A Novel Hybrid Model for Task Dependent Scheduling in Container-based Edge Computing

Tingting Lu*, Fanping Zeng*†, Guozhu Chen*, Wenjuan Shu*, Jingfei Shen* and Weikang Zhang*
*School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China
†Anhui Province Key Lab of Software in Computing and Communication, Hefei, Anhui, PR China
{tingtlu, chengz18, shuwj, ericjeff, buttman}@mail.ustc.edu.cn, billzeng@ustc.edu.cn

2021 IEEE International Conference on Communications Workshops (ICC Workshops) | 978-1-7281-9441-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICCWorkshops50388.2021.9473877

***Abstract*—In traditional edge computing, the task from the Internet of Things (IoT) is usually offloaded to edge server. It will be uploaded to the remote cloud if the edge server cannot process it. A task can be processed on the server, only if the server has configured the corresponding function program. However, each edge server can only configure a small number of functions due to the limited computing, storage, and bandwidth resources. Moreover, modern tasks from IoT devices become more and more diverse, which are also accompanied by complex dependencies. It increases the processing time overhead to the task processed in remote cloud due to huge transmission delay. In this paper, we design a container-based edge computing system, where a task can be executed on a server only if the server has configured the corresponding container, if not the server can fetch it from other edge servers or remote cloud. Based on the system, we propose a novel hybrid model, called CBASGA, with the aim to minimize the job complete time, which combines Chaos-based Beetle Antennae Search and Genetic Algorithm. Our experimental results show that the designed system reduces the average job completion time by 4.2% compared with the comparison system, and CBASGA reduces the average job completion time by at least 21.7% compared with baselines.**

***Index Terms*—Task scheduling, Container configuration, Edge computing**

I. Introduction

Edge computing (EC) provides resource services (e.g., computing, storage, bandwidth, etc) to nearby users by deploying small servers (called *edge server*) in the edge of the network [1]. Nearby users can offload their tasks or jobs to edge server for processing, so that the data transmission time can be greatly reduced.

Due to that the application pre-configured by the developer in edge server is limited, the edge server can only process a few tasks. It can not take full advantage of edge computing. So some scholars have proposed containerizing the applications required for IoT devices, which can be automatically obtained from the cloud when there is no corresponding containerized application for processing task [2]. Furthermore, Martin Fowler and James Lewis jointly propose the concept of microservices and define a microservice as a small service composed of a single application, which can be deployed, scaled, and tested independently [3]. Many works have applied the idea of microservices to edge computing [4, 5]. But there are still many challenges in applying microservices to EC.

First, the storage and computing resources of edge services are limited. Only a few containers (i.e., microservices) can be configured in the edge server. That is to say, if a task is planned on an edge server that has no corresponding container, the server can configure the container which is downloaded from the remote cloud. After the container has been configured, their server can process the task. Therefore, it is a key challenge to design a task scheduling algorithm that considers the container configuration in EC.

Second, the application (also called a job) from IoT device consists of some tasks that have complex dependencies, each of which needs a suitable container. Specifically, the communication among servers will occur between dependent tasks when they are allocated on different servers respectively. Furthermore, because of the dependency relationship, the start time of task will be limited by its direct predecessors. In other words, a task can only be executed after all of its direct predecessors have been completed. So how to deal with the complex inter-task dependency is another challenge in EC. There have been many researches on task scheduling and related server configuration (e.g., [6, 7]). But they consider either scheduling the whole job individually [7, 8] or configuring the related servers independently [9].

In this paper, we first propose a container-based edge system in which containers can be transferred between edge servers. The edge servers have limited resources and each server has a different performance. We then propose an efficient approximation algorithm, named CBASGA. The algorithm combines Chaos-based Beetle Antennae Search (CBAS) and Genetic Algorithm (GA) in addition to heuristic initialization to optimize the tradeoff between task dependency and container configuration. Finally, we adopt a cluster trace from Alibaba to evaluate CBASGA. The simulation results show that the designed system is efficient to process dependent tasks and CBASGA has excellent performance.

