End-to-End QoS Aggregation and Container Allocation for Complex Microservice Flows

Min Zhou, Yingbo Wu(✉), and Jie Wu

School of Big Data and Software Engineering, Chongqing University, Chongqing, China
1395234171@qq.com, wyb@cqu.edu.cn, 201824131016@cqu.edu.cn

Abstract. Microservice is increasingly seen as a rapidly developing architectural style that uses containerization technology to deploy, update, and scale independently and quickly. A complex microservice flow that is composed of a set of microservices can be characterized by a complex request execution path spanning multiple microservices. It is essential to aggregate quality of service (QoS) of individual microservice to provide overall QoS metrics for a complex microservice flow. Besides, leveraging the cost and performance of a complex microservice flow to find an optimal end-to-end container allocation solution with QoS guarantee is also a challenge. In this paper, we define an end-to-end QoS aggregation model for the complex microservice flow, and formulate the end-to-end container allocation problem of microservice flow as a nonlinear optimization problem, and propose an ONSGA2-DE algorithm to solve this problem. We comprehensively evaluate our modeling method and optimization algorithms on the open-source microservice benchmark Sock Shop. The results of experiments show that our method can effectively assist in the QoS management and container allocation of complex microservice flow.

Keywords: Microservice flow · Container allocation · Optimization · Quality of service

1 Introduction

Microservice architecture is a development model that has become more and more popular in recent years. Microservice-based applications typically involve the execution and interoperability of multiple microservices, each of which can be implemented, deployed, and updated independently without compromising application integrity [1,2]. In recent years, especially due to the proliferation of cloud technologies, the focus has been on applying workflow techniques on distributed environments. Motivated by this observation, we envision workflows that are composed of microservices, i.e., services which move away from the more

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 154–168, 2021.
https://doi.org/10.1007/978-3-030-67540-0_9

traditional large monolithic back-end applications by splitting the functionality into a set of smaller more manageable services [3]. Microservice flow, containing a set of microservices with data dependency between each other, is processed by multiple microservices that correspond to the tasks in the workflow. Microservice flow is a new manifestation of application independence. This independence in microservice flow management dramatically improves the scalability, portability, renewability, and availability of applications. Nevertheless, the price is high-cost overhead and complex resource usage dynamics [4]. Blindly allocating resources to the application to meet increasing loads is not cost-effective. Cloud providers would like to automatic elasticity management. Minimize the cloud resources allocated to an application to reduce operational cost while ensuring that the chosen cloud resources can support the service level agreement (SLA) agreed by the client. Currently, there is a lack of research on microservice flow in the field of microservices. Therefore, end-to-end quality of service (QoS) modeling [5] and container allocation optimization are one of the most critical concerns for complex microservice flows.

Different from traditional monolithic applications, end-to-end complex microservice flows bring two new challenges to QoS aggregation and container resource allocation. Firstly, the microservice flow is characterized by a complex request execution path spanning multiple microservices [6], forming a directed acyclic graph (DAG). Secondly, microservices run as containers hosted on virtual machines (VMs) clusters in a cloud environment, and application performance is often degraded in unpredictable ways, such as [8,9]. Traditional cloud providers support simple and relatively fast resource allocation for applications. It is not a simple task to run a specific application based on QoS requirements to determine an optimal and cost-effective container allocation strategy. Container allocation is a typical nonlinear optimization problem and has been proved to be an NP-hard problem [7]. An effective container resource allocation method not only satisfies the load requirements of the cluster but also ensures other QoS indicators, such as the reliability of the cluster. So far, minimal efforts have been directly devoted to QoS management and container allocation optimization for microservice flows.

Aiming at the above challenges, we define an end-to-end QoS aggregation model method for the complex microservice flow and represent the container allocation problem of microservice flow as a nonlinear optimization problem which minimizes the communication cost and the load balancing between containers under a user-specified response time constraint, and propose an ONSGA2-DE algorithm to solve this problem. In particular, we make the following contributions.

(a) End-to-end QoS aggregation can provide an overall QoS metric evaluation method for complex microservice flows. To the best of our knowledge, this is the first attempt at designing an end-to-end QoS aggregation model for the complex microservice flow.
(b) End-to-end container allocation provides an overall configuration optimization solution for complex microservice flows that meet SLAs. We represent

the container allocation problem of microservice flow as a nonlinear optimization problem and solve effectively by the proposed ONSGA2-DE algorithm.
(c) We demonstrate the feasibility of using the proposed QoS aggregation model and container allocation strategy to make effective container scaling and achieve container allocation optimization of microservice flow.

