2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics

A Parallel-batch Multi-objective Job Scheduling Algorithm in Edge Computing

Xingguang Zhao, Xing Guo*, Yiwen Zhang, Wei Li
AnHui University
Hefei, China
1970902100@qq.com, guoxingahu@qq.com, zywahu@qq.com, 1506375560@qq.com

***Abstract*—Edge computing emerges as a novel technique to improve the quality of computation experience for mobile devices which offload computation-intensive jobs to the edge cloud. How to scheduling jobs with limited computation resource on edge servers is a significant scientific problem in edge computing. To handle this problem, in this paper, firstly we propose a Multi-objective job scheduling algorithm with parallel-batch job processing way which optimizes objectives: minimize the overhead and the number of fail jobs which overstep their deadline in edge cloud. Then, considering the unequal performance of the servers and the different slack of waiting for processing of jobs, we define two factors, the processing efficiency of the server and the priority of the job, to optimize the effectiveness of our algorithm. Finally, we design the simulation experiments to compare our algorithm with some existing scheduling algorithms. The results of experiments show our algorithm is efficient and reliable in reducing the overhead and guaranteeing the timeliness of jobs scheduling in edge cloud.**

Keywords—job scheduling; edge computing; heuristic algorithms; parallel-batch job processing

I. INTRODUCTION

With the development of information technology, the demand for resources for applications on mobile devices is also continuous growing [1], [2]. This feature is particularly prominent in real scenario such as smart cities, Cyber-physical-social systems [3]. Due to the resource limitations of mobile device, offloading computation-intensive applications to remote cloud is a way which alleviate the insufficient resource bottleneck on mobile devices, but it also produce a large network delay, affecting the user service experience [4]. Edge cloud can provide sufficient computing resource for mobile device with low latency, which deploy a number of small scale edge servers than remote cloud at the edge of network.

Edge computing is a burgeoning technology which convenient to public live. For instance, edge computing plays a strategic supporting role in the Internet of Vehicles technology: the vehicle-mounted smart device transmits the nearby traffic information such as location images, roadway information, and traffic flow to the edge cloud through wireless network. Vehicle-mounted smart devices can also send different service requests such as request the images of destination, route planning, and traffic conditions information to the nearest edge cloud data center. The edge servers analyzes and processes the collected traffic information for different service requests, and returns the obtained result to the Vehicle-mounted smart devices. The processing grade of the service requests is diverse, for instance, request get the live route planning has more high processing grade than get the image of destination; the service request send from the fire engine will be priority processing relative to ordinary vehicle. In the practical application of IOV technology, the edge servers could not process all collected service requests simultaneously due to the computing and storage resource constrain. Therefore it is indispensable to design a excellent task scheduling strategy to guarantee the service request would be processed timely and economize the resource of edge servers so as to process more service requests. Most of the existing job scheduling methods adopt job process mode that one server only process one job in a moment while the parallel job processing mode is rarely considered[16].

In this paper, we design a parallel job scheduling algorithm to solve the scheduling problem based in edge cloud on the process mode that job scheduling on parallel-batch processing machines (P-BPM), called P-MACO algorithm (Parallel-batch multi-object ant colony algorithm). In parallel-batch processing mode, jobs are grouped into a batch and processed on the edge servers. The release time of the batch is the latest arrival time and the process time of the batch is the longest process time among all jobs in the batch [5]. Uzsoy [6] proved that this job scheduling process is an NP-hard problem, Xu [7] and Jia [8],[9] also proved the ant colony algorithm is an efficient method to figure out this problem. Our contributions can be summarized as follows:

- We first invoke the P-BPM schedule mode to solve the schedule problem in edge cloud. And a P-batch multi-objective job scheduling algorithm is proposed based on the ant colony algorithm, which is called the P-MACO algorithm.
- We propose a method to save running overhead for the phenomenon that the difference in processing efficiency due to different performance parameters between edge servers and processing jobs timely with considering their deadlines.
- We design the simulation experiments to compare our P-MACO with some existing scheduling algorithms to evaluate the performance of our algorithm. The results of experiments show that P-MACO algorithm significantly reduces the execution overhead of edge servers and guarantees processing job timely

The remainder of the paper is organized as follows. In chapter II, we introduce the related work on job scheduling for edge calculations. In chapter III, we describe and define

978-1-5386-7975-3/18/$31.00 ©2018 IEEE
DOI 10.1109/Cybermatics_2018.2018.00109

abstractly the problems we posed. In chapter IV, we specifically describe the proposed algorithm. In chapter V, we verify the performance of P-MACO through simulation experiments. Finally, we conclude the paper in chapter VI.