The remainder of the paper is organized as follows. Section II expounds our edge system model. The task scheduling and correlation allocation algorithms are shown in section III. Section IV presents the performance evaluation for CBASGA. We conclude this paper in section V.

II. System Model and Problem Formulation

A. Edge System Model

The edge system consists of K heterogeneous edge servers and one remote cloud, denoted as $S = \{s_0, s_1, ..., s_K\}$, where s_0 depicts the remote cloud. We use $r_{k,k'}$ as the data transfer

978-1-7281-9441-7/21/$31.00 ©2021 IEEE

rate between server s_k and $s_{k'}$, and assume $r_{k,k'} = r_{k',k}$. Therefore, the transmission time of ω unit data from s_k to $s_{k'}$ ($k \neq k'$) is $\frac{\omega}{r_{k,k'}}$. If $k = k'$, the transmission time is set to 0, that is, $r_{k,k} = +\infty$, as the communication time between tasks within the server is usually very small.

Each server has a different maximum number of container configurations simultaneously, which is called *the server capacity*, denoted as $C = \{c_0, c_1, ..., c_K\}$, where c_0 depicts the maximum number of containers owned by the remote cloud simultaneously, other c_i $(i = 1, 2, ..., K)$ represents the maximum number of containers owned by the i-th edge server simultaneously. In particular, when the number of containers configured by a server reaches its maximum, the server can use a policy to free up some of the resources occupied by the existing containers to configure the new containers. The policy is the least recently used (LRU) algorithm in our paper.

To better illustrate our system model, we present the designed edge system in Fig. 1. The system consists of three edge servers, each of which is responsible for job requests in its own domain, and one cloud. The capacity of each edge server is set to two, whereas the cloud has all containers. A job request consists of two hidden tasks (virtual task created for scheduling convenience) and five ordinary tasks (actual task to be processed) are sent to server s_2 for execution. After scheduling, two hidden tasks (task 0 and task 6) and two ordinary tasks (task 2 and task 5) are place on s_2, task 1 and 3 are on s_3, and task 4 are on s_1. Specifically, s_2 is not configured the container that can process task 5, so it fetches the corresponding container from edge server s_1 which holds the container. Since hidden task 0 and task 2 are on the same server which has the container needed for task 2, so task 2 can be executed immediately. As the initial data is sent from s_2 to s_3, s_3 can process task 1 after configuring the corresponding container from the cloud. As there is no precedence constraint between task 1 and 2, they can be processed in parallel. After receiving the outputs of task 2, task 3 and 4 can be executed on s_1 and s_2 respectively. After obtaining the processing results for task 1, 3 and 4, s_2 can execute task 5 and generate the final result to the hidden exit task 6. It is worth noting that the two tasks with dependencies (such as task 2 and task 3) are not processed on the same server because the scheduler estimates that the two tasks can be completed earlier if performed separately.

B. Request and Job

The job request from the device is first submitted to the nearest edge server, called *the local server*, and then sent to the edge system. The job consists of a set of dependent tasks, denoted as $\{v_1, v_2, ..., v_J\}$. Specifically, *the entry task* is a task without any predecessor task and *the exit task* is a task without any successor tasks. In order to facilitate unified scheduling, we add *a hidden entry task* (v_0) and *a hidden exit task* (v_{J+1}). Let *the hidden entry task* connect to each actual entry task and *the hidden exit task* connect to each actual exit task. It's important to note that the completion of

