Ant Colony Algorithm for Container-based Microservice Scheduling in Hybrid Cloud

Miao Lin¹, Jianqing Xi¹, Weihua Bai ² and Jiayin Wu³

¹School of Software Engineering, South China University of Technology, Guangzhou 510006, China
²School of Computer Science, Zhaoqing University, Zhaoqing 526061, China
³School of Computer, Guangdong Vocational College of Post and Telecom, Guangzhou 510630, China
linmiao.nick@qq.com

Abstract. The microservice model divides the application into a group of loosely coupled and collaborative fine-grained services, which results in the workflow scheduling problem, especially in the complex environment of resource diversity such as hybrid cloud. To solve this problem, this paper establishes a workflow scheduling model of microservice based on multiple container instances and proposes an ant colony optimization (ACO) algorithm to optimize finish time and resource cost. The algorithm proposed optimizes update strategies of multi-objective heuristic information to improve the selection probability of the dominant path. The comparative experiments show that the proposed optimization algorithm achieves better results in scheduling objectives in the hybrid cloud.

1. Introduction

The microservice-based application is designed as a group of independent, fine-grained modular services by the business difference. The function of the application is co-realized through a group of microservices, and microservices communicate by the lightweight communication mechanism[1]. As a lightweight virtualization technology of the operating system layer, Docker container[2] can provide an independent execution environment and file system for the application running. Container is an excellent tool for encapsulating, isolating, and deploying microservices.

At present, with the expansion of application function and the doubling of the amount of industry data, especially in peak hours, the existing clusters of enterprises can not satisfy service requirements, so it needs to increase the investment in hardware facilities. But in addition to the peak hours, most of the equipment maintains a not high utilization rate, resulting in a lot of resource waste and increased operating costs. Therefore, to reduce hardware investment and maintenance costs while improving application performance, more and more enterprises turn to the hybrid cloud as the enterprise application deployment platform. In cloud computing, application services and cloud resources are diverse, especially in the hybrid cloud. In the face of private cloud with relatively low cost but low...
performance and public cloud with relatively high cost but high performance, how to deploy services in the hybrid cloud to improve performance and reduce cost has become a crucial issue.

Container deployment is a typical NP-hard problem. Currently, many researchers have applied the ant colony algorithm to solve the scheduling problem of the virtual machine in cloud computing. ACO algorithm is a global optimization algorithm with probability and uncertainty, which is easy to get the global optimal solution. This paper establishes a workflow scheduling model of microservice based on multiple container instances considering the optimization of cloud resource cost and the finish time of the microservice application in the hybrid cloud. Ant colony algorithm is applied to solve the multi-objective optimization problem of container scheduling.

2. Related work
This paper introduces the related work from three aspects: workflow scheduling in hybrid cloud/multi-cloud, workflow scheduling based on container/microservice, and the application of the ACO algorithm in workflow scheduling.

Firstly, the related work about workflow scheduling in hybrid cloud/multi-cloud is showed here. For example, Chang et al. [3] proposed an agent-based workflow scheduling mechanism in the hybrid cloud. The mechanism adopts a rough set theory to estimate the execution time of the task and a heuristic algorithm to realize dynamic job dispatching, aiming to reduce the usage of the VM and improve resource utilization. Pasdar et al. [4] proposed a two-phase scheduling algorithm with a genetic algorithm-based static phase and dynamic programming-based dynamic phase (Hybrid Scheduling for Hybrid Clouds, HSHC), aiming to address the scheduling of scientific workflows in hybrid clouds considering data placement from two aspects of makespan and costs. Mohammadi et al. [5] addressed the problem of scientific workflow scheduling in multi-cloud settings under deadline constraints to minimize associated financial costs. The work proposed integer linear programming models that can be solved within a reasonable time by available solvers and analyzed the treatment of optimal cost under variations in deadline and workflow size. Gao et al. [6] proposed a Workflow Mapping algorithm for Financial Cost Optimization (WMFCO) to minimize the financial cost of deadline-constrained scientific workflows executed in multi-cloud environments, considering storage requirements, I/O operations, and data transfers.