The rest of this paper is organized as follows. In Sect. 2, we introduce our system model. In Sect. 3, we propose a heuristic algorithm for container allocation optimization. Next, in Sect. 4, performance evaluations of our solution are presented, and Sect. 5 discusses related research. Section 6 concludes the article.

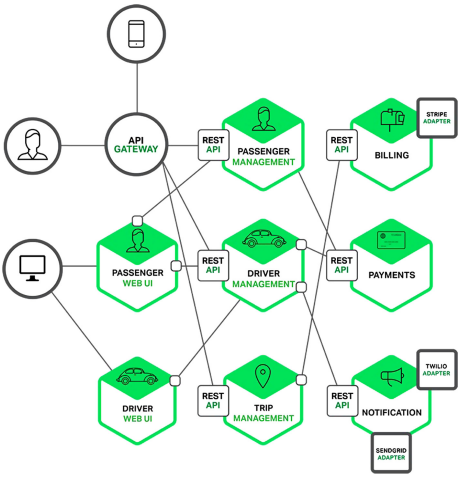


Fig. 1. Microservice composition and deployment [10].

2 System Model

2.1 Microservice Flow

As shown in Fig. 1, the microservice architecture [10] defines a framework that structures an application structure as a set of loosely coupled, collaborative, and interconnected services, each implementing a set of related functions. A workflow is defined as "the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules" by the workflow management coalition (WFMC) [11]. In particular, each workflow is implemented as a tightly coupled set of tasks and has its workflow execution plan that specifies how to run the workflow on a distributed computation infrastructure [12].

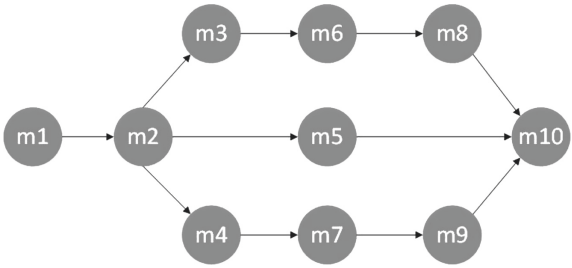


Fig. 2. Microservice flow with ten microservices.

Definition 1. *Microservice Flow.*

The microservice flow is the set of microservices and the interoperability between them. These interoperability relationships are established when microservices consume results generated by other microservices. According to the definition of cloud workflow, a microservice flow $W = (M, E)$ is modeled as a DAG where the set of vertices $M = \{m_1, m_2, \cdots, m_n\}$ represents microservice ms_i, and E is a set of directed edges that represent data or control dependencies between the microservices. A dependency e_{ij} is a precedence constraint of the form (m_i, m_j), where, $m_i, m_j \in M$, and $m_i \neq m_j$. It implies that microservice ms_j (child microservice) can only be started after the microservice ms_i (parent microservice) has completed execution and transferred reliant data to ms_j. Thus, a child microservice cannot execute until all its parent microservices have finished execution and satisfied the required dependencies. For a complex microservices flow, we abstract it, as shown in Fig. 2. Each microservice ms_i takes responsibility to complete a specific subfunction for requests to the microservice flow. Each microservice is deployed and scaled independently without impacting the rest of the application.

Each microservice is characterized as a tuple $(msreq_i, res_i, QoS_i, set_i)$, where $msreq_i$ is the number of microservice id i requests required to satisfy one application workload; res_i represents the computational resources consumed to satisfy one request of the microservice id i; QoS_i represents the QoS parameters for an instance of the microservice id i; set_i is the set of consumer microservices for the microservice ms_i. Each microservice is executed in the system encapsulated in one or more containers $cont_k$. Similar to existing work, it is guaranteed that each request for microservice ms_i would be assigned to one container.

2.2 QoS Aggregation Model

In a workflow management environment, a modeller will create an abstract workflow description, which is used for the integration of available services into the workflow. In the case of a workflow consisting of services only, the result of the integration process can be seen as a service composition [13]. Since the motivation for this work is to use the composition of microservice in the workflow domain,

Table 1. QoS aggregation model for four composition modes.

Parameters	Sequential	Parallel	Conditional	Loop
Response time (Q_{RT})	$\sum_{i=1}^{n} q_{RT}\|ms_i\|$	$max(q_{RT}\|ms_1\|, q_{RT}\|ms_2\|, \cdots, q_{RT}\|ms_n\|)$	$\sum_{i=1}^{n} pr_i * q_{RT}\|ms_i\|$	$h * q_{RT}\|ms_i\|$
Reliability (Q_{Rel})	$\sum_{i=1}^{n} q_{Rel}\|ms_i\|$	$max(q_{Rel}\|ms_1\|, q_{Rel}\|ms_2\|, \cdots, q_{Rel}\|ms_n\|)$	$\sum_{i=1}^{n} pr_i * q_{Rel}\|ms_i\|$	$h * q_{Rel}\|ms_i\|$
Availability (Q_{Ava})	$\sum_{i=1}^{n} q_{Ava}\|ms_i\|$	$max(q_{Ava}\|ms_1\|, q_{Ava}\|ms_2\|, \cdots, q_{Ava}\|ms_n\|)$	$\sum_{i=1}^{n} pr_i * q_{Ava}\|ms_i\|$	$h * q_{Ava}\|ms_i\|$