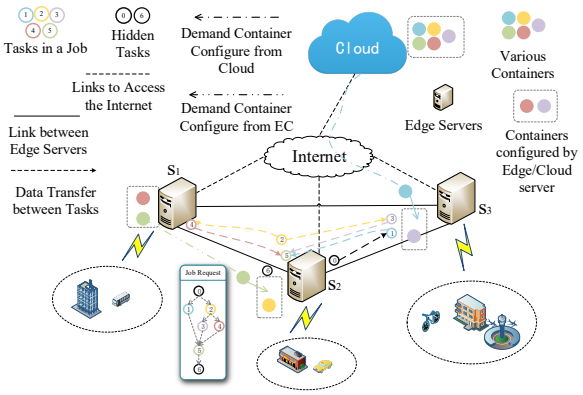

Fig. 1: System Model

the hidden exit task implies the completion of a job. We model them as directed acyclic graph (DAG), denoted as G(V, E, W). Here, $V = \{v_0, v_1, ..., v_{J+1}\}$, each node represents a task. An directed link e = $<v_i, v_j>$ indicates that there is a dependency between v_i and v_j, that is, the task v_j cannot be processed until task v_i has completed. E is the set of all directed links. The link weight w_{ij} are determined by the amount of data transferred from v_i to v_j. W is the set of all link weight. The processing time of a task v_j on server s_k is denoted as $p_{j,k}$. Considering that redeploying a task can be consuming a lot of time, we don't allow a task to preempt the server to avoid additional processing costs.

C. Container Configuration

In this paper, we assume that each type of task has one and only one container that can execute it. We define a task container map function, $map(\cdot)$, to represent the container needed for the task. In other words, if the task v_j is processed on a server s_k, it must have configured the corresponding container $\tau_j = map(v_j)$. If not, the server has to suffer configuration time $\Delta_{\tau_j,k,x}$, which depends largely on the container transfer time over the network to fetch the container from surrounding edge server or the cloud s_x.

III. Algorithm

Before introducing the algorithm, let's explain the encoding method of the scheduling solution. Our encoding solution indicates each task of a job is allocated to which servers. For example, Fig. 2 shows the encoded solution of the tasks of the job. We take the encoded solution in the upper left corner of the figure as an example, the value of t_0 and t_6 is 1 presents the hidden tasks are allocated to local server s_1. The value of t_2 and t_4 is 2 indicates task 2 and 4 are assigned to server s_2 and so on.

The proposed algorithm includes three important parts. The first part is a heuristic algorithm for initialization. The second part is the automatic gene combination realized by CBAS. The third part is CBASGA algorithm which improve GA with results from the first and second parts.

A. Initial Task Allocation

During the initialization phase, we greedily select the server for each task, which allows the earliest completion of a task among all servers. Specifically, the assigned sever of the hidden exit (entry) task is the local server in our system. The details of the initial task allocation are shown in Algorithm 1.

Specifically, a job G is released to local server s_l. The two hidden tasks (v_0 and v_{J+1}) will be placed and executed on s_l. Before the initialization starts, we need to prioritize the tasks. So we first find the longest directed path from the task concerned to the exit task, and calculate the sum of the task processing time and the communication time between tasks on the longest path [10]. Then, according to the decreasing order of the calculation result, we can prioritize these tasks (Line 1). The earliest finish time of task v_j on server s_k is denoted as $F_{j,k}$, which is set to $+\infty$ initially (Line 2). The value of $F_{0,l}$ is 0 because task v_0 is virtual (Line 3). With corresponding container configuration, for each direct predecessor task v_i of task v_j, the value of $F_{j,k}$ is equal or greater than the sum of the finish time of v_i, the communication time between v_i and v_j and the processing time of v_j on s_k, where we make $F_{j,k}$ equal to the aforementioned maximum sum (Line 8). Without corresponding container configuration, we need to add additional container configuration time from the server with the corresponding container to s_k (Line 11). We place task v_j on the server with minimum $F_{j,\cdot}$ after one iteration (Line 14-15). With the acquisition of the earliest completion time of each task, we end up with $F_{J+1,l}$, which also is the earliest completion time of the job. From this process, we can know where each task is assigned, and then encode it to get the initial solution.