Secondly, there is some related work about workflow scheduling based on container/microservice. To reduce the workflow execution time and the network bandwidth consumption, Zhang et al. [7] proposed a cost-efficient and latency-aware workflow scheduling algorithm that strategically loads the containers into VMs and executes the tasks on the VMs. The algorithm bases on "Stretch Out and Compact" which can stretch out the tasks along the resources by critical path analysis and then find the inefficient slots within the computing resources and eventually compact the tasks into those slots. Wang et al. [8] proposed an Elastic Scheduling for Microservices (ESMS) that integrates task scheduling with auto-scaling, aiming to minimize the cost of virtual machines while meeting deadline constraints. In the ESMS algorithm, the authors used a statistics-based strategy to determine the configuration of containers under a streaming workload and proposed an urgency-based workflow scheduling algorithm that assigns tasks and determines the type and quantity of instances for scale-up. Rajasekar et al. [9] proposed a strategy of scheduling and resource provisioning planned particularly for WaaS platforms. The algorithm is dynamic and scalable to adjust to improve in the workload and platform. It supports containers to deal with the inefficiency of resource utilization and targets to reduce the overall execution cost of infrastructure resources as fulfilling each single workflow deadline constraint.

Thirdly, the ACO algorithm is essentially a heuristic global optimization algorithm in the evolutionary algorithm, which has been used to schedule workflow in the field of cloud computing. For example, to reduce the expenditure of workflow under the deadline and constraints, Jia et al. [10] designed an estimation model of the task execution time according to virtual machine settings in real public clouds and execution data from practical workflows and proposed an adaptive ACO algorithm to meet the quality of service and orchestrate tasks. Yuan et al. [11] proposed a multi-scientific...
workflows security-deadline constraint cost optimization algorithm (MSW-SDCOA). Firstly, based on data flow dependency, the MSW-SDCOA algorithm compresses scientific workflow and reduces the number of task nodes to save scheduling cost. Secondly, through the optimizing HEFT algorithm, a scheduling sequence is formed to realize overall multi-objective optimization scheduling. Lastly, by optimizing the update strategy of pheromone and heuristic information in ACO, the cost optimization effect is further improved. Kianpisheh et al.[12] described the user requirements by defining constraints on the workflow makespan and/or its execution cost, and proposed probability of violation (POV) of constraints as a criterion for the schedule robustness. An ant colony system is then used to minimize aggregation of violation of constraints and the POV. Nasr et al.[13] presented a hybrid algorithm (CR-AC) for workflow scheduling in the cloud computing environment to balance the workload on the available resources and achieve minimum schedule length without violating the deadline constraint. The CR-AC algorithm combines both the chemical reaction optimization (CRO) to find a near-optimal solution rapidly and the ACO algorithm to improve the solution quality.

Although many algorithms have been designed for workflow scheduling in hybrid cloud/multi-cloud, most of these works are based on the virtual machine. Currently, the research of workflow scheduling based on container/microservice is more about flexibility and scalability of the scheduling, ignoring the impact of multiple container instances of microservice on workflow scheduling. There remain open issues that have not been completely addressed in container-based microservice scheduling. ACO algorithm has been widely used in various scheduling problems and achieved good results. It has certain advantages to solve such combinatorial optimization problems[14]. Different from the previous work, this paper considers the workflow scheduling problem of microservice based on multiple container instances in hybrid cloud and proposes an ant ACO algorithm to optimize finish time and resource cost.

3. Problem description

3.1. System model

Microservice application $APP$ is formalized as a tuple $<MS\_SET, DATA, LINK>$, where $MS\_SET=\{ms_i|i=1,2,\ldots,m\}$ is the microservice set of the application $APP$; $DATA=\{data_{ix}\}$ is the set of the amount of data need to be transmitted in a request between microservices; $LINK=\{link_{ij}\}$ is the set of the number of requests between microservices within a unit of user service requests during a specific time slot.