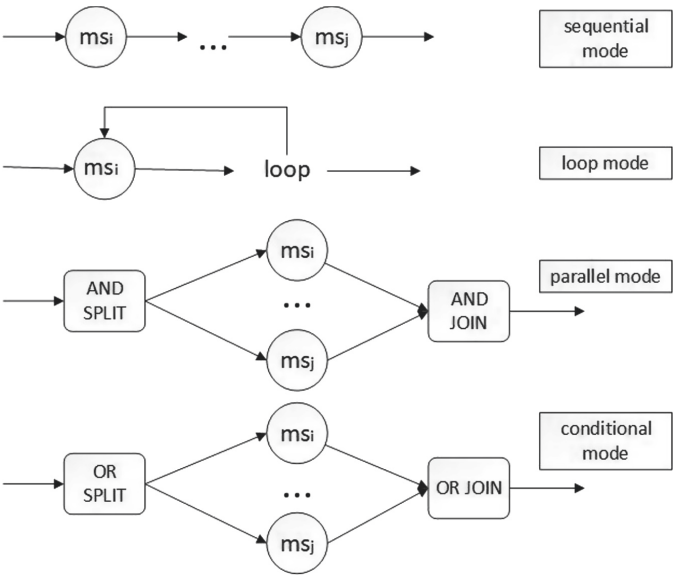


Fig. 3. Different patterns of the composition model.

the composition patterns are based on structural elements as used in workflow descriptions. Services in a microservice flow can be composed based on four basic modes: sequential mode, loop mode, conditional mode, and parallel mode, as shown in Fig. 3. A complete microservice flow can be easily decomposed into these four modes. Applying this procedure, the aggregation can be performed on the basis of each composition pattern. [13] has presented a composition model that can serve as a basis for aggregation of service properties regarding QoS dimensions. Inspired by it, this paper uses QoS aggregation to model the QoS of the entire microservice flow. There are five aggregation operations for developing an aggregation formula, including summation, average, multiplication, min, and max. Here, we choose QoS_i as $(q_{RT_i}, q_{Rel_i}, q_{Ava_i})$. These three indicators are the most representative of QoS. Based on them, the aggregation method of microservice flow is given. Table 1 presents the response time, the reliability,

and the availability calculation formulas based on the four basic relations of sequential, parallel, conditional, and loop, respectively, where n is the number of microservice.

3 Solution: Container Allocation Optimization

3.1 Optimization Model

Container allocation mechanisms of microservice flow are based not only on the resource capacity but also consider the nature of the workload. Each microservice of the microservice flow has multiple running instances, and each microservice instance is encapsulated in one container. The scalability of microservice flow means that the container allocation can be appropriately scaled according to the resource usage rate, ensuring that some microservice instances would not become the performance bottleneck of system. In this paper, we focus on the allocation of container with QoS guarantee, so the communication cost and load balancing are formulated as optimization objectivities, and the QoS of response time is modeled as the QoS constraint. If some microservice flow applications have high QoS requirements for the availability and reliability, the optimization model can also include the QoS of availability and reliability as constraints. Table 2 presents the key notations.

The first optimization objective is the communication cost. The communication cost among microservices is related to two key factors: the network distance between containers assigned to two interoperable microservices, and the number of requests between the two microservices. Consider that consumer microservices and provider microservices may run multiple container instances simultaneously. This paper uses the average network distance of all the container pairs between consumer microservice and provider microservice to calculate the communication cost between two microservices. The communication cost for node j to communicate with other nodes is formalized in (1).

$$commCost_j = \sum_{i=1}^{n} \frac{x(j,i)}{scale_i} \sum_{ms_k \in set_i \wedge m \in list(ms_k)} dist(j,m) \tag{1}$$

For the second optimization objective, we consider that the cluster needs to be load balanced. The cluster is balanced when the use of the nodes is uniform across the cluster, i.e., the consumption of computational resources is uniformly distributed among the VMs [14]. A high $scale_i$ of a microservice is penalized when the container is underused, and a low $scale_i$ is penalized when the number of microservice requests is very high. With $scale_i$ changes, we can achieve automatic elasticity management. We use the standard deviation of computing resource usage to evaluate the balance of the cluster. The resources consumed by each node are formalized in (2).

$$nodeUsage_j = \sum_{i=1}^{m} \frac{ureq \times msreq_i \times res_i}{scale_i} \times x(j,i) \tag{2}$$

Table 2. Summary of the system model.