II. RELATED WORK

With the development of cloud computing technology, resource-intensive applications of resource-limited device could be offloaded to the remote cloud which has rich computing resource. Some task scheduling method for cloud computing has been proposed, Hajeer [20] proposed a method to handling diverse big data with data-aware HDFS and evolutionary clustering technique. However offloading heavy computation jobs to remote cloud suffers form high network latency which leads to decreasing service experience of users. Edge computing can overcome this shortcoming though transfer the data center from remote to the edge of network. Edge computing has received more and more attention in the recent period. Wang presented a novel tensor-based cloud-edge computing framework for Cyber-physical-social systems [3],[10]. This framework has a great advantage in practical scenarios for detecting and processing big data(such as smart cities ,traffic detection, etc) [11], [12]. Due to the limit edge computing resources, a properly strategy for resources allocation and job scheduling is an important issue for the application of Cyber-physical-social systems [3]. How to design an advantage job scheduling strategy to allocating the limit edge computing resources is an urgent problem to be solved.

The problem of job scheduling in edge cloud has been studied extensively in recent years, a number of methods are proposed in different perspectives. Some studies design scheduling strategies by improving the edge cloud hierarchy and framework, Kosta et al [13] proposed a framework named 'ThinkAir' to increase the cloud resource utilization rate and enhance the execution efficiency for jobs which offloaded to cloud; Tong et al [14] proposed a hierarchical edge cloud architecture, they regarded the edge cloud as a tree hierarchy of geo-distributed servers to efficiently utilize the cloud resource for jobs. Some studies design scheduling strategies by optimizing the processing delays of jobs, Zhao et al [15] studied into the scheduling of heterogeneous cloud in order to maximize the probability that tasks can have the delay requirements met, and an optimal algorithm is proposed according to their problem formulation; Tan et al [16] studied the online job dispatching and scheduling problem in edge-cloud, they posed a general model for the problem to minimize the total weight response time of all jobs on edge servers and mobile devices. Some studies design scheduling strategies by reducing the operation energy consumption of servers, hang et al [17] investigated the scheduling policy for collaborative execution in mobile cloud computing to minimize the energy consume by the mobile device and solved the problem with the 'LARAC' algorithm; Zhu et al [18] developed an energy-aware scheduling algorithm named EARH for independent, aperiodic real-time tasks in virtualized clouds to improve the system's schedule ability for real-time tasks and save energy.

These aforementioned studies have obtained some achievement for job scheduling problem in edge cloud, but these methods also have several deficiencies. These scheduling strategies process jobs in serial processing and these algorithms always focus on one optimization objectives in most case. In this paper, we design a parallel job scheduling algorithm with P-BPM in edge cloud to reducing the execution overhead of edge servers and guarantying the timeliness of processing jobs, furthermore we consider the differences among the parameter of edge servers and the deadline of jobs while optimizing our algorithm. We expect our work could inspire further studies on the job of scheduling problem in edge cloud.

III. PROBLEM DESCRIPTION

The studies can be denoted as a P-batch job scheduling problem with the purpose that minimizing the execution overhead of edge servers and the amount of jobs which overstep their deadline. We assume there are M heterogenous servers on edge cloud, denote as $S=\{S_1,S_2,...,S_M\}$, and N indivisible jobs, denote as $J=\{j_1,j_2,...,j_n\}$. Each server S_i has such attributes as processing rate P_i , the energy consumption coefficient for per execution unit v_i [15] and capacity C . Each job J_j has such attributes as release time r_j , deadline d_j , processing priority g_j and size s_j . In edge cloud, a job J_j cannot be processed until the job arrive to a server S_i from mobile device, the delay of communication t_{ji} exists in the transmission procedure of jobs on mobile device, so the arrival time of job J_j to edge servers S_i is $r_{ji}^{'}=r_j+t_{ji}$. The processing time of job is $p_{ji}=s_j/P_i$. The processing priority of job is constantly changing over time. The processing priority of job at a particular moment is related to the distance from the deadline of job to the current moment, a job with less distance has high priority.