Microservice $ms_i$ is formalized as a tuple $<Calc\_Reqsti, Str\_Reqsti, Link\_Thri>$, where $Calc\_Reqsti$ is the computing resources required to fulfill a request for $ms_i$; $Str\_Reqsti$ is the storage resources required to fulfill a request; $Link\_Thri$ is the threshold of the number of requests for the $ms_i$. The values of $Calc\_Reqsti$, $Str\_Reqsti$, and $Link\_Thri$ depend on the implementation of $ms_i$. $Link_i=U\sum_{i=1}^{m} link_{i,j}$ is the total number of requests for the $ms_i$ from the other microservices, and $U$ represents the number of units of user requests in a specific time slot. Each microservice is encapsulated in a container, and a microservice can have multiple container instances. $Scale_i$ is the number of container instances for $ms_i$ and is calculated as $Scale_i=\frac{Link_i}{Link\_Thri}$. The values of $Link_i$ and $Scale_i$ depend on the number of units of user requests in a specific time slot. $cont_{ij}$ represents the $j$th container instance of microservice $ms_i$, $pred(ms_i)$ represents the set of the predecessors of microservice $ms_i$; $succ(ms_i)$ indicates the set of the successors of microservice $ms_i$.

Hybrid cloud $CLOUD$ is formalized as a tuple $<VM\_SET, B, D>$, where $VM\_SET=\{vm_i\}$ represents the set of VM types, and $vm_{ix,j}$ indicates the $j$th instance of $vm_i$ (VM with type $x$); $B=\{bx,y\}$ represents the set of network bandwidth between VMs with type $x$ and type $y$; $D=\{dx,y\}$ represents communication latency between VMs with type $x$ and type $y$. VM $vm_i$ is characterized as a tuple $<Calc_{ix}, Str_{ix}, ss_{ix}, price_{ix}>$, where $Calc_{ix}$, $Str_{ix}$ respectively represent the computational capacity and storage capacity of $vm_i$;
sx indicates the computing power of vmx; price indicates the price per unit time of vmx. Particularly, the price per unit time of VM in private cloud is 0. In addition, VM_SETp represents the set of VMs in the public cloud. vmx,y = alloc(conti,j) indicates that container instance conti,j of microservice msi is deployed to VM instance vmx,y.

3.2. Computation Model

Microservice container conti,j is deployed to VM vmk,l. The service execution time of conti,j is formulated as:

$$ET(conti,j,vmk,l) = \frac{Calc_\text{Req}_t}{s_k}.$$  

msi is the predecessor of msj. Microservice container contp,j is deployed to VM vmk',l'. The data transmission time from conti,j to contp,j is formulated as:

$$TT(conti,j, vmk,l, contp,j', vmk',l') = \begin{cases} b_{k,k'} + d_{k',l'}, & \text{otherwise} \\ 0, & \text{if } k = k' \text{ and } l = l' \end{cases}.$$  

msq is the predecessor of msi. The latest start time of container conti,j is calculated according to the latest finish time of all the containers consuming the service provided by container conti,j. Therefore, the latest start time of container conti,j is formulated as:

$$LST(conti,j, vmk,l) = \max \left\{ \max_{\forall msp \in prec(msi)} \left[ \max_{i=1,...,Scale_i} \left[ LST(contqj', vmk',l') + ET(contqj', vmk',l') \right] \right] + TT(contqj', vmk',l', conti,j, vmk,l) \right\}.$$  

The application may exist many microservices without predecessors, so we need to set up a virtual entry microservice msentry for the application. msentry is the predecessor of all the microservices without predecessors. Similarly, set up a virtual exit microservice msexit. The latest start time, the service execution time, and the data transmission time of msexit are respectively set as:

$$LST(msentry, *) = ET(msentry, *) = TT(msentry, *) = 0.$$  

The maximum finish time of processing service by the application is: $MAKESPAN = LST(msexit, *)$.

The total cost per unit time of renting virtual machines for application in the public cloud is:

$$COST = \sum_{vmx,y \in VM_SETp} price_x .$$

3.3. Scheduling Objectives

The scheduling objective model is as follows:

Minimize MAKESPAN

Minimize COST

s.t. $\sum_{i=1}^{m} \sum_{j=1}^{Scale_i} \frac{Link_{Calc_\text{Req}_t} \leq Calc_x, \forall vm_{x,y}}{Scale_i} \leq Calc_x, \forall vm_{x,y}$  

s.t. $\sum_{i=1}^{m} \sum_{j=1}^{Scale_i} \frac{Link_{Str_\text{Req}_t} \leq Str_x, \forall vm_{x,y}}{Scale_i} \leq Str_x, \forall vm_{x,y}$

Equations (1),(2) represent respectively the optimization of the two objectives. Equations (3),(4) are the constraint conditions of the VM resource capacity. Particularly, in this paper, the service requests for each microservice are evenly distributed among multiple container instances.
4. Scheduling algorithm

4.1. Scheduling sequence generation

We need to determine the priority of the container and generate a scheduling sequence with the priorities of the containers in descending order. The formula of the priority is as follows:

\[
\text{rank}(ms_i) = \max_{ms_j \in \text{output}(ms_i)} [\text{rank}(ms_j)] + 1.
\]  

(5)

Here \( \text{rank}(ms_{exit}) = 0 \).

