An Emergency Resource Scheduling Model Based on Edge Computing

Songyan Zhong[1], Tao He[1], Mengyu Li[2(✉)], Lanlan Rui[2], Guoxin Xia[2], and Yu Zhu[2]

[1] Network Information Department, China Aerospace Science and Industry Corporation Limited, Beijing, China
576678349@qq.com, 541629067@qq.com

[2] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
1006564282@qq.com, 1912696398@qq.com, 736070995@qq.com, 915808818@qq.com

Abstract. In recent years, more and more researches are focusing on the field of disaster emergency management, especially in emergency resource dispatching. In a disaster scenario, a control center needs to obtain a large amount of real-time information at the scene to make decisions timely and accurately. Traditional Cloud Computing has many shortcomings in terms of delay and bandwidth, and Edge Computing (EC) will be more suitable for this scenario. We apply EC to the emergency rescue to cope with the problem of latency needs. First, we propose an edge-based emergency rescue architecture that consists of three layers: Cloud Layer, Edge Layer, and Resource Layer. Based on this, we give a resource scheduling model that requires the collaboration of Cloud and Edge Layers. The Cloud Layer gives a partition for these tasks, and all sub-tasks are assigning to the edge servers to get a locally optimal solution. Finally, these solutions from different edge servers are summarized to the Cloud Layer to get a globally optimal solution. We compare our algorithm with CS-GA and VRP. The simulation results show that RSE has good performance in scheduling time and cost.

Keywords: Resource scheduling · Edge Computing · Cooperation model

1 Introduction

Frequent disasters like earthquakes have caused serious losses to lives and properties, and it is the reason why more and more attention being paid to the field of disaster emergency management [1], especially the emergency resource dispatching. Emergency resource scheduling is the core of emergency management, it mainly studies how to use intelligent decision theory and computer as aided tools to quickly and effectively get an optimal emergency resource allocation plan. Through this scheduling, the emergency resources can reach the destination as soon as possible, and the emergency rescue can be carried out as soon as possible, thus minimizing economic losses and casualties caused by disasters.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved
S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 353–366, 2019.
https://doi.org/10.1007/978-3-030-22971-9_31

With the development of information technology, emergency resource dispatching is gradually combined with advanced communication technologies to achieve better scheduling. Amounts of information exist in the disaster scene, such as road blockage, collapse, temperature, and humidity. The significant information should be collected and analyzed to avoid blind scheduling, thus saving the rescue time and effect. Therefore, constructing a real-time, high-speed information transmission system is of critical importance in emergency resource scheduling.

In a disaster scenario, it is still difficult to achieve high-bandwidth and real-time communication between the disaster scene and the rescue center with the current communication technology. As an emerging model, Edge Computing (EC) brings a new idea for reducing latency and bandwidth cost, that is deploying the server at the edge of the network. The emergence of EC realizes the decentralization of Cloud Computing, which can effectively avoid the time and bandwidth waste caused by the long-distance transmission, and also reduce the computing load of the cloud center. In this paper, we apply the idea of Edge Computing to the emergency rescue. This scene requires a quick response time and large amounts of data for making decisions. In this way, the efficiency of emergency rescue can be improved.

The contributions of the paper are as follows.

(1) Basing on the disaster scenario, we proposed a cooperating layered architecture which consists of the cloud layer, the edge layer, and the resource layer. In this architecture, a task can be divided and assigned to the edge layers for calculation.
(2) We propose an emergency resource scheduling model basing on latency and cost, which requires the collaboration of cloud and edge layers. The cloud layer receives tasks and divides them, then the divided sub-tasks can be assigned to the edge layer. The edge layer searches for a locally optimal solution for these sub-tasks within its service scope. After all locally optimal solutions are summarized to the cloud layer, the globally optimal solution is calculated, and the resource scheduling plan is completed.