For these fail jobs which completed exceeds its deadline, we define a collection $F=\{J_j \mid ct_j > r_j+d_j, J_j \in J\}$ for count these fail jobs. We let B denote the batch set and the kth batch scheduled on S_i denoted by B_{ki}. The release time of batch $Rb_{ki}=\max\{r_{ji}^{'} \mid J_j \in B_{ki}\}$ is the latest arrive time of the jobs in B_{ki} . The processing time of batch $Pb_{ki}=\max\{p_j \mid J_j \in B_{ki}\}$ is the longest processing time of the jobs in B_{ki} . The size of batch $Sb_{ki}=\sum_{J_j \in B_{ki}} s_j$ is the largest size of jobs in B_{ki} , and Sb_{ki} cannot exceed the capacity C .

Each job in job set should be assigned into a unique batch and processed on a unique server. When all jobs have been assigned, we also need to sort the batches on each server in ascending order according to the release time of batch. After sort procedure, the starting time $ST_{ki}=\max\{R_{ki},CT_{(k-1)i}\}$ and completion time

$CT_{ki} = ST_{ki} + PT_{ki}$ of B_{ki} is also determined, then a feasible solution σ will be constructed after all above procedure.

For heterogenous edge servers, their processing rate and energy consumption are not identical. We define processing energy consumption ratio EP_i to evaluate the processing efficiency for edge server S_i :

$$EP_i = \frac{p_i}{v_i} \tag{1}$$

In order to determine a job of a batch can be completed before its deadline whether or not, we define the latest start time $st_{ji}^{\max}$ for job J_j :

$$st_{ji}^{\max} = r_j + d_j - p_{ji} \tag{2}$$

According to the formula(2), the latest start time $st_{ki}^{\max}$ of a batch B_{ki} is :

$$st_{ki}^{\max} = \arg\min_{J_j \in B_{ki}} \{st_{ji}^{\max}\} \tag{3}$$

The processing priority g_j of job J_j , relative to its latest start time, we define it as:

$$grade_{ji} = \frac{1}{st_{ji}^{\max} - r_{ji}^{'}} \tag{4}$$

The first optimization objective of our algorithm is minimize the execution overhead Z of edge servers, include the execution time $C_{\max}$ and total execution energy consumption E . The execution time is the maximum completion time among all batches, i.e. $C_{\max} = \max_{B_{ki} \in \sigma} \{CT_{ki}\}$. To calculate the total energy consumption E , we first set the energy consumption of edge server S_k process per job J_j is $e_j = s_j * v_k$, then the the energy consumption of edge server S_k process per batch B_{ki} is $Eb_{ki} = \sum_{J_j \in B_{ki}} e_j$, thus the the total energy consumption of all edge server S_k is $E = \sum_{i=1}^{m} Eb_{ki}$. The second optimization objective is minimizing the number of jobs whose completion time oversteps its deadline in collection F . Two optimization objectives formulated as:

$$Minimize \;\; Z = C_{\max} + E \tag{5}$$

$$Minimize \;\; card(F) \tag{6}$$

Equation (5) presents the first optimization objective is minimizing the execution overhead of all edge servers for processing job set according to the adapting job scheduling strategy; Equation (6) expresses the second optimization objective is minimizing the count of fail jobs that complete untimely.

IV. PARALLEL MULTI-OBJECTS ANT COLONY OPTIMIZATION SCHEDUING ALGORITHM

With the problem described above, we next propose our job scheduling algorithm. We first detail the steps for constructing solution in section A and then describe the complete steps of our P-MACO algorithm in section B.