The scheduling sequence generation process of the containers is shown in Algorithm 1 as follows:

**Algorithm 1: Scheduling sequence generation algorithm**

Input: APP;
Output: Scheduling sequence \( Q \);

Begin
\[
Q = \{\};
\]
for each \( ms_i \) in APP
if \( ms_i = ms_{exit} \)
Add tuple \( (ms_{exit}, 0) \) to \( Q \);
else
Calculate the \( \text{rank}(ms_i) \) according to Equation (5);
for \( j = 1 \) to \( \text{Scale}_i \)
Add tuple \( (\text{cont}_{i,j}, \text{rank}(ms_i)) \) to \( Q \);
end for
end if
end for
Sort \( \text{cont}_{i,j} \) in \( Q \) in descending order of \( \text{rank}(ms_i) \); return \( Q \);
End

4.2. Heuristic information

\( \eta_{i,j,x,y}^k \) is the heuristic information which represents the expectation of ant \( \text{Ant}_k \) to deploy container \( \text{cont}_{i,j} \) to VM \( \text{vm}_{x,y} \). Heuristic information \( \eta_{i,j,x,y}^k \) is defined as follows:

\[
\eta_{i,j,x,y}^k = \left( \frac{q \times s_i}{\sum_{j=1}^q (s_j)} \right) \times \left( \frac{1}{\text{LST}(\text{cont}_{i,j}, \text{vm}_{x,y}) + \text{ET}(\text{cont}_{i,j}, \text{vm}_{x,y})} \right) \times \left( 1 - \frac{\text{num}_{x,y}}{\text{num}} \right). \]  

(6)

Here \( p(\text{vm}_{x,y}) = \{ \text{price}_x, \text{vm}_{x,y} \in \text{VM\_SET}_u \} \); \( \epsilon \) is a small enough positive number; \( \text{num} = \sum_{i=1}^m \text{Link}_i \) is total number of service requests for all the containers; \( \text{num}_{x,y} = \sum_{i=1}^m \sum_{j=1}^{\text{Scale}_i} [\text{Link}_i]_{\text{cont}_{i,j}} \text{allocate}(\text{cont}_{i,j}) \) is the number of service requests for all the containers already deployed to \( \text{vm}_{x,y} \), dynamically calculated in processing of the solution constructing; \( q \) is the number of VMs currently available. \( \omega, \xi, \psi \) are the preference regulators.

In Equation (6), the first item represents the cost-performance of VM \( \text{vm}_{x,y} \) among all available VMs. The second item is about time. Since the goals of this paper include minimizing the finish time, relatively low start time and execution time mean that \( \text{vm}_i \) is more suitable for microservice \( ms_i \) than other types of VMs. The third item is about the number of service requests for the containers that have
been deployed to the VM $vm_{x,y}$, aiming to avoid that the containers are always deployed to few VMs and increase the probability of other VMs.

$\omega$ is used to adjust the weight of cost performance of VMs in heuristic information. The larger its value is, the free private cloud VMs or public cloud VMs with better cost-performance are more appeals to microservices. If the finish time of multiple solutions is within the time threshold and the cost is more than a certain cost threshold in the last iteration, the value of $\omega$ could be adjusted appropriately, that is, the value of $\omega$ could be set to the number of solutions whose cost is more than the cost threshold.

$\xi$ is used to adjust the weight of the finish time of microservice in heuristic information. The larger its value is, the more microservices prefer the VMs that can finish services earlier. If the finish time of multiple solutions is more than a certain time threshold in the last iteration, a smaller value should be set for $\xi$ (be set as the ant colony size minus the number of solutions whose finish time is more than the time threshold).

$\psi$ is used to determine the weight of service request congestion degree in heuristic information. The larger its value is, the more microservices prefer the VMs with fewer service requests. In the early stage of the ACO algorithm running, to obtain more feasible solutions, the value of $\psi$ can be set to close to the size of the ant colony; but in the later stage of the ACO algorithm running, the objectives need to be optimized rather than just find more feasible solutions, so the value of $\psi$ is always close to or equal to 0.