B. Chaos-based Beetle Antennae Search Algorithm

The original Beetle Antennae Search Algorithm (BAS) [11] algorithm has good optimization performance and convergence in continuous space, but it's not suitable for discrete space. To solve this problem, we discretize the model and then propose the Chaos-based Beetle Antennae Search Algorithm. For a job, we first generate a random direction vector (i.e., a scheduling solution) in the given system as follows:

$$\vec{b} = rndi(K, J) \tag{1}$$

where *rndi(K,J)* represents a function. The function can randomly generate an *J*-dimensional vector, each dimension of which is an integer value not exceeding *K*. After obtaining the scheduling solution $\vec{b}$, the position of the left-tentacles and right-tentacles are defined as below:

$$x_l = x^t - d^t\vec{b} \tag{2}$$

$$x_r = x^t + d^t\vec{b} \tag{3}$$

where x^t is the position after the t-th fly of beetle. d^t is the sensing length of antennae. Then, we further generate iterative model which determines the direction and distance of the beetle at the next time, as follows:

$$x^t = x^{t-1} + \delta^t\vec{b} \cdot sign(f(x_r) - f(x_l)) \tag{4}$$

Algorithm 1 Initial Task Allocation Procedure

Input: $G(V, E, W), S$ with configured containers, local server s_l

Output: *initial solution*

1: assume $v_0, v_1, ..., v_{J+1}$ are the task precedence of G.
2: $F_{j,k} := +\infty$ for $0 \le j \le J+1$ and $0 \le k \le K$.
3: let $F_{0,l} := 0$.
4: **for** $j = 1$ *to* J **do**
5: **for** $k = 0$ *to* K **do**
6: find P_j, $P_j = \{(v_i, s_y)|\ v_i$ is a direct predecessor task of v_j, s_y is the server to which v_i is assigned$\}$
7: **if** server s_k with container $map(v_j)$ **then**
8: $F_{j,k} = \max_{(v_i,s_y)\in P_j}\{F_{i,y} + \frac{\omega_{i,j}}{r_{y,k}} + p_{j,k}\}$
9: **else**
10: find $S'_{map(v_j)}$, $S'_{map(v_j)} = \{s_z|\ s_z$ is the server with container $map(v_j)$ in edge server or cloud$\}$.
11: $F_{j,k} = \max_{(v_i,s_y)\in P_j, s_z\in S'_{map(v_j)}}\{F_{i,y} + \frac{\omega_{i,j}}{r_{y,k}} + p_{j,k} + \Delta_{map(v_j),k,z}\}$
12: **end if**
13: **end for**
14: let u = $\arg\min_{0\le k\le K} F_{j,k}$
15: assign task v_j to server s_u
16: **end for**
17: $F_{j+1,l} = \max_{(v_i,s_y)\in P_{j+1}}\{F_{i,y} + \frac{\omega_{i,j}}{r_{y,l}}\}$
18: construct the initial solution from $F_{J+1,l}$.
19: **return** *initial solution*

where δ is the size of the search step; *sign(·)* represents a sign function; *f(·)* is a function to be optimized. Our goal is to minimize job completion time (i.e., *makespan*), so we use the reciprocal of the makespan as the function.

$$f(x) = \frac{1}{makespan(x)} \tag{5}$$

With iteration, the step size gradually decreases for a more refined search and the distance vector is adjusted with it, they are shown as follows:

$$\delta^t = h\delta^{t-1} \tag{6}$$

$$d^t = \delta^t / c \tag{7}$$

where *h* is coefficient of the step size, *c* is a constant preferably between 2 to 10.

Note that some vectors are operated to be non-integer, such as x_l and x_r, in order to make them operate in *f*, we deal with them as follows:

$$x' = |x'|\ mod\ K \tag{8}$$

the formula is to convert the non-integer number x' into an integer number that can indicate server identification.