A. Solution Construction

The P-MACO algorithm is based on ant colony algorithm and solves our study scheduling problem with multiple iterations. The solution for our problem is a policy which is used to assign jobs into batch and schedule constructed batch on edge servers. We reference the batch construction policy in [8], build the batch and schedule the batch on edge servers simultaneously, when a server has been chosen to build a empty batch, a unscheduled job would be assigned into the empty batch randomly. After all unscheduled jobs have been assigned into a batch on an exact edge server, a solution is constructed completely. A pareto-optimal solution sets will be established to compare these generated solutions and get pareto-optimal solutions.

1)Batch construction: In our algorithm, the procedure of assigning a job into a new batch B_{bi} as follows: First, select a job from the job set randomly and add it to the new batch B_{bi} . Second, create two candidate set $CL1$ and $CL2$ based on the current batch B_{bi} and select a job to add in B_{bi} .Then repeat second step until the size of B_{bi} exceeds the server's capacity or the all candidates set are empty. A new batch will be construction after an edge server has been selected, so we must determine an edge server according to the decision rule firstly. We prefer the server which faster completes their processing procedure and has higher processing energy consumption ratio, as follows:

$$\arg\min_{M_i \in M} \{CT_{M_i} \bullet \frac{1}{EP_{M_i}}\} \tag{7}$$

In theoretically all jobs require to traversing in order to detect whether it is appropriate to be assigned into a batch which is a time-consuming process. We improve search efficiency by creating candidate sets: before selecting a new job to add into a batch, two different job candidate sets $CL1$ and $CL2$ will be created first.

We define the job candidate sets $CL1$ as:

$$CL1 = \{J_j \mid J_j \in J\} \tag{8}$$

$$\text{s.t. } r_{ji}^{'} \le ST_{ki} \tag{9}$$

$$s_j \le C - Sb_{ki} \tag{10}$$

$$r_{ji}^{'} \le st_{ki}^{\max} \tag{11}$$

$$ST_{ki} \le st_{ji}^{\max} \tag{12}$$

The definition of job candidate sets $CL2$ as:

$$CL2=\{J_j \mid J_j \in J\} \tag{13}$$

$$\text{s.t. } r_{ji}^{'} > ST_{ki} \tag{14}$$

$$s_j \le C - Sb_{ki} \tag{15}$$

$$r_{ji}^{'} \le st_{ki}^{\max} \tag{16}$$

$$ST_{ki} \le st_{ji}^{\max} \tag{17}$$

The arrival time of each job to the server S_i which the batch B_{ki} on in $CL1$ does not overstep the start time of batch B_{ki}, while the arrival time of all jobs in $CL2$ is no more than the start time of batch B_{ki}. Additionally, two constraints should be satisfied: 1)the size of jobs in $CL1$ and $CL2$ are smaller than the residual space of server S_i; 2)The arrival time of any job does not exceed the latest start time of the batch B_{ki}. The constraint 1) ensure the size of each batch will not overstep the capacity of server and constraint; 2) ensure all in current batch will not overstep its deadline under the influence of waiting for new job to be added in batch.

The process of assigning a job into a new batch B_{bi} is as follows: First, select a job from the job set randomly and add it to the new batch B_{bi}, then create a candidate set $CL1$ and $CL2$ set based on the current batch B_{bi} and select a job to add in B_{bi} until the size of B_{bi} exceeds the server's capacity or the candidates set are empty.

Pheromone trails and heuristic information are used to guide the job selection from the candidate set to add in batch. We build a pheromone matrix for jobs in job set and define the pheromone trails τ_{hj} as the probability that the job J_h and job J_j are grouped together. Pheromone trails τ_{bj} is applied to the probability that job J_j in candidate set is assigned to batch B_{bi}. The value of τ_{bj} is set as the total pheromone trail of the candidate job and the jobs in batch B_{bi}:

$$\tau_{bj} = \frac{\sum_{J_h \in B_b} \tau_{hj}}{|B_b|} \tag{18}$$

The setting of heuristic information is determined by the studied problem. The optimization objectives of our study problem are reducing execution overhead of edge servers and optimizing the timeliness of processing jobs which in relation to $C_{\max}$, E and g_j. While in chapter Ⅲ, we had presented the calculation of energy consumption E is determined by edge servers, we only define the heuristic information for $C_{\max}$ and g_j. The impact on the job execution time mainly relative to two aspects:1)the start time of job;2)the processing time of job and the timeliness of processing job can be regulated by the processing priority. Due to the arrival time of jobs in $CL1$ and $CL2$ are different, we need define heuristic information η_{kj}^1 and η_{kj}^2 for $CL1$ and $CL2$ respectively. The arrival time of jobs in $CL1$ are less than the start time of current batch B_{ki}, so these jobs in does not postpone the start time of batch B_{ki}, η_{kj}^1 is defined as:

