Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6621094, 11 pages
https://doi.org/10.1155/2021/6621094

Research Article

ACEA: A Queueing Model-Based Elastic Scaling Algorithm for Container Cluster

Kui Li,[1,2] **Yi-mu Ji**,[1,3,4,5] **Shang-dong Liu**,[1] **Hai-chang Yao**,[1] **Hang Li**,[1] **Shuai You**,[1] **and Si-si Shao**[1]

[1]*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[2]*The Second People's Hospital of Nantong, Nantong 226002, China*
[3]*Nanjing Center of HPC China, Nanjing 210023, China*
[4]*Institute of High Performance Computing and Bigdata, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[5]*Jiangsu HPC and Intelligent Processing Engineer Research Center, Nanjing 210003, China*

Correspondence should be addressed to Yi-mu Ji; jiym@njupt.edu.cn

Received 17 December 2020; Revised 19 January 2021; Accepted 10 May 2021; Published 26 May 2021

Academic Editor: Yuanpeng Zhang

Elastic scaling is one of the techniques to deal with the sudden change of the number of tasks and the long average waiting time of tasks in the container cluster. The unreasonable resource supply may lead to the low comprehensive resource utilization rate of the cluster. Therefore, balancing the relationship between the average waiting time of tasks and the comprehensive resource utilization rate of the cluster based on the number of tasks is the key to elastic scaling. In this paper, an adaptive scaling algorithm based on the queuing model called ACEA is proposed. This algorithm uses the hybrid multiserver queuing model (M/M/s/K) to quantitatively describe the relationship among number of tasks, average waiting time of tasks, and comprehensive resource utilization rate of cluster and builds the cluster performance model, evaluation function, and quality of service (QoS) constraints. Particle swarm optimization (PSO) is used to search feasible solution space determined by the constraint relation of ACEA quickly, so as to improve the dynamic optimization performance and convergence timeliness of ACEA. The experimental results show that the algorithm can ensure the comprehensive resource utilization rate of the cluster while the average waiting time of tasks meets the requirement.

1. Introduction

