–1 (2018). https://doi.org/10.1109/jiot.2018.2874954
10. L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, Y. Pan, Stochastic load balancing for virtual resource management in datacenters. IEEE Trans. Cloud Comput. **PP**(99), 1–1 (2016)
11. K. R. R. Babu, A. A. Joy, P. Samuel, in *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*. Load balancing of tasks in cloud computing environment based on bee colony algorithm, (2015), pp. 89–93. https://doi.org/10.1109/ICACC.2015.47
12. X. He, Z. Ren, C. Shi, F. Jian, A novel load balancing strategy of software-defined cloud/fog networking in the internet of vehicles. Chin. Commun. **13**(S2), 145–154 (2016)
13. Y. A. Chen, J. P. Walters, S. P. Crago, in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. Load balancing for minimizing deadline misses and total runtime for connected car systems in fog computing, (2017), pp. 683–690. https://doi.org/10.1109/ispa/iucc.2017.00107
14. X. Wang, Z. Ning, L. Wang, Offloading in internet of vehicles: A fog-enabled real-time traffic management system. IEEE Trans. Ind. Inf. (2018). https://doi.org/10.1109/tii.2018.2816590
15. Z. Ning, X. Wang, F. Xia, J. J. Rodrigues, Joint computation offloading, power allocation, and channel assignment for 5g-enabled traffic management systems. IEEE Trans. Ind. Inf. (2019). https://doi.org/10.1109/tii.2019.2892767
16. Z. Ning, J. Huang, X. Wang, J. J. P. C. Rodrigues, L. Guo, Mobile edge computing-enabled internet of vehicles: toward energy-efcient scheduling. IEEE Netw. (2019). https://doi.org/10.1109/mnet.2019.1800309
17. Z. Ning, Y. Feng, X. Kong, L. Guo, X. Hu, H. Bin, Deep learning in edge of vehicles: exploring trirelationship for data transmission. IEEE Trans. Ind. Inf. (2019). https://doi.org/10.1109/tii.2019.2929740
18. Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, F. Xia, Deep reinforcement learning for vehicular edge computing: an intelligent offloading system. ACM Trans. Intell. Syst. Technol. (2019). https://doi.org/10.1109/tvt.2019.2935450
19. G. Li, Y. Liu, J. Wu, D. Lin, S. Zhao, in *Sensors*. Methods of resource scheduling based on optimized fuzzy clustering in fog computing, (2019). https://doi.org/10.3390/s19092122
20. G. Li, J. Wang, J. Wu, J. Song, Data processing delay optimization in mobile edge computing. Wirel. Commun. Mob. Comput. **2018** (2018). https://doi.org/10.1155/2018/6897523
21. G. Li, S. Xu, J. Wu, H. Ding, Resource scheduling based on improved spectral clustering algorithm in edge computing. Sci. Program. **2018**(5), 1–13 (2018)
22. T. YU, R. Lanlan, X. Qiu, Research on sdn-based load balancing technology of server cluster. J. Electron. Inf. Technol. **40**, 3028–3035 (2018)
23. S. Cai, J. Zhang, J. Chen, J. Pan, J. University, Load balancing technology based on naive bayes algorithm in cloud computing environment. J. Comput. Appl. **34**(2), 360–364 (2014)
24. J. m. Li, S. L. Hua, Q. R. Zhang, C. S. Zhang, Application of native bayes classifier to text classification. J. Harbin Eng. Univ. **24**(1), 71–74 (2003)
25. R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing towards balanced delay and power consumption. IEEE Internet Things J. (2016). https://doi.org/10.1109/jiot.2016.2565516
26. W. J. Liu, M. H. Zhang, W. Y. Guo, Cloud computing resource schedule strategy based on MPSO algorithm. Comput. Eng. **37**(11), 43–42 (2011)
27. X. Li, P. Tian, M. Kong, A new particle swam optimization for solving constrained optimization problems. J. Syst. Manag. **16**(2), 120–134 (2010)
28. S. Yi, Z. Hao, Z. Qin, Q. Li, in *Third IEEE Workshop on Hot Topics in Web Systems and Technologies*. Fog computing: platform and applications, (2015). https://doi.org/10.1109/hotweb.2015.22
29. J. Zhu, D. Xiao, Multi-dimensional qos constrained scheduling mechanism based on load balancing for cloud computing. Comput. Eng. Appl. **49**(9), 85–89 (2013)

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com