$$\eta_{kj}^1 = \frac{1}{|PT_{ki} - p_{ji}| + 1} \bullet grade_j \tag{19}$$

PT_{ki} is the process time of current batch B_{ki} on edge server S_i, p_{ji} is the process time of job J_j on edge server S_i and $grade_j$ present the scheduling priority. The job would be preferred adding into current batch B_{ki} under construction which prolongs fewer process time of current batch B_{ki} and possess higher scheduling priority.

Jobs in are $CL2$ later than the start time of current batch, this could lead to put the start time back of current batch, η_{kj}^2 is defined as:

$$\eta_{kj}^2 = \frac{1}{|PT_{ki} - p_{ji}| + 1} \bullet \frac{1}{r_{ji}^{'} - ST_{ki}} \bullet grade_j \tag{20}$$

The arrive time $r_{ji}^{'}$ jobs in $CL2$ is later than the starting time of current batch B_{ki}, therefore the job would be preferred adding into current batch B_{ki} which extends fewer starting time of current batch B_{ki}.

According to the above description, the probability of choose job J_j and add it into batch B_{bi} is formulated as follows:

$$P_{bj} = \begin{cases} \dfrac{\tau_{bj}^{\alpha}(\eta_{bj}^1)^{\beta}}{\sum_{J_j \in CL1} \tau_{bj}^{\alpha}(\eta_{bj}^1)^{\beta}}, J_j \in CL1 \\ \dfrac{\tau_{bj}^{\alpha}(\eta_{bj}^2)^{\beta}}{\sum_{J_j \in CL2} \tau_{bj}^{\alpha}(\eta_{bj}^2)^{\beta}}, J_j \in CL2 \\ 0 \qquad\qquad\qquad , otherwise \end{cases} \tag{21}$$

The τ_{bj} and η_{bj} are the pheromone trail and heuristic information, guide the job selection form the candidate set

to add in current construction batch. Parameter $\alpha+\beta=1$,determining the relative importance of pheromone trail and heuristic information.

2) The update of pheromone trail: The update of pheromone is divided into two parts: the update of local pheromone and the update of global pheromone. The local pheromone update is implemented once an ant adds a job to a batch, the value of corresponding pheromone should be updated so as to reduce the probability of selecting the job which has been accessed and strengthen the search in unknown solution space. The update of local pheromone as follows:

$$\tau_{hj}=(1-\rho_l)\cdot\tau_{hj}+\rho_l\cdot\tau_{hj}^0 \tag{22}$$

τ_{hj}^0 is the initial value of pheromone trail corresponding job J_h and J_j in pheromone matrix. ρ_l is pheromone evaporating factor which control the speed of pheromone evaporation for the local update of pheromone.

The global pheromone update is implemented according to the update of the non-dominated solutions (NDS). After an ant constructed a new solution, the new solution should insert into NDS and compare with other pre-existing pareto-solutions of NDS for optimization objectives, then update the NDS. The rule is defined as follows:

$$\tau_{hj}=(1-\rho_g)\cdot\tau_{hj}+\rho_g\cdot\sum_{sol\in NDS}\Delta\tau_{hj} \tag{23}$$

$$\Delta\tau_{hj}=\begin{cases}\dfrac{1}{Z}, \textit{if job } J_h \text{ and } J_j \text{ are in the same batch;}\\ 0 \qquad\qquad \textit{, otherwise}\end{cases} \tag{24}$$

Z is the execution overhead of edge servers process job set according to a solution, ρ_g is pheromone evaporating factor for the global update of pheromone.