Parameter	Description
$ureq$	Number of user workloads
$msreq_i$	ms_i requests number needed for each workload
res_i	Computational resources required for a ms_i request
QoS_i	QoS parameters for a ms_i instance
set_i	The set of consumer microservice for ms_i
$scale_i$	Number of containers for ms_i
$dist(i,j)$	Network distance between vm_i and vm_j
$x(j,i)$	The number of container assigned on vm_j for microservice ms_i
$list(ms_i)$	List of containers to which ms_i is allocated
Q_{RT}	Response time for the entire microservice flow
Q_{Rel}	Reliability for the entire microservice flow
Q_{Ava}	Availability for the entire microservice flow
T	The response time for the entire microservice flow specified in the service level agreement
pr_i	Conditional probability
h	Loop times
ms_i	Microservice with id.i
$cont_k$	Container with id.k
vm_j	Virtual machine with id.j

To summarize, our aim can be defined as a multi-objective optimization problem, where the solution establishes the allocation of containers to the VMs, by minimizing (i) the communication cost and (ii) the load imbalance between containers. We formulate the optimization problem as follows: Given a microservice flow instance and a set of available VMs, we wish to find an allocation $A(t) : ms_i(cont_k) \rightarrow vm_j, vm_j \in VMs$, by minimizing

$$\sum_{j \in N} commCost_j, \tag{3}$$

$$\sigma(nodeUsage_1, nodeUsage_2, \cdots, nodeUsage_j), \tag{4}$$

subject to the constraint

$$Q_{RT} < T, \tag{5}$$

$$\sum_{j \in N} x(j,i) = scale_i. \tag{6}$$

This optimization problem is NP-complete because all the possible container allocation strategies should be evaluated to find the optimal solution. A metaheuristic method must be employed to address the problem [15].

Table 3. ONSGA2-DE algorithm execution parameters.

Parameter	Value
populationSize	200
generationNumber	200
Mutation probability	0.25
Scale factor F	2
Control ratio k	rand(0,1)
P_{cmax}	1
P_{cmin}	0

3.2 Proposed Container Allocation Algorithm

The container allocation problem is represented as a multi-objective optimization problem. The Non-dominated Sorting Genetic Algorithm-II (NSGA2), which is considered a standard approach for multi-objective optimization problems in resource management problem for container allocation. NSGA2 [16] has high computational efficiency and high-quality solution set when processing the low-dimensional multi-objective optimization problem. In Differential Evolution (DE) Algorithm, the most novel feature is its mutation operation. In the initial stage of the algorithm iteration, the individual differences of the population are considerable. Such a mutation operation will make the algorithm have robust global search ability. By the end of the iteration, the population tends to converge, and the differences between individuals decrease, which makes the algorithm have strong local search capabilities. In order to further improve the diversity of the population and prevent the algorithm from converging to a local optimum, we propose a heuristic ONSGA2-DE algorithm in this paper. The algorithm introduces opposition-based learning in the iterative process of the population, and a crossover and mutation strategy of DE algorithm is applied as the operator. The ONSGA2-DE algorithm increases the diversity of the population and prevents local convergence.

Opposition-Based Learning. Apply the opposition-based learning to the evolution process, calculate the reverse individual X^* for the individual X in each generation of the evolution process. During opposition-based learning, the population is evolved towards better fitness value by the specific evolutionary strategy. And the better individuals are selected from both the original population and the opposite population to produce the next generation. Similarly, an opposite point in the multi-dimensional case can be defined as follows [17]: Let $P(x_1, x_2, \cdots, x_D)$ be a point in D- dimensional space, where, $x_1, x_2, \cdots, x_D$ are real numbers and $x_j \in [a_j, b_j], j = 1, 2 \cdots, D$. The opposite point of P^* is denoted by $P^*({x_1}^*, {x_2}^*, \cdots, {x_D}^*)$ where