### 4.3. Pheromone updating

$\tau_{i,j,x,y}(t)$ is the pheromone on the path $path(i,j,x,y)$ at time $t$. The formula of pheromone updating is as:

$$
\tau_{i,j,x,y}(t+1) = (1 - \rho)\tau_{i,j,x,y}(t) + \Delta\tau_{i,j,x,y}
$$

Here $\tau_{i,j,x,y}(t+1)$ is the updated pheromone. The initial value of the pheromone can be set to $\tau_{i,j,x,y}(1)=[ET(cont_{i,j},vm_{x,y})]^{-1}$. The volatile factor of pheromone is denoted by $\rho \in (0, 1)$.

$\Delta\tau_{i,j,x,y}$ is the increment of the pheromone on the path $path(i,j,x,y)$ after one iteration. It is calculated as:

$$
\Delta\tau_{i,j,x,y} = \sum_{k=1}^{K} \Delta\tau_{i,j,x,y}^{k}
$$

Here $K$ is the size of the ant colony; $\Delta\tau_{i,j,x,y}^{k}$ is the pheromone on the path $path(i,j,x,y)$ left behind by $Ant_k$.

The evaluation function of the solutions for the scheduling problem in this paper is as follows:

$$
F(X) = MAKESPAN \times COST
$$

$\Delta\tau_{i,j,x,y}^{k}$ is related to the evaluation function of the solutions, calculated as:

$$
\Delta\tau_{i,j,x,y}^{k} = \begin{cases} 
\frac{1}{F(X^{k})}, & \text{if } path(i,j,x,y) \in PATH_k \\
0, & \text{otherwise}
\end{cases}
$$

Here $PATH_k$ is the complete path traveled by $Ant_k$; $X^k$ is the current solution corresponding to $PATH_k$.

### 4.4. Transition probability

The probability of $Ant_k$ selecting $path(i,j,x,y)$ is as follows:

$$
p_{i,j,x,y}^{k}(t) = \frac{[\tau_{i,j,x,y}(t)]^{\alpha} \times [\eta_{i,j,x,y}^{k}(t)]^{\beta}}{\sum_{vm_{x,y} \in VM_{AVL_SET}^{i,j}} [\tau_{i,j,x,y}(t)]^{\alpha} \times [\eta_{i,j,x,y}^{k}(t)]^{\beta}}, \text{if } vm_{x,y} \in VM_{AVL_SET}^{i,j} \subseteq VM_{AVL_SET}^{i,j}
$$

$$
0, \text{ otherwise}
$$
Here $VM_{AVL\_SET} =$ \{VMs with containers deployed\} $\cup$ \{Idle VMs with private cloud types\} $\cup$ \{Idle VMs with public cloud types\}; $VM_{AVL\_SET,i}^k$ is the subset of $VM_{AVL\_SET}$, which is the set of the VMs satisfying the resource requirements of container $cont_{i,j}$.

$\alpha$, $\beta$ are the regulating factors of pheromone and heuristic information in the transition probability, respectively.

### 4.5. Algorithm implementation

In this paper, we propose an Ant Colony Optimization algorithm for Container-based Microservice Scheduling in Hybrid Cloud (ACO_CMSHC for short). The ACO algorithm is shown in Algorithm 2.