The details of the CBAS are shown in Algorithm 2. More importantly, the performance of BAS after discretization is poor, so we alleviate it by adding a logistic chaotic map in the first $iter/2$ iterations (Line 8-12) [12]. Specially, the purpose of generating the chaos value from a deterministic nonlinear

system is to locally adjust x^t regularly (Line 11-12). From the experiment, we find that the algorithm is non-convergence. Fortunately, it doesn't affect the purpose of obtaining a better solution. Finally, we update the current optimal scheduling.

Algorithm 2 CBAS

Input: $G(V, E, W), K, J$, the number of iteration $iter$, f
Output: Φ

```
1: generate integer solution x using (1)
2: Let φ be the the current maximum value of the function
   f, and set φ = 0
3: for t in iter do
4:    update b⃗ using (1)
5:    update the variable δ, d^t via (6),(7)
6:    compute x_l, x_r following (2)(3), and Convert them
      using (8)
7:    compute x^t via (4)
8:    if t < iter/2 then
9:       generate integer solution x' using (1)
10:      x' ← μx'(1 − x')
11:      x^t ← x^t + δ * (x' − x^t)/2
12:   end if
13:   convert x^t using (8)
14:   if f(x^t) > φ then
15:      Φ ← x^t
16:      φ ← f(x^t)
17:   end if
18: end for
19: return Φ
```

C. CBASGA Algorithm

In this part, the task allocation solutions are called chromosomes. The details of the CBASGA is shown in Algorithm 3. We get a better chromosome from the initialization phase. But the basis of genetic operators is population, we randomly generate $N-1$ chromosomes. Then, we perform T generations, each of which comprises five operations (i.e., selection, crossover, mutation, automated gene combination, and evaluation).

The selection operation is an important part from the previous generation to the next generation. Before the t-th generation, the population $\{D_{t-1,n}\}_{n=1}^{N}$ are given fitness values obtained in the previous generation or initialization, which directly impact the probability that each chromosome will survive the selection process. We perform a proportionate roulette wheel selection to determine which chromosome survives. The selection probability function is defined as follows:

$$P_{select}(D_{t,n}) = \frac{fitness(D_{t,n})}{\sum_{n=1}^{N} fitness(D_{t,n})} \tag{9}$$

For the crossover operation, the two paternal chromosomes are selected with probability p_c, and then the crossover points are randomly selected on them. It can produce two new chromosomes when the two paternal chromosomes are crossed at the crossover point. For example, there are two parent chromosomes, which are two encoding solution of scheduling of the job with five tasks on the left side of Fig. 2. They are crossing at cross-point that is between t_2 and t_3, and the right side of the figure is the result of crossing. The crossed chromosomes are new allocation solutions, which have new fitness value respectively. From the crossed chromosomes, the mutant chromosomes are selected with a probability p_m. Then the mutation operation randomly changes the mutational site of the chromosome from one to another, trying new possibilities while keeping the good traits.

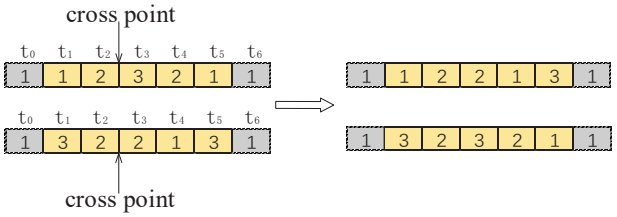

Fig. 2: An example of two chromosomes crossing

In automated gene combination, we call CBAS with related parameters. Its steps have been detailed in the previous part. After the operation, it only obtains one chromosome, but the chromosome has a high fitness. Adding the chromosome with relatively high fitness to each generation of the algorithm, it can gradually improve the fitness of the whole population.

The evaluation operation is to evaluate the fitness value of each chromosome by the fitness function to provide an indicator for the selection operation. In the same way, we still use the formula (5) as the fitness function. This means the goal of the algorithm is to find the minimum job completion time.