$$ {x_j}^* = a_j + b_j - x_j \tag{7} $$

Algorithm 1. ONSGA2-DE

1: **INPUT:** $populationSize, generationNumber,$
2: $mutationPro, P_{cmax}, P_{cmin}, F, k$
3: **OUTPUT:**Pareto optimal
4: Generate randomly P_t of size $populationSize$, $t = 0$
5: **for** $t < generationNumber$ **do**
6: **if** $rand(0, 1) > P_c$ **then**
7: **for** $i < populationSize$ **do**
8: for $child_i, father_1, father_2, father_3$ are selected by Tournament
9: ${child_i}^{new} = father_1 + F(father_2 - father_3)$
10: **if** $random() < mutationPro$ **then**
11: $mutation({child_i}^{new})$
12: **end if**
13: $Q_t = Q_t \cup {child_i}^{new}$
14: **end for**
15: **else**
16: **for** $i < populationSize$ **do**
17: ${child_i}^{new} = k(a_i + b_i) - child_i$
18: $Q_t = Q_t \cup {child_i}^{new}$
19: **end for**
20: **end if**
21: $R_t = Q_t \cup P_t$
22: $fitness = calculateFitness(R_t)$
23: $fronts = fastNonDominatedSort(R_t, fitness)$
24: $distances = calculateCrowdingDistances(R_t, fronts)$
25: $P_{t+1} = SelectTopN(R_t, fronts, distances)$
26: $t = t + 1$
27: **end for**
28: $Solution = fronts[1]$

However, considering that in the later stage of the algorithm, the population P has formed a certain rule and is close to the optimal solution region. It is of little significance to solve the reverse population in the later stage of the algorithm and the running speed of the algorithm is slowed down. In every generation of evolution, its opposite individual X^* is calculated by a certain probability of opposition, and in the process of evolution, P_c(probability) is linear decline. Therefore, the following method is designed in this paper:

$$P_c = P_{cmax} - \frac{t}{populationSize}(P_{cmax} - P_{cmin}) \tag{8}$$

In each algorithm iteration, a set of new individuals, with the same size as the population, is generated (lines 6 to 20 in Algorithm 1). Including the crossover and mutation of the DE algorithm to generate a new population (lines 7 to 14). And generate a new population through opposition-based learning (lines 16 to 19). These new individuals are joined with the individuals in the parent population (line 21), obtaining a set that is double the size of the population. Only half of the individuals need to be selected for the offspring. The front

levels and crowding distances of the new set are calculated and applied to order the solution set (lines 22 to 25). The best half of the solutions are selected as individuals of the offspring, and the remainder are rejected. The time complexity of the algorithm is related to the population P and the number of optimization objectives. The time complexity is $O(MN^2)$, N represents the population size, and M represents the number of optimization objectives. Table 3 summarizes the parameters for the execution of the ONSGA2-DE algorithm.

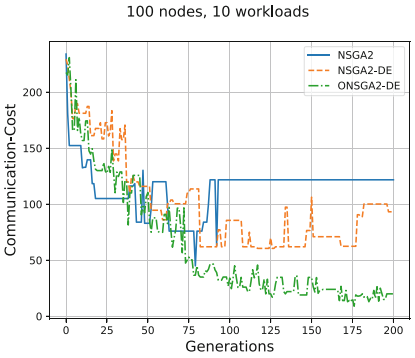
4 Experiment

For container allocation of microservice flows, we use Sock Shop. It is an open-source microservices benchmark specifically for container platforms and a microservices demo application. The Sock Shop simulates an e-commerce site, with the specific goal of helping to demonstrate and test existing microservices and cloud-native technologies. The parameter values for the microservices stack was estimated from [7].

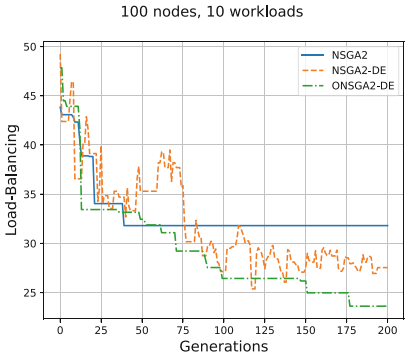
In order to study the results of our method under different experimental conditions, we set up several different experimental configurations. Workload changes |workloads| = [10, 20, 30, 40] reflect the flexible management of the application. The capacity of the cluster is the number of available |VMs| = [100, 200] and their computational capacities cap = 800. Network distance among the nodes is $dist(i, j)$ = [1.0, 4.0]. The simulation $q_{RT}(ms_i)$ = 0.458, 0.891, 0.514, 0.17, 0.523, 0.364, 0.683, 0.371, 0.236, 0.537. Using the QoS aggregation formula, we calculate the Q_{RT} of the microservice flow instance. And it is less than the response time specified in the service level agreement. The comparisons of the three algorithms are performed in two aspects: communication cost, and load balancing of containers. Among them, we need to normalize two optimization objectives. This $Solution(x)$ was calculated as the weighted average of the normalized values of the two objectives in (9). $max_{commCost}$, $min_{commCost}$ are the maximum and minimum values of the communication cost, respectively. $max_{nodeUsage}$, $min_{nodeUsage}$ are the maximum and minimum values of the standard deviation of the resource usages of the nodes.