**Algorithm 2: ACO_CMSHC**

Input: $Q$, CLOUD, APP, $K$ and $N_{\text{max}}$;  
Output: $PATH$;  
Begin  
$t=1$;  
Initialize pheromone matrix $\tau_{i,j,x,y}(t)$;  
$VM_{AVL\_SET} =$ \{All types of VMs in hybrid cloud\};  
while (true) 
for each Ant$_k$  
for each $cont_{i,j}$ in $Q$  
for each $vm_{x,y}$ in $VM_{AVL\_SET}$  
Calculate the heuristic information $\eta_{i,j,x,y}^k$ by Equation (6);  
end for  
for each $vm_{x,y}$ in $VM_{AVL\_SET}$  
Calculate the transition probability $p_{i,j,x,y}^k$ by Equation (11);  
end for  
According to the calculated probability, select a VM $vm_{x,y}$ and add path $(i,j,x,y)$ to $PATH_k$;  
if $vm_{x,y}$ is a new VM  
$VM_{AVL\_SET} = VM_{AVL\_SET} \cup \{vm_{x,y}\}$; Initialize pheromone on the new paths;  
end if  
end for  
Calculate the evaluation $F(\lambda^k)$ for complete path $PATH_k$ by Equation (9);  
Update the optimal solution $PATH$ found so far;  
end for  
$t=t+1$;  
if $t \leq N_{\text{max}}$  
for every path $(i,j,x,y)$  
Calculating pheromone increment $\Delta \tau_{i,j,x,y}$ by Equation (8);  
Update pheromone matrix $\tau_{i,j,x,y}(t)$ by Equation (7);  
end for  
If necessary, adjust the values of factors $\omega$, $\xi$ and $\psi$ according to the characteristics of the solutions generated by the ant colony in this iteration;  
for each Ant$_k$  
Empty $PATH_k$;  
end for  
else  
return $PATH$;  
end if  
end while  
End
5. Experimental evaluation

5.1. Experiment setup

5.1.1 Test data for the microservice application. The test data set for this paper refers to our previous work[14][14]. The test data set contains an application with 17 microservices. Section V(A)(1) of our previous work shows the parameters of microservices in the application, the number of requests, and the volume of data transmission among microservices when the application receives a unit of user service requests.

5.1.2 Test data for hybrid cloud. The reference configuration considers two data centers: data center A as private cloud offers 2 VM types (vm1 and vm2) and data center B as public cloud offers 3VM types (vm3, vm4, and vm5). The characteristics of the VM types are shown in Table 1 and Table 2.

| VM type | Calci | Stri | P (x)$ | si |
|---------|-------|------|--------|----|
| vm1     | 200   | 200  | 0      | 2  |
| vm2     | 150   | 150  | 0      | 1.5|
| vm3     | 300   | 250  | 0.3    | 3  |
| vm4     | 250   | 200  | 0.25   | 2.5|
| vm5     | 350   | 350  | 0.4    | 3.5|

Table 2. Network parameter setting between virtual machines

| Parameter                                | Value       |
|------------------------------------------|-------------|
| Network bandwidth between VMs in the private cloud(Mbps) | 250         |
| Communication delay between VMs in the private cloud(ms)  | 70          |
| Network bandwidth between VMs in the public cloud(Mbps)  | {300, 350, 400} |
| Communication delay between VMs in the public cloud(ms)  | {10, 30, 50}  |
| Network bandwidth between VMs across clouds(Mbps)       | 200         |
| Communication delay between VMs across clouds(ms)        | 90          |

5.1.3 Algorithm parameter setting

5.2. Algorithm for comparison

To verify the effectiveness of the ACO_CMSHC algorithm, the experiments compare the ACO_CMSHC algorithm with the ESMS algorithm[8] and the MSW_SDCOA algorithm[11] mentioned in related work. We compare the three algorithms above from two aspects: service finish time and resource cost of a single workflow with multiple container instances in the hybrid cloud.

5.3. Experimental results and analysis

To study our approach under different conditions, we executed the optimization process for seven experimental configurations: the number of user requests varies between x1.0 times and x4.0 times, with x0.5 times of user requests as a span. Ten tests are carried out under each experimental configuration, and take the average of the experimental results.
5.3.1 Resource cost.

Figure 1 shows the comparative results of the resource cost from the three algorithms with different number of user requests in the hybrid cloud. The figure indicates that with the increase of user requests, the resource costs of the three algorithms increase at different speeds. The MSW_SDCOA algorithm has the maximum growth and volatility because the MSW_SDCOA algorithm compresses the workflow based on data dependency to save scheduling cost and reduce the number of scheduling tasks, but the probability of low resource utilization increase by ignoring the resource requirements of the compressed tasks. When deploying multiple compressed tasks to the public cloud, the resource cost is relatively increased. The scheduling effect of the ESMS algorithm is the second best. The ACO_CMSHC algorithm has the lowest increase speed and the best optimization effect. With different user request configurations, the experiments compared the resource cost of the three algorithms under various finish time constraints. Figures 2, 3, 4, and 5 show the experimental results. The ACO_CMSHC algorithm has the best performance, followed by the ESMS algorithm.