The rest of this paper is organized as follows. Section 2 discusses some related work about Edge Computing, disaster rescue, and allocation algorithm. Section 3 describes the disaster rescue scene and introduces the Cooperating Layered Architecture for an emergency. Moreover, an emergency resource scheduling model is also detailed described. Section 4 gives the simulation of the mechanism we proposed and the effectiveness of the proposed method is demonstrated by simulation results. Section 5 draws our conclusions and portrayed the directions of ongoing work.

2 Related Work

There are many types of research focusing on resource scheduling and give solutions to it. Based on the single material scheduling model and multi-material scheduling model [2], a multi-rescue and multi-material emergency scheduling algorithm that satisfies the continuous emergency material consumption is proposed, and the shortest path is used in this solution. By introducing disaster emergency management [1], a different algorithm based on non-dominated sorting is proposed to solve multiple emergency

point multi-resource scheduling problems. It also proposes a CS-GA-based algorithm to minimize transportation costs and demand costs of emergency supplies. In order to minimize disaster losses, [7] proposes a scheduling model of emergency materials under uncertain travel time conditions, and the fuzzy set method is used to select the maximum satisfactory path in the fuzzy network.

Edge Computing has been applied in many fields because of its significant advantages. The typical model is Mobile Edge Computing (MEC) [8]. MEC model stresses setting up edge server between cloud center and edge devices, and the computing tasks are performed on the edge server. However, terminals are regarded as the devices with little computing capabilities, thus edge server undertakes this responsibility. Some studies [9–13] concentrate on the migration of intensive mobile computing task to the edge server of the network. Moreover, MEC is deemed to be a promising solution for handling video streaming services in the context of smart cities [14]. The valuable data can be transmitted to the application server to reduce core network traffic. Augmented Reality (AR) mobile applications can also combine the MEC because these applications have inherent collaborative properties in terms of data collection in the uplink, computing at the edge, and data delivery in the downlink [15]. Besides, in [16, 17], authors give an integration of MEC and 5G to meet the need of 5G in extensibility and latency.

Edge Computing can effectively solve the problems of delay and bandwidth, which are the outstanding problems in emergency resource scheduling. Therefore, we will discuss how to have a combination of them.

3 The Edge-Based Resources Scheduling Model

3.1 Architecture

In an emergency scene, there is an emergency center which needs a large amount of real-time information to make accurate decisions and coordinate operations. However, the transmission of road video surveillance and traffic information requires high bandwidth, and the limited conditions will slow down the decisions. Therefore, we propose an edge-based architecture to improve the performance of emergency command.

As is shown in Fig. 1, the architecture includes three layers: Cloud Layer, Edge Layer, and Resource Layer. The Cloud Layer is the emergency command and data center. Commanders in cloud center make higher-level decisions such as rescue plan, give remote training for on-site rescuers. Besides, multi-screen recording helps commanders get ongoing real-time information. Edge Layer is set at the edge of the network. For example, with Edge Computing server deployed on the base stations, some work such as video management and analysis can be handled in the Edge Layer. In this way, we can get a better exploitation of communication bandwidth. Resource Layer is the actual resources in the disaster scene. We list some parts such as road video surveillance, traffic information, and mobile emergency, in fact, disaster rescue is a complex process and not restricted to these parts. How to allocate these resources will influence the progress of rescue work. In the following part, we describe this issue and propose our method.

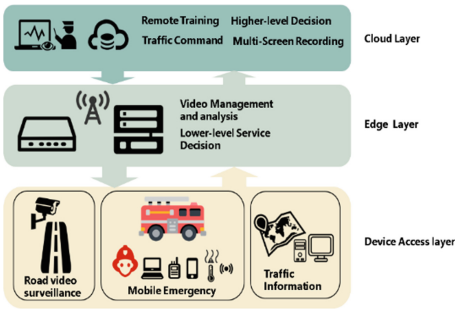

Fig. 1. Edge-based layered emergency architecture

3.2 Problem Model

The proposed emergency architecture can be seen as a typical use case of Mobile Ad hoc Network (MANET). The sensors, wireless devices, and rescuers with smart devices can communicate with each other which are not relying on a preexisting infrastructure, such as routers in wired networks or access points in managed (infrastructure) wireless networks.