$$Solution(x) = \frac{1}{2} \times \frac{commCost(x) - min_{commCost}}{max_{commCost} - min_{commCost}} + \frac{1}{2} \times \frac{nodeUsage(x) - min_{nodeUsage}}{max_{nodeUsage} - min_{nodeUsage}} \tag{9}$$

As shown in Fig. 4(a), (b), experiments with ten workloads, and 100 nodes are performed. We can observe the change of the iterative process of two optimization objectives under constraints. Figure 4(a) is an iterative process of communication cost for the three algorithms. Communication cost is measured in terms of the average network distance and request number between all interacting microservices. We want two microservice instances with interaction to be distributed on the same node or adjacent nodes. This can reduce communication cost. We

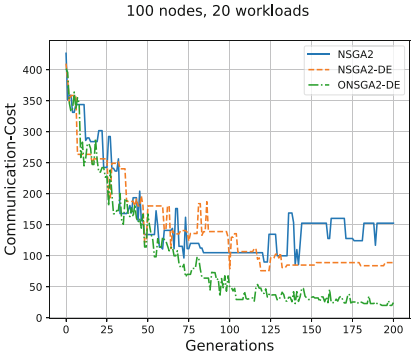


(a) Communication Cost

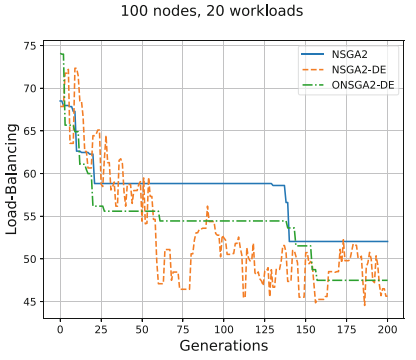


(b) Load Balancing

Fig. 4. Values of the two objective functions throughout the algorithm generations for the experiment with 10 workloads and 100 VMs.

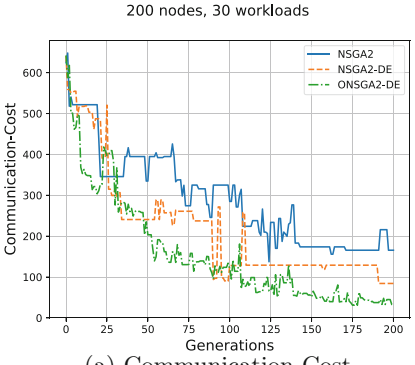


(a) Communication Cost

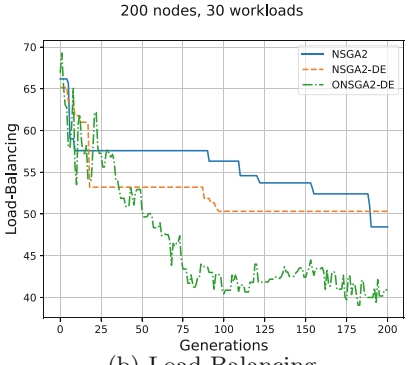


(b) Load Balancing

Fig. 5. Values of the two objective functions throughout the algorithm generations for the experiment with 20 workloads and 100 VMs.

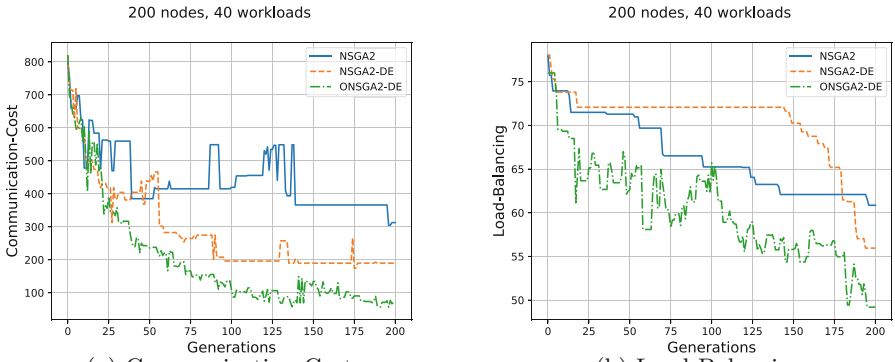


(a) Communication Cost



(b) Load Balancing

Fig. 6. Values of the two objective functions throughout the algorithm generations for the experiment with 30 workloads and 200 VMs.