IV. Performance Evaluation

In this section, we use the job information data collected by the Alibaba cluster trace program over 8 days from 4,000 heterogeneous machines as our data set source [13]. Without loss of generality, we randomly choose 1000 jobs from it, each of which has between 2 and 59 tasks. Then, we use it to evaluate the designed system and CBASGA algorithm.

A. Experiment Settings

a) Edge and Cloud Clusters: Our simulations are carried out in an edge-cloud cluster, where there are 4 edge servers and a remote cloud. The number of containers of each server can hold is limited, which is set to 3 by default. Similar to [14], we generate values for data transmission time among edge servers uniformly from the interval $[5, 100]$ ms. Similarly, the processing time of each task is drawn uniformly from the interval $[10, 200]$. Considering that the data transmission to the cloud can consume a lot of latency, we set it in the range of $[100, 2000]$ ms. The container configuration times from the edge are set in $[300, 700]$ ms. In view of that configuring

Algorithm 3 CBASGA

Input: $G(V, E, W), S$ with configured containers, local server s_l, the number of generations T, the number of chromosomes in each generation N and the crossover and mutation probabilities p_c and p_m

1: get $D_{0,1}$ from Algorithm 1, and initialize population $\{D_{0,n}\}_{n=2}^{N}$ by randomly assigning tasks to each server according to the G task precedence
2: **while** $t < T$ **do**
3: Selection: *select* a new generation $\{D_{t,n}\}_{n=1}^{N}$ with a proportionate roulette wheel
4: Crossover: for each pair $\{(D_{t,2n-1}, D_{t,2n})\}_{n=1}^{\lfloor N/2 \rfloor}$, perform *crossover* with probability p_c at selected crossover point
5: Mutation: for each crossover $\{D_{t,n}\}_{n=1}^{N}$, make *mutation* with probability p_m at selected mutation point
6: Automated Gene Combination: *product* a chromosome with high fitness from CBAS
7: Evaluation: *evaluate* the fitness of the population $\{D_{t,n}\}_{n=1}^{3N+1}$ by calculating the makespan of the schedule
8: **end while**
9: **return** the schedule with the highest fitness

the container from the cloud suffers a long latency, so by default we set it as 20 times of the configuration time between two edge servers. The cloud has all containers and thus the configuration time is 0 there. The processing time in the cloud server is set 0.75 times (on average) of the processing time in edge servers, as the cloud generally provides more computing resources than edge servers. For the comparison system, we restrict it to configuring the container only from the cloud.

b) Comparison Baselines: According to the previous section, we can know CBASGA is a hybrid algorithm, which contains three common policies (the earliest completion time, CBAS and GA. To examine the performance of CBASGA, we compare CBASGA with them respectively. Specially, for the earliest completion time policy, we use OnDoc [15]. But it is not suitable for containers to be configured from the edge, we modify it to adapt our system.

B. Experiment Results

In this part, we first exhibit the simulated results of overall performance based on the 1000 jobs. Then, we conduct flexibility experiments using three specific DAGs [14], to study the effect of container configuration time, the capacity of the edge server, the communication computation ratio (CCR) and the number of edge servers.

1) The Overall Performance: Fig. 3 shows the average completion time in our system and the comparison system during the execution of the four algorithms respectively. Compared to the system which configures containers only from the cloud, the job completion times of using four algorithms are less in designed system, which reduces the average job completion by 4.02%. In addition, we can also