We give a resource scheduling model of our scenario first. Variety of services exist in emergency rescue scenarios, such as emergency rescue, material dispatch. We define the service as a set $S = \{S_1, S_2, \ldots S_m\}$, where m is the total number of services. Resources/Devices at the rescue scene are defined as $D = \{d_1, d_2, \ldots d_r\}$, where r is the total number of resource types. Each service has various resource demands. For example, emergency services require rescuers, search and rescue tools, communication equipment, and environmental information. Therefore, a service S_i can be divided into multiple sub-services. It can be expressed as: S_i is composed of many sub-services such as $s_1, s_2, \ldots, s_n$, and each sub-service can be regarded as an atom, and its demand for resources will be single. For example, the above rescue task can be divided into 6 sub-tasks, acquiring environment information (s_1), dispatching rescuers (s_2), dispatching vehicles (s_3), scheduling search and rescue tool (s_4), scheduling communication device (s_5), and on-site rescue (s_6). Each sub-service corresponds to 0 or 1 resources, such as sub-service s_1 requires resource d_1, d_1 represents sensor information, sub-service s_2 requires resource d_2, and d_2 represents rescuer. The serial number of sub-service and resource does not need to be one-to-one. We only need to find the best resource among the multiple resources to complete the scheduling, which will be our next task.

As is shown in Fig. 2, the top level is the Service Layer, which describes all the services S in a rescue scenario. The middle layer is the Atomic Service Layer, which can be expressed as a set $s = \{s_1, s_2, \ldots s_n\}$. This layer is a division of top-level service. A node in a top layer is corresponded to one or more Atomic Service Layer nodes, and there is a certain order of execution among these atomic services. For example, the environmental inspection task s_1 needs to be executed first to ensure that the rescue can be performed, and resource scheduling can be performed after s_1 has been completed. We can use topological sorting to find the execution order of these atomic services, ensuring the normal interaction between subservices. The bottom layer is the Resource

Layer, which displays all the resources that can be scheduled at the rescue site, including rescue team, materials, intelligent terminals, sensors, and rescue equipment.

Our aim is to find the best combination of resources that can work together to support a service and ensure the quality of service, and next, we will elaborate on how to find the mapping from Service Layer node to a Resource Layer node group.

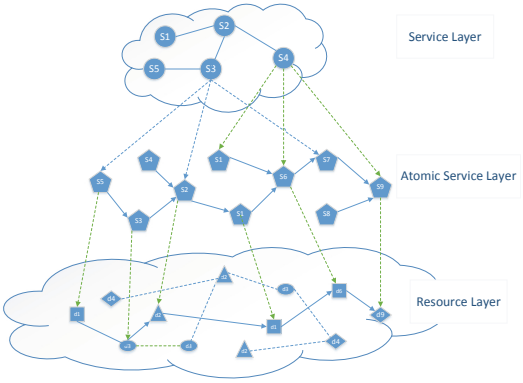

Fig. 2. Three-layer service model

3.3 Resource Scheduling Algorithm

Main Idea

We propose a resource scheduling model that requires the collaboration of cloud and edge layers. When a task arriving at the command center in the cloud layer, the task will be divided into sub-tasks, and the division is expressed in Fig. 2. And then, these sun-tasks are assigning to the edge servers to search for the Locally Optimal Solution (LOS). Finally, these solutions from different edge servers are summarized to the cloud center to get a Globally Optimal Solution (GOS). The process of our algorithm is shown in Fig. 3.