(a) Communication Cost (b) Load Balancing

Fig. 7. Values of the two objective functions throughout the algorithm generations for the experiment with 40 workloads and 200 VMs.

clearly see that communication cost decreases as iteration progress. This also means that the distribution of nodes is ideal. Figure 4(b) shows the iterative process of the standard deviation. The standard deviation value represents the degree of balance of the cluster load. We can see that the standard deviation value gradually decreases with the process of iteration, which also means that the load between the containers is approaching equilibrium. From Fig. 4(a), (b), we can see that the NSGA2 algorithm performs worst, and the ONSGA2-DE algorithm performs best. Our proposed algorithm performs better than the contrasted algorithms in terms of convergence speed and optimal value. Because the ONSGA2-DE algorithm uses reverse learning, it has optimized the diversity and convergence of the population. The crossover and mutation stages of the ONSGA2-DE algorithm are combined with the DE algorithm. Then adds DE operator to generate new individuals, which improves the generation stage. So it is better than NSGA2-DE.

Figure 5 shows the comparative results of the two objective functions for the three algorithms with 20 workloads and 100 nodes. Increased user requests and the values of the two objective functions increased correspondingly. We increase the number of workloads to 30 and 40, and the number of nodes to 200 and observe the changes in the two objective values again in Fig. 6, Fig. 7. In general, we can see that the ONSGA2-DE algorithm works best. The ONSGA2-DE algorithm can find an optimal solution, and the value of the optimization objective is always better than the other two algorithms.

5 Related Work

QoS management for cloud applications has been an important research topic for many years [18]. Due to many complex interactions and performance changes caused by cloud computing, significant challenges are presented in modeling system performance, identifying key resource bottlenecks, and effective

management [19,20]. From web services to cloud services, QoS management research focuses on QoS requirements specification, and QoS service selection [21] that is based on QoS aggregation technology [13]. End-to-end QoS aggregation model can provide an overall QoS metric evaluation method for complex microservice flows. Cloud providers would like to minimize the cloud resources allocated to an application to reduce operational cost while ensuring that the chosen cloud resources can support the SLA.

Since entering the era of cloud computing, resource allocation methods in the cloud have been widely studied. However, more research on cloud application mainly focuses on resource allocation of VMs to achieve performance-oriented load balancing or energy-oriented load integration [22]. With the development of container technology, some practical container allocation solutions are proposed, but there are still some important problems to be solved in container-based microservice management. Container allocation will improve the overall performance and resource utilization of the system. This characteristic of container allocation helps to reduce energy consumption and achieve better system improvements and user satisfaction. Some research focused on the optimization of resource management in the field of microservice applications [9,23]. According to some of the surveys released, it included a wide range of techniques for addressing resource management issues, such as scheduling [27], provisioning [24], and distribution [25]. We can describe the container allocation problem as a multi-objective optimization problem, which can be solved by evolutionary algorithms [26]. Also, genetic algorithms are often used in resource management of cloud environments. The adequacy of genetic algorithms, especially NSGA2, has been widely proven [16]. Guerrero et al. [7] used genetic algorithms to optimize container allocation in cloud structures, which enhanced system provisioning, system performance, system failure, and network overhead. Bao et al. [5] proposed a performance modeling and workflow scheduling method for microservice-based applications in the cloud, and a heuristic scheduling scheme was used for an application containing multiple microservices. Gribaudo et al. [9] proposed a performance evaluation method for applications based on large-scale distributed microservices.

To the best of our knowledge, the proposed method differs from the existing efforts in three aspects: (i) Our work is oriented end-to-end complex microservice flows. (ii) We propose an end-to-end QoS aggregation model and container allocation method for the complex microservice flow. (iii) We prove that the heuristic ONSGA2-DE algorithm implements a container allocation strategy and automatic elasticity management.

6 Conclusion

Microservice architecture styles are becoming increasingly popular in both academia and industry. QoS management and container allocation are two critical issues for microservice flows. In this paper, we have proposed a QoS aggregation model based on microservice flow, formulate a microservice flow container

allocation problem to minimize communication cost and load imbalance between containers, and solve the problem by using ONSGA2-DE algorithm. Our work on QoS management and container resource optimization has been validated and evaluated through experimental and simulation results. Experimental results show that our model can find an optimal decision scheme to guarantee the QoS of microservice flow.

Future work will be pursued in several directions. Firstly, we assumed that VMs had the same capacity, but in reality, there were different types of VMs. In future work, we will consider the effects of changes in the capacity and type of VMs on the experimental results. Secondly, this paper focuses on the QoS management and container resource allocation of a single flow consisting of multiple microservices. Therefore, microservices can be maintained separately, scaled, or discarded as necessary. The problem of allocation for multiple complex microservice flows is the direction of our future research.

Acknowledgment. This work was supported in part by National Key Research and Development Project under grant 2019YFB1706101, in part by the Science-Technology Foundation of Chongqing, China under grant cstc2019jscxmbdx0083.