Algorithm 1: The LOS Algorithm

```
1: for i = 1 to m
2:    sort the batches execution sequence on server S_i in ascend order
      of release times of the batches
3: for j = 1 to m
4:    for j = 1 to B_km
5:       if j = 1
6:          ST_ji = RT_ji
7:       else
8:          ST_ji = max(RT_ji, CT_(j-1)i)
9:          CT_ji = ST_ji + PT_ji
10:   end for
11: end for
```

3) Local optimization of start time: When a new batch has been constructed, a job will be selected to add to the new batch randomly, the initial information of the batch is determined by the chosen job. Thus the batches may be eventually disorder after all jobs have been assigned. Therefore it is integrant to adjust the order of these batches according to their start time in order that construct a solution. We design Local optimization of start time (LOS) algorithm to implement the sorting work. Algorithm LOS is described in algorithm 1.

B. The P-MACO Algorithm

Algorithm 2: The P-MACO Algorithm

```
1: Initialize pheromone matrix, pheromone evaporating factor,
   parameters α, and β , and job list J = {1,2,3,...,n}
2: t = 1
3: if t < T_max
4:    a = 1
5:    if a < N_a
6:       if U ≠ ∅
7:          ant a choose server S_i and construct a new batch B_bi on
            S_i ;select a job from J randomly and insert in the new batch;
            update J
8:          construct candidate lists CL1 and CL2
9:          if CL1 ≠ ∅
10:            select a job from CL1 and insert to B_bi ;update local
               pheromone trail and J
11:         else if CL2 ≠ ∅
12:            select a job from CL2 and insert to B_bi ;update local
               pheromone trail and J
13:         else
14:            insert batch B_bi to the batches execution sequence and use
               LOS Algorithm to update the batches execution sequence of
               server S_i
15:         insert generated solution to NDS, update NDS and global
            pheromone trail
16:      a++
17:   t++
18: Output NDS
```

During constructing the solution, once a job has been assigned into a batch, the job should be deleted from the job set. Thus the job set is $\varnothing$ means all jobs have assigned complete. The pseudo-code of the P-MACO algorithm is shown in Algorithm 2. To schedule job set $J=\{1,2,3,...,n\}$, during each iteration, each ant first selects the appropriate server S_i according to equation (7), constructs a new batch B_{bi} and selects one job form U add to B_{bi} randomly (line 7). Then establishing two candidates set $CL1$ and $CL2$ according to equation (8)(9)(10)(11)(12) and (13)(14)(15)(16)(17), select a job from the candidate set according to equation (21), according to equation (22) to update the local pheromones trail simultaneously. (line8-14). All jobs are assigned. After that, the generated solution is compared with other solutions in NDS, the NDS and global pheromone are updated according to equation (23) (line 15). After all iterations are completed, the generated optimal solution is output (line 18).

V. EXPERIMENTS

A. Experiment Settings

We implement our algorithm with Python 3.5. Since no P-BPM job scheduling algorithm has been proposed in edge computing, we compare our algorithm with two scheduling algorithms, namely, FCFS (First-Come-First-Serve) and PACO(Pareto-based ant colony optimization) [9]. FCFS algorithm is a classic scheduling algorithm, all jobs are processed with the sequence of the arrival time of jobs on each server. PACO is a algorithm with P-BPM scheduling mode, while the arrival time of jobs should be adjusted as the sum of release times and network delay in our environment, and the method of calculating energy consumption is adjusted to our method. The selection of edge server for FCFS and PACO is the server which completes its current process first. We imitate the method in [9] to generate five different job sets containing the number of jobs (100, 200, 300, 400, 500) respectively in 60 seconds. Four groups' edge servers (4×2) are used to handle jobs. Two servers in each group have the same processing parameters. The energy consumption coefficient v_i and processing rate P_i for each server group are set to (0.5/MB, 2MB/S), (1.0/MB,4MB/S), (1.5/MB,6MB/S), (2.4/MB,8MB/S) respectively. The details of other parameter settings are shown as Table 1.