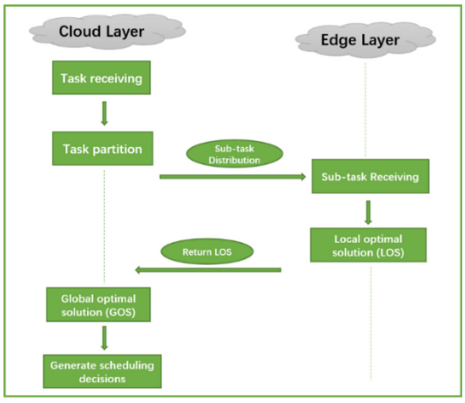

Fig. 3. The process of scheduling strategy making

Service Layer—Atomic Service Layer

From the Service Layer to the Atomic Service Layer is a division of a complete task, which is mainly determined by the control center located in the cloud layer. The principle of division is the independence of tasks. We need to ensure that the requirements for resources/equipment of each sub-task are single after dividing. Each service can be expressed as $S_i(P_i, D, L_i, C_i)$: service location P, resource requirement D, delay demand L, cost requirement C. The mapping from Si to a set of atomic services $\{s_j(P_i, d_j, L_i, C_i)\}$ is done by a set of defined partitioning rules, $R_i = \{r_1, r_2, \ldots r_n\}$. R_i is a set of n-dimensional vectors, where n is the total number of atomic services, and R_i is Boolean, indicating whether the partition contains the sub-service. For example, service S_1 consists of atomic services s_2, s_3, s_4, then $R_i = \{0, 1, 1, 1, 0 \ldots 0\}$.

$$S_i(P_i, D, L_i, C_i) \xrightarrow{R_i(r_1, r_2 \ldots r_n)} \{s_j(P_i, d_j, L_i, C_i)\} \tag{1}$$

After the cloud center completes the task partition, the sub-task set $\{s_j(P_i, d_j, L_i, C_i)\}$ will be dispatched to the edge servers to find the corresponding resources and devices. Due to the limited coverage of edge servers, each edge server can only find the optimal resources for the tasks within the corresponding service scope. Therefore, each edge server can provide a LOS for the sub-task set $\{s_j(P_i, d_j, L_i, C_i)\}$, and solve the GOS by summing the LOS to the cloud center.

Atomic Service Layer—Resource Layer

Works on the Cloud Layer

Servers at the edge layer each maintain a table named res_tab, which counts for the resources within the service scope, including sensors, videos, personal information, and materials. After the sub-service collection $\{s_j(P_i, d_j, L_i, C_i)\}$ arrives at the edge server, each server looks for the best resource combination for these sub-services in the current scope (Table 1).

Table 1. An example of res_tab.

res_id	Type	Position	Cost	Speed
d_1^1	d_1	p_1	c_1	v_1
d_1^2	d_1	p_2	c_2	v_2
…	…	…	…	…
d_n^1	d_n	p_n	c_n	v_n

A formula defined in (2) gives the locally optimal solution $F(s_j)$ of the sub-service s_j. d_j^k represents all resources of type d_j in the scope, $\rho_{i,m}$ is the mth quality indicator of service s_j, and $\alpha_{i,m}$ is the weight of the mth quality indicator of service s_j. We choose scheduling latency and cost as our QoS indicators.

$$\mathrm{F}(s_j) = \min\left\{ \mathrm{d}_j^k * \sum\nolimits_m \alpha_{j,m} * \rho_{j,m} \right\} \tag{2}$$

The formula (2) can be redefined as formula (3):

$$s.t. \begin{cases} \mathrm{F}(s_j) = \min\left\{ \rho_{j,1} * (dis|P_i - sp_k|)/sv_k + \rho_{j,2} * sc_k \right\} \\ |P_i - sp_k|/sv_k \le L_i \\ \frac{|P_i - sp_k|}{sv_k} * sc_k \le C_i \end{cases} \tag{3}$$

Where $dis|P_i - p_k|$ calculates the distance between the target point and the current resource point, c_k is the unit transportation cost of s_j, v_k is the moving speed of s_j, L_i is the highest delay that can be tolerated, and C_i is the highest cost.

For each task s_j in the sub-task set $\left\{ s_j\left(P_i, d_j, L_i, C_i\right) \right\}$, we find an optimal solution in the current server scope. LOS of all edge servers are summarized in the cloud layer to find the GOS.

Works on the Edge Layer

It is mentioned in the problem model that there are certain interactions between these sub-tasks. Therefore, when we look for GOS in the cloud layer, the scheduling sequence we find needs to satisfy the sequential execution relationship of the sub-tasks. We present a solution based on topological sorting.