References

1. Dragoni, N., et al.: Microservices: yesterday, today, and tomorrow. Present and Ulterior Software Engineering, pp. 195–216. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67425-4_12
2. Jamshidi, P., Pahl, C., Mendonca, N.C.: Microservices: the journey so far and challenges ahead. IEEE Softw. **35**(3), 24–35 (2018)
3. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D.: Examining the challenges of scientific workflows. Computer **40**(12), 24–32 (2007)
4. Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L., Villari, M.: Open issues in scheduling microservices in the cloud. IEEE Cloud Comput. **3**(5), 81–88 (2016)
5. Bao, L., Wu, C., Bu, X.: Performance modeling and workflow scheduling of microservice-based applications in clouds. IEEE Trans. Parallel Distrib. Syst. **30**(9), 2114–2129 (2019)
6. Rahman, J., Lama, P.: Predicting the end-to-end tail latency of containerized microservices in the cloud. In: IEEE International Conference on Cloud Engineering, pp. 200–210 (2019)
7. Guerrero, C., Lera, I., Juiz, C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. J. Grid Comput. **16**(1), 113–135 (2018). https://doi.org/10.1007/s10723-017-9419-x
8. Barakat, S.: Monitoring and analysis of microservices performance. J. Comput. Sci. Control Syst. **5**(10), 19–22 (2017)
9. Gribaudo, M., Iacono, M., Manini, D.: Performance evaluation of massively distributed microservices based applications. In: European Council for Modelling and Simulation (ECMS), pp. 598–604 (2017)
10. Pattern: Microservice architecture (2019). http://microservices.io/patterns/microservices.html
11. Wiley, J.: Workflow Handbook (2019). http://pl.wikipedia.org/wiki/Workflow

12. Liu, L., Zhang, M., Lin, Y., Qin, L.: A survey on workflow management and scheduling in cloud computing. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE (2014)
13. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS aggregation for web service composition using workflow patterns. In: 8th IEEE International Conference on Enterprise Distributed Object Computing, pp. 149–159 (2004)
14. Rusek, M., Dwornicki, G., Orłowski, A.: A decentralized system for load balancing of containerized microservices in the cloud. In: Świątek, J., Tomczak, J.M. (eds.) ICSS 2016. AISC, vol. 539, pp. 142–152. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-48944-5_14
15. Wei, G., Vasilakos, A.V., Zheng, Y., Xiong, N.: A game-theoretic method of fair resource allocation for cloud computing services. J. Supercomput. **54**(2), 252–269 (2010). https://doi.org/10.1007/s11227-009-0318-1
16. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
17. Tizhoosh, H.R.: Opposition-based learning: a new scheme for machine intelligence. In: International Conference on Computational Intelligence for Modelling, Control and Automation, pp. 695–701. IEEE (2005)
18. Ghosh, Q., Longo, F., Naik, V.K., Trivedi, K.S.: Modeling and performance analysis of large scale IaaS clouds. Future Gener. Comput. Syst. **29**(5), 1216–1234 (2013)
19. Vakilinia, Q., Ali, M.M., Qiu, D.: Modeling of the resource allocation in cloud computing centers. Comput. Netw. **91**, 453–470 (2015)
20. Jindal, A., Podolskiy, V., Gerndt, M.: Performance modeling for cloud microservice applications. In: 10th ACM/SPEC International Conference on Performance Engineering (ICPE 2019), pp. 25–32 (2019)
21. Liangzhao, Z., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30**, 311–327 (2004)
22. Yuan, H., Li, C., Du, M.: Optimal virtual machine resources scheduling based on improved particle swarm optimization in cloud computing. J. Softw. **9**(3), 705–708 (2014)
23. Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., Steinder, M.: Performance evaluation of microservices architectures using containers. In: IEEE 14th International Symposium on Network Computing and Applications (NCA), pp. 27–34. IEEE (2015)
24. Khazaei, H., Barna, C., Beigi-Mohammadi, N., Litoiu, M.: Efficiency analysis of provisioning microservices. In: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 261–268. IEEE (2016)
25. Akhter, N., Othman, M.: Energy aware resource allocation of cloud data center: review and open issues. Clust. Comput. **19**(3), 1163–1182 (2016). https://doi.org/10.1007/s10586-016-0579-4
26. Lucken, C., Baran, B., Brizuela, C.: A survey on multi-objective evolutionary algorithms for many-objective problems. Comput. Optim. Appl. **58**(3), 707–756 (2014). https://doi.org/10.1007/s10589-014-9644-1
27. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing: issues and challenges. J. Grid Comput. **14**(2), 217–264 (2016). https://doi.org/10.1007/s10723-015-9359-2