TABLE I. FACTORS AND LEVELS

Factors	Value
r_j	U[1，60s]
s_j	U[1，10MB]
C	50MB
d_j	U[1，10s]
(ρ_l,ρ_g)	(0.5,0.5)
(α,β)	(3，4)
t_v	10MB/S
ant number	2*(job number)
$T_{\max}$	200

The performance measure of these algorithms is compared on two metrics: the total execution overhead and the number of fail jobs which overstep its deadline. Due to the number of Pareto optimal solutions obtained by P-MACO algorithm may more than one, we choose the solution with the fewest number of fail job as the ultimate solution to compare the algorithms with the reason that ensuring the timeliness of job process is more important than saving server running costs in job scheduling problem of edge cloud [19].

B. Effectiveness evaluation

1) Execution overhead: We first study the overhead of processing job set on edge servers. The execution overhead is depicted in Fig.1, the P-MACO algorithm outperforms FCFS and PACO algorithm on all five job sets. The FCFS algorithm performs worst and the performance of PACO algorithm between P-MACO and FCFS. We explain this phenomenon through Fig.2 and Fig.3.The performance of algorithms for two subdivisions of the overhead, execution time $C_{\max}$ and total energy consumption E, are depicted in Fig.2 and Fig.3 respectively. In Fig.2, the P-MACO algorithm cost less execution time than other algorithms. The FCFS algorithm only processes jobs in order of their arrival time in sequence on each server, this mode will lead to too much waiting process time for jobs which later arrive to edge servers. As a result, the total execution time for whole job set will be augment along with the increase in amount of jobs. For PACO algorithm, the chosen of edge server for processing jobs according to the complete time of each server, reckon without the processing rate for different edge servers, so the execution time of job processing is longer than P-MACO algorithm. The compare of energy consumption is shown in Fig. 3, the P-MACO algorithm consumes less energy than other algorithm. The reason is that FCFS and PACO choose their server for processing jobs prefer the server which complete its process jobs while P-MACO chooses edge server with the processing energy consumption ratio. It result to P-MACO chooses more faster and less energy consumption servers and reduce the execution overhead for all servers.

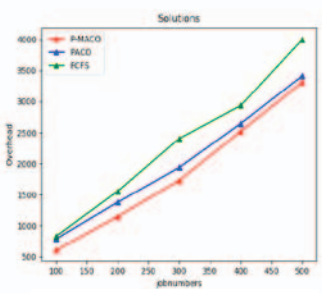

Fig.1 Overhead of edge servers

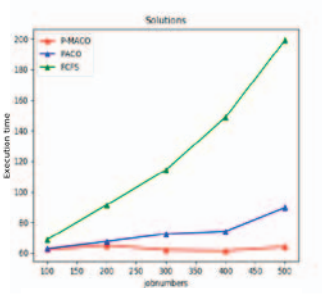

Fig.2. The execution time

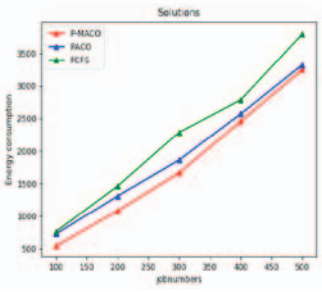

Fig.3. Energy consumption

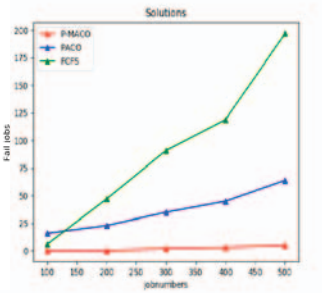

Fig.4. Number of fail jobs

2) The number of fail jobs: Then we study the number of fail jobs in job scheduling, the performance for three algorithms in all five job sets are depicted in Fig.4.It's clear that P-MACO performs best in all five job sets while the performance of FCFS and PACO algorithm not so well. The reason is that in P-MACO, select job to add in batch consider that 1) prefer job with high processing priority and 2) the arrival time of new job will not exceed the latest start

time of batch, while FCFS algorithm and PACO algorithm only with the view to the residual space of server result in some jobs may beyond their deadline due to waiting for new add-in jobs. As the number of job set increases, the fail job will increase in job scheduling with the FCFS and PACO algorithm.