We can obtain an adjacency matrix of the logical graph $G_i(s, e)$ of these sub-services at the Atomic Service Layer, and we need to find a solution that meets the task constraints from the start to the end of the critical path.

$$G_i(s, e) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{4}$$

According to the logic diagram $G_i(s, e)$, we obtain the topological sorting sequence $\mathrm{Q}_i(s_{j-1}, s_j)$ derived from the graph $G_i(s, e)$, and Q_i records each node and its corresponding precursor node. Based on this sequence and the quality parameter requirements, we can generate a resource set $D_i = \{\mathrm{V}, \mathrm{Gbit}\}$ of the execution service. V is a set defined as $\mathrm{V} = (d_1, d_2, \ldots d_{\mathrm{j}})$. d_{j} is the resource of the service s_j, and the corresponding Gbit is used for identification.

For example, we get an execution sequence (S_1, S_2, S_3, S_4), where the starting node for topological sorting is S_1. If $\mathrm{V} = (d_1, d_2, d_3, \mathrm{NULL})$ and $\mathrm{Gbit} = 1110$, it means that there are three resources $d_{k_1}, d_{k_2}, d_{k_3}$ satisfy the services S_1, S_2, S_3, and the service S_4 is not found yet. Resources. Therefore, at the beginning, D_i is set to the initial value, such as $\mathrm{V} = (\mathrm{NULL}, \ldots, \mathrm{NULL})$ and $\mathrm{Gbit} = 0\ldots0$.

The selection process of resources is as follows. The locally optimal solution $d_j \in U$, d_j returned by the edge server has three parameters, and each resource item $d_j(p_j, l_j, c_j)$ records its position parameter, delay parameter and cost parameter. The selected GOS node is stored in the set V, and the record type is $d_j(p_j, L_i^j, C_i^j)$. Starting

from the initial service of the topological sorting sequence, each type of service uses the formula (5) to select the best resource among the currently available resources, selects it and puts it into the set V, and updates the current residual delay L_i^j and the remaining cost C_i^j. The non-first node taken from the topological ordering will refer to the remaining resource status of its predecessor node, and use its updated L_i and C_i as thresholds. This iterative calculation is terminated until the entire topology sorting sequence is completed, or the constraints are not met. If the optimal scheduling is not found during the execution of the algorithm, according to the scenario set in this paper, the algorithm will find a relatively optimal resource scheduling scheme without setting the cost and delay threshold. We use the following algorithm to describe how the cloud layer looks for the optimal set of resources (Table 2).

$$\begin{cases} \mathrm{E}(s_j) = \min\{\rho_{j,1} * l_j + \rho_{j,2} * c_j\} \\ l_j \le L_i^{j-1} \\ l_j * c_j \le C_i^{j-1} \end{cases} \tag{5}$$

Table 2. Algorithm of scheduling.

Algorithm

Input U, Q_i, L_i, C_i
Output V
While $Q_i \neq$null
  take out s_j at the head of Q_i.
  find the best d_j in set U using (3)
  update: $L_i^j = L_i^{j-1} - l_j$
    $C_i^j = C_i^{j-1} - c_j$
  add $d_j(p_j, L_i^j, C_i^j)$ to set V.
  If $L_i^j < 0$ or $C_i^j < 0$
    Break
If $Q_i \neq$null
  For s_j in Q_i
    Find the best d_j in set U using
      $\mathrm{E}(s_j) = \min\{\rho_{j,1} * l_j + \rho_{j,2} * c_j\}$
    Add $d_j(p_j)$ to set V
Return V

Finally, we find the mapping from the task set $\{s_j(P_i, d_j, L_i, C_i)\}$ of the Atomic Service Layer s to the resource set $\{d_j(p_j, l_j, c_j)\}$, completing the resource scheduling from Service Layer S to Resource Layer D.