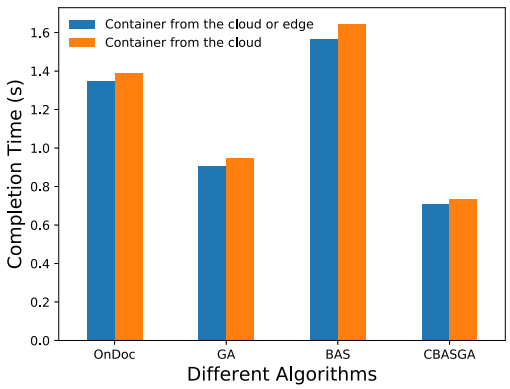

Fig. 3: Comparison of completion time in different scenes

see that our CBASGA algorithm has excellent performance in both our system and the comparison system. Table I shows that CBASGA can reduce 47.3%, 21.7%, 54.8% of the completion time on average compared to OnDoc, GA and BAS respectively in designed system (46.9%, 22.4%, 55.3% in the comparison system). Although GA and BAS are both bionic algorithms, the discretized BAS which is not convergent has lower performance. CBASGA's percentage of completed job is consistently above the three baselines.

TABLE I: Comparison of OnDoc, GA, BAS and CBASGA

Scheduler \ Time (s)	Min	Max	Mean
OnDoc	0.381	3.15	1.346
GA	0.204	5.663	0.906
BAS	0.222	6.927	1.568
CBASGA	0.164	1.763	0.709

2) Flexibility Analysis: Here, we analyze CBASGA's flexibility to the effect of multiple factors: the container configuration time, the capacity of the edge server, CCR, and the number of the edge servers.

a) Effect of the Container Configuration Time: To explore the effect of the different scale of container configuration time, we scale the configuration time from 0.2× to 4.0× of the original value while keeping the other parameters unchanged. The job completion times under different configuration time are shown in Fig. 4. As the configuration time increases, so does the completion time on the whole. The OnDoc has the worst performance, because the simple greedy strategy only considers the current optimal and loses sight of the other possible better solutions. GA's performance is generally better than BAS, because the discretized BAS is equivalent to a random selection algorithm. CBASGA is the least effected algorithm compared to the other algorithms.

b) Effect of the Capacity of Edge Server: Fig. 5 presents the completion time of the four algorithms at different edge server capacity. Except for CBASGA, none of the other three algorithms shows a significant decrease in the completion time

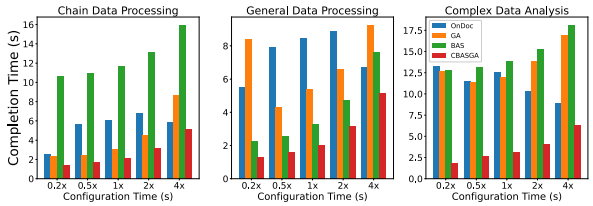

Fig. 4: Completion time of different container configuration time

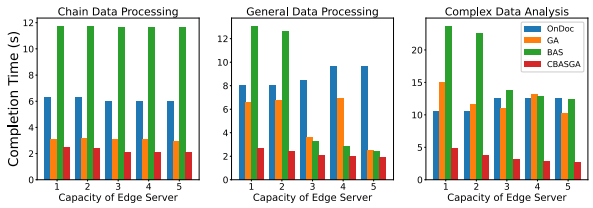

Fig. 5: Completion time of different capacity of edge servers

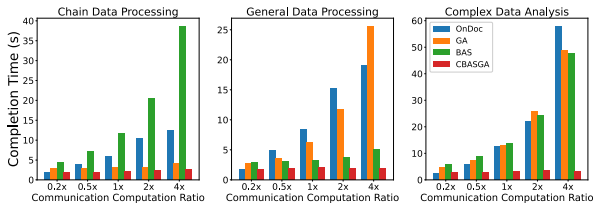

Fig. 6: Completion time of different communication computation ratio

with the increase of capacity. Since the increased capacity is matched with random containers in our experiment, the increased capacity may not be used by the task (on account of the task is executed by configuring the corresponding container). Therefore, there is no significant reduction.