5.3.2 Finish time.

Figure 6 shows the comparative results of the finish time from the three algorithms with different number of user requests in the hybrid cloud. The ESMS algorithm and the MSW_SDCOA algorithm focused on resource cost optimization under the deadline constraint. From Figure 6, the MSW_SDCOA algorithm is better than the ESMS algorithm on the whole. The ACO_CMSHC algorithm proposed in this paper fully considers the resource differences in the hybrid cloud, so the optimization performance is better than the MSW_SDCOA algorithm.
6. Conclusion
In this paper, we have established a workflow scheduling model of microservice based on multiple container instances in the hybrid cloud and proposed an ACO algorithm to optimize the finish time and the resource cost of the microservice application. In the ACO algorithm, to prevent the algorithm from falling into the local optimal solution and obtain the global optimal solution as soon as possible, the selection probability of the optimal path is improved by optimizing update strategies of heuristic information. Through the evaluation and feedback mechanism, improve the quality of offspring generation solutions as much as possible. Compared with other related algorithms, the ACO algorithm proposed performs best in reducing the finish time of the application and resource cost of the public cloud.

In future work, we plan to apply the proposed scheduling algorithm to a real cloud container cluster. And we will try to reduce the time complexity of the algorithm. In addition, it can also include other optimization objectives and other intelligent optimization algorithms, such as the genetic algorithm.

References
[1] Lewis J and Fowler M 2014 Microservices [Online] Available: https://martinfowler.com/articles/microservices.html.
[2] Merkel D 2014 Docker: Lightweight Linux containers for consistent development and deployment Linux J. 239 Art. no. 2.
[3] Chang Y S, Fan C T, Sheu R K, Jhu S R and Yuan S M 2017 An agent-based workflow scheduling mechanism with deadline constraint on hybrid cloud environment Journal of Communication Systems 31 pp 1-17
[4] Pasdar A , Almi’Ani K and Lee Y C 2018 Data-Aware Scheduling of Scientific Workflows in Hybrid Clouds ICCS 2018, LNCs 10862 pp 708-714.
[5] Mohammadi S, Pedram H, PourKarimi L 2018 Integer linear programming-based cost optimization for scheduling scientific workflows in multi-cloud environments The Journal of Supercomputing 74 pp 1-29.
[6] Gao T, Wang, Y, Wu C, Li R, Hou A and Xu M 2019 Minimizing financial cost of scientific workflows under deadline constraints in multi-cloud environments the 34th ACM/SIGAPP Symposium(Limassol, Cyprus) Part F1477 pp 114-121
[7] Zhang W, Liu Y, Wang L, Li Z and Goh R S M 2018 Cost-Efficient and Latency-Aware Workflow Scheduling Policy for Container-Based Systems IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)
[8] Wang S , Ding Z and Jiang C J 2020 Elastic Scheduling for Microservice Applications in Clouds IEEE Transactions on Parallel and Distributed Systems 32 pp 98-115
[9] Rajasekar P and Palanchiamy Y 2020 Scheduling multiple scientific workflows using containers on IaaS cloud Journal of Ambient Intelligence and Humanized Computing 2020(1).
[10] Jia Y, Chen W, Yuan H, Gu, T, Zhang H, Gao Y and Zhang J 2021 An Intelligent Cloud Workflow Scheduling System With Time Estimation and Adaptive Ant Colony Optimization IEEE Transactions on Systems, Man, and Cybernetics: Systems 51 pp 634-649
[11] Yuan Y W, Bao Z Q, Yu D J and Li W Q 2018 Multi-Scientific Workflow Scheduling Algorithm Based on Multi-Objective in Cloud Environment Journal of Software 29 pp 3326-3339
[12] Kianpisheh S, Charkari N M and Kargahi M 2016 Ant colony based constrained workflow scheduling for heterogeneous computing systems Cluster Computing 19 pp 1-18.
[13] Nasr A A , El-Bahnasawy N A , Attiya G and El-Sayed A 2019 Cost-Effective Algorithm for Workflow Scheduling in Cloud Computing Under Deadline Constraint Arabian Journal for Science and Engineering 44 pp3765-3780.
[14] Lin M, Xi J, Bai W and Wu J 2019 Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud IEEE Access 7 pp 83088-83100