4 Evaluation

In this section, we give a simulation scenario according to [3, 4]. The experiment uses MATLAB R2017b as experiment platform, which is suitable for Windows 10, CPU Intel 2.5 GHz, 8 GB. The simulation runs for 30 min. We give three times experiments to avoid errors.

We construct a simulation area as 10 * 10 (km^2). There are two emergency rescue centers in the area with coordinates (0, 0) and (10000, 10000). The rescue center has full kind of resources, but it is far away from the rescue point; the coordinates and resources of the rescue center are shown in Table 3. There are scattered resource distributions in other locations throughout the region. We use a random function to distribute periodically. The random function is shown in Eq. (6), and the distribution period is set to 5 min, and 5 resources are distributed each time. We set up a rescue zone with a center (5000, 5000) and a radius of 4000, in which a service request will be generated. The task is generated every 3 min and 10 task requests will be generated during the simulation time. The random function that generates the task satisfies Eq. (7) (Fig. 4).

Table 3. Settings for the resource center.

Name	X-coordinate	Y-coordinate	Resources
res_center1	0	0	$s_1, s_2, s_3, s_4, s_5, s_6, s_7$
res_center2	10000	10000	$s_1, s_2, s_3, s_4, s_5, s_6, s_7$

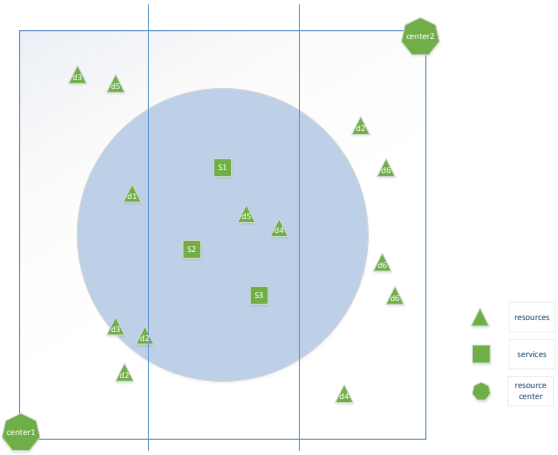

Fig. 4. Simulation area explanation

$$\begin{cases} res\_type = rand\_1()\%7 + 1 \\ x = rand\_2()\%10000 \\ y = rand\_3()\%10000 \end{cases} \tag{6}$$

$$\begin{cases} ser\_type = rand\_4()\%3 + 1 \\ x = rand\_5()\%10000 \\ y = rand\_6()\%10000 \\ (x - 5000)^2 + (y - 5000)^2 \leq 4000^2 \end{cases} \tag{7}$$

In the experiment, three edge servers and one central server are set. The edge server has an ideal service range and is divided into three areas. The corresponding x-axis range is (0, 3333), (3333, 6666), (6666, 1000), the edge server can count all resources in the area, maintain a resource table and update periodically. There are three types of rescue services in the simulation scenario. Each type of service, sub-service, resource requirement, delay threshold, and cost requirement setting are shown in Table 4. (Note: v_j and c_j are 0, indicating that the service is a network service. For example, scheduling sensor information), the scheduling cost information of specific resources is shown in Table 5.

Table 4. Settings for the services.

service_type	sub_service	res_type	L_i(h)	C_i
S_1	s_1, s_2, s_3, s_4	d_1, d_2, d_3, d_4	0.5	20
S_2	s_2, s_3, s_5, s_6	d_2, d_{36}, d_5, d_6	1	15
S_3	s_1, s_4, s_5, s_6, s_7	d_1, d_4, d_5, d_6, d_7	0.3	30

Table 5. Settings for the resources.

res_type	trans_cost(c_j)	trans_speed(v_j)
d_1	0	0
d_2	50	30
d_3	80	35
d_4	40	20
d_5	60	15
d_6	90	20
d_7	130	10

We calculate the actual scheduling time of all tasks LL_i, and record the ratio of LL_i to the task's threshold time L_i as $P_L = L_i/LL_i$, which is a parameter of our comparison experiment. On the other hand, we calculate the actual scheduling cost of all tasks CC_i, and record the ratio of CC_i to the threshold cost C_i of the task as $P_C = C_i/CC_i$, which is another parameter. We calculated data for different tasks in three experiments and

recorded them in Table 6. In order to avoid the randomness of the experiment, we will use the random letter to generate the data needed for the experiment in advance, and use the same value for the three experiments.