VI. CONCLUSION

In this paper, first we study the job scheduling problem in the edge cloud and propose a parallel batch job scheduling algorithm for minimize the execution overhead of edge servers and the number of fail jobs of job sets, called P-MACO algorithm. Then we consider two issues, the parameter of different edge servers and the process priority of different jobs which affect the scheduling process, defining the processing efficiency of the server and the priority of the job to guide the construction of batches and the selection of processing servers. At last, we design a set of experiments to evaluate the performance of our algorithm in execution overhead and the amount of fail jobs with other algorithms. The result shows that the proposed job scheduling algorithm is superior to the compared algorithms. In the future, we will focus on other factors which influence the job scheduling in edge cloud such as balancing peak load among edge servers and the instability of network transmission. Furthermore, how to shorten the time complexity is also an unresolved issue to optimize job scheduling algorithm so that implement it on a real edge computing scenario.

REFERENCE

[1] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-Saving offloading by jointly allocating radio and computational resources for mobile edge computing," IEEE Access, vol. 5, pp.11255-11268, June 2017.

[2] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 2546–2559, September 2016.

[3] X. Wang, Laurence T. Yang, X. Xie, J. Jin, and M. Jamal Deen, "A cloud-edge computing framework for cyber-physical-social services," IEEE Communications Magazine, vol.55, no.11, pp. 80-85, 2017.

[4] J. Pan, and J. Mcelhannon, "Future edge cloud and edge computing for internet of things applications," IEEE Internet of Things Journal,vol.99, pp.1-1, 2017.

[5] P. Damodaran, M. Vélez-Gallego, and J. Maya, "A grasp approach for makespan minimization on parallel batch processing machines," Journal of Intelligent Manufacturing, vol.22, pp.767-777, 2011.

[6] R. Uzsoy, "Scheduling a single batch processing machine with non-identical job sizes," International Journal of Production Research, vol.32, pp.1615–1635, 1994.

[7] R. Xu, H. Chen, and X. Li, "Makespan minimization on single batch-processing machine via ant colony optimization," Computers & Operations Research, vol. 39, pp.582–593, 2012.

[8] Z. Jia, X. Li, and J. Y.-T. Leung, "Minimizing makespan for arbitrary size jobs with release times on P-batch machines with arbitrary capacities," Future Generation Computer Systems, vol.67, pp.22-34, 2017.

[9] Z. Jia, Y Li, J.Y.-T. Leung, and K Li, "Bi-criteria ant colony optimization algorithm for minimizing makespan and energy consumption on parallel batch machines," Applied Soft Computing, vol.55, pp.226-237, 2017.

[10] X. Wang, L. T. Yang, J.. Feng, X. Chen, and M. Jamal Deen, "A tensor-based big service framework for enhanced living environments," IEEE Cloud Computing Magazine, vol. 3, No. 6, pp.36-43, 2016.

[11] X. Wang, L. T. Yang, H. Liu, and M. Jamal Deen, "A big data-as-a-service framework: state-of-the-art and perspectives," IEEE Transactions on Big Data, vol. 15,pp.80-85, 2017.

[12] X. Wang, L. T. Yang, X. Chen, and J. Han, "A tensor computation and optimization modelfor cyber-physical-social big data," IEEE Transactions on Sustainable Computing, 2017.

[13] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading," IEEEInfocom2012, pp.945-953, 2013.

[14] L. Tong, Y. Li and W. Gao, "A hierarchical edge cloud architecture for mobile computing," IEEE Infocom, pp.1-9, 2016.

[15] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," IEEEICC2017,2017.

[16] H. Tan, Z. Han, X. Li, and F. Lau, "Online job dispatching and scheduling in edge-clouds," IEEE Infocom 2017, pp.1-9, 2017.

[17] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," IEEE Infocom2013, vol.12, pp.190-194, 2013.

[18] X. Zhu, L. T. Yang, H. Chen, J. Wang, and S. Yin, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," IEEE Transactions on Cloud Computing, vol.2, pp.168-180, 2014.

[19] T. Li, M. Wu, M. Zhao, and W. Liao, "An overhead-optimizing task scheduling strategy for ad-hoc based mobile edge computing," IEEE Access, vol. 5, pp.5609-5622, 2017.

[20] M. Hajeer, D. Dasgupta, "Handling big data using a data-aware hdfs and evolutionary clustering technique," IEEE TRANSACTIONS ON BIG DATA, 2016.