c) Effect of the Communication Computation Ratio: Similar to study the effect of the container configuration time, we scale the CCR from $0.2\times$ to $4.0\times$ of the original value to explore the effect of CCR. As shown in Fig. 6, while increasing the CCR, the completion times of using three baselines are increasing gradually except CBASGA. This is due to the increased proportion of communication time, resulting in increased data transfer and container configuration time. Compared to the three algorithms, CBASGA shows that the job completion time increases little with the increase of the communication computation ratio, so it is the most efficient. That is because CBASGA tends to try to place tasks on different servers to find a better scheduling.

d) Effect of the Number of Edge Servers: Fig. 7 shows the completion time of the four algorithms as the number of edge servers grows. In our experiment, the container configuration of the new edge server is also random, which make it unlikely that the new edge server could be used by the task. In complex data analysis, we can see an overall downward trend. As the large number of tasks in complex jobs partially

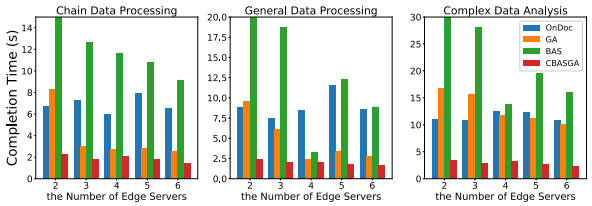

Fig. 7: Completion time of different number of edge servers

offsets this randomness. That is to say, the more tasks there are, the more likely there are to match the container of a new edge server. OnDoc fluctuates a lot because it only considers the earliest completion of the current task, and the appearance of a new edge server will affect it, resulting in changes in the final job completion. GA and BAS fluctuate under the influence of random initialization and parameters. Compared to them, CBASGA presents the best performance among the three DAGs.

V. Conclusion

In this work, we propose a container-based edge system, which can reduce significantly container configuration time by obtaining containers from nearby servers. Then, we propose a novel hybrid model to schedule tasks within independent task constraint and server capacity constraints. By purposeful initialization and automated gene combination, CBASGA can evolve directionally in a highly adaptable initialization population. Based on the real cluster trace from Alibaba, the experimental results show that the proposed algorithm can efficiently reduce the job completion time in a variety of scenarios compared with baselines. Although we used CBAS with low time complexity to construct a genetic combination, the overall time complexity of CBASGA is still relatively high. A feasible solution is to separate the calculation process of CBAS from CBASGA and make CBASGA and CBAS execute concurrently. Furthermore, the task dependencies in the experiment come from Alibaba's data-trace and how to decompose and extract the dependencies from the actual job is a new challenge in the future.

Acknowledgment

This work is supported partly by the National Key R&D Program of China 2018YFB2100303, 2018YFB0803400 and National Natural Science Foundation of China (NSFC) under grant 61772487.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint container placement and task provisioning in dynamic fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 028–10 040, 2019.

[3] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.
[4] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled iot," *IEEE Internet of Things Journal*, 2020.
[5] "Edgex foundry," https://www.edgexfoundry.org/.
[6] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.
[7] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
[8] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2287–2295.
[9] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.
[10] K. Shin, M. Cha, M. Jang, J. Jung, W. Yoon, and S. Choi, "Task scheduling algorithm using minimized duplications in homogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1146–1156, 2008.
[11] X. Jiang and S. Li, "Bas: beetle antennae search algorithm for optimization problems," *arXiv preprint arXiv:1710.10724*, 2017.
[12] B. Alatas, E. Akin, and A. B. Ozer, "Chaos embedded particle swarm optimization algorithms," *Chaos, Solitons & Fractals*, vol. 40, no. 4, pp. 1715–1734, 2009.
[13] "Alibaba trace," https://github.com/alibaba/clusterdata, 2018.
[14] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proceedings of the International Symposium on Quality of Service*. Association for Computing Machinery, 2019.
[15] L. Liu, H. Huang, H. Tan, W. Cao, P. Yang, and X.-Y. Li, "Online dag scheduling with on-demand function configuration in edge computing," in *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 2019, pp. 213–224.