Table 6. Result of resource scheduling.

Index	S_1	S_3	S_1	S_2	S_2	S_3	S_1	S_2	S_3	S_3
P_L	1.32	1.21	1.37	1.16	1.10	1.16	1.28	1.18	1.15	1.08
P_C	1.35	1.31	1.41	1.20	1.14	1.20	1.30	1.24	1.23	1.12

From the data, it can be found that two service requests with the same type are too concentrated in a time, the optimal schedule may have been used by the previous schedule, and the performance of the second schedule will be significantly lower. For example, the last two columns of data in Table 6, the same type of request S_3, the former is numerically superior to the latter.

We apply the same environmental settings to the algorithms proposed in [3, 4] and give a comparison of these results. [3] identified as CS-GA proposed a decision-making algorithm based on CS-GA to minimize the transportation cost and demand cost of emergency materials. Another genetic algorithm optimization path based on an emergency distribution VRP model in [4], we identify it as VRP. Our algorithm is identified as RSE in the comparison chart.

In comparison, it can be found that P_L and P_C values of the RSE are relatively high, which shows that the actual scheduling time LL_i and the actual scheduling cost CC_i of RSE are lower, which is due to the distributed edge cloud can quickly sense the resource update status within the service scope and obtain more accurate resource status, thus achieving a more accurate scheduling scheme than the remote centralized control of the cloud (Figs. 5 and 6).

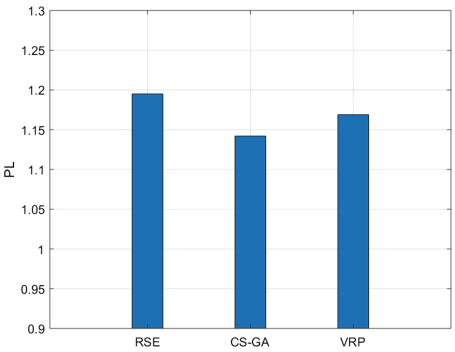

Fig. 5. Comparison of P_L

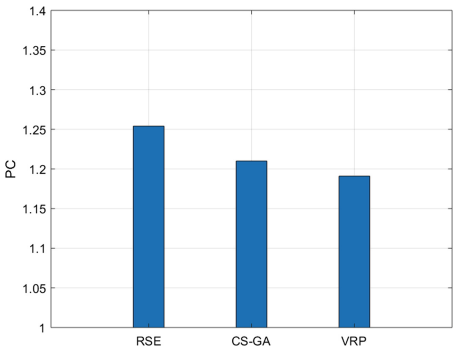

Fig. 6. Comparison of P_C

We adjust the service request rate and give a comparison again. The above results are basing on a service request rate of 1/3 (10 requests every 30 min), and then we give a comparison with changing rates from 1/6 to 1. From Fig. 7, it can be found that three algorithms have a decreasing P_L in the case of increased rate, and the drop is obvious at 0.0667, but the performance of RSE is still better than the other two. In Fig. 8, P_C of the three algorithms also decrease with increasing rate, but the decrease of P_C is not particularly obvious, because the transportation cost does not fluctuate too much within the limited range determined. The performance of RSE is still the best among them.

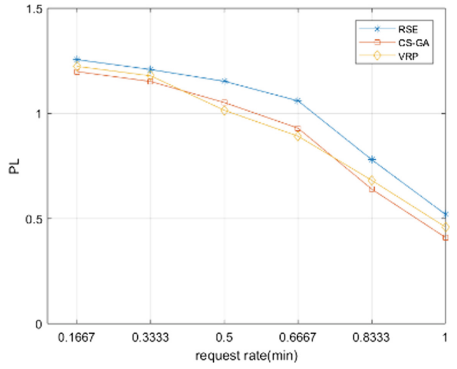

Fig. 7. Comparison of P_L

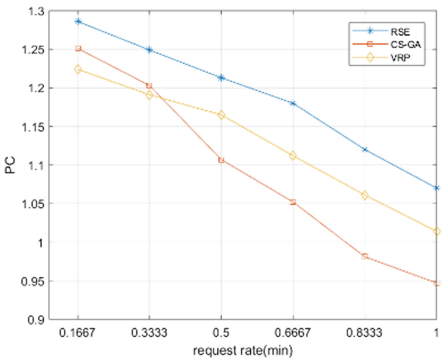

Fig. 8. Comparison of P_C

5 Conclusion

Current emergency communication systems are facing multiple challenges, such as insufficient bandwidth and delayed response, which seriously slow down the emergency scheduling and rescue. The idea of Edge Computing is suitable for these problems. We propose an edge-based architecture to realize an emergency system which can assign some process from the cloud center to the edge of the network. Then, based on this architecture, we give a resource scheduling model which takes latency and cost into consideration. Through the collaboration of cloud and edge layers, the algorithm can achieve a significant performance on scheduling time and transmission cost. However, this model is focusing on a single-service scenario, when it comes to multi-service and multi-resource scenarios, the algorithm should be improved to adopt these factors. The following work is to optimize RSE to fix the scenario of multi-service. Further, we will enrich the content of our architecture in more situations.

References

1. Zhang, T.: Research on emergency resource scheduling in disaster emergency management. Hefei University of Technology (2015)
2. Chai, X.R.: Research on disaster emergency rescue material dispatching system. University of Science and Technology of China (2009)
3. Cai, J.M.: Research on time-varying and reliability related issues of earthquake disaster emergency logistics. Central South University (2012)
4. Zhao, T.: Research on optimization of emergency disaster material distribution system for sudden natural disasters in China. Dalian Maritime University (2011)
5. Xue, H., Cai, B., Jiang, C., Yuan, Y.: Research on the robust decision of emergency material distribution for chain retail supply chain. In: Chinese Control and Decision Conference (CCDC), Shenyang, pp. 1922–1927 (2018)
6. Gu, Y.: Study on scheduling model of emergency materials distribution after natural disaster. In: 2nd International Workshop on Intelligent Systems and Applications, Wuhan, pp. 1–4 (2010)

7. Sun, Q., Kong, F., Zhang, L., Dang, X.: Study on emergency distribution route decision making. In: International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), Jilin, pp. 338–341 (2011)
8. Satria, D., Park, D., Jo, M.: Recovery for overloaded mobile edge computing. Future Gener. Comput. Syst. **70**, 138–147 (2017)
9. Zhao, T., Zhou, S., Guo, X.: A cooperative scheduling scheme of l cloud and Internet cloud for delay-aware mobile cloud computing. In: Proceedings of IEEE GLOBECOM Workshops, pp. 1–6 (2015)
10. Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., Buyya, R.: Mobile code offloading: from concept to practice and beyond. IEEE Commun. Mag. **53**, 80–88 (2015)
11. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**, 2795–2808 (2016)
12. You, C., Huang, K., Chae, H., Kim, B.H.: Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wirel. Commun. **16**, 1397–1411 (2017)
13. Sardellitti, S., Scutari, G., Barbarossa, S.: Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inf. Process. Over Netw. **1**, 89–103 (2015)
14. Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., Flinck, H.: Mobile edge computing potential in making cities smarter. IEEE Commun. Mag. **55**, 38–43 (2017)
15. Lin, S., Cheng, H.F., Li, W., Huang, Z., Hui, P., Peylo, C.: Ubii: physical world interaction through augmented reality. IEEE Trans. Mob. Comput. **16**, 872–885 (2017)
16. Hu, Y., Patel, M., Sabella, D., et al.: Mobile edge computing: a key technology towards 5G. ETSI White Paper (2015)
17. Zhang, K., et al.: Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. IEEE Access J. **4**, 5896–5907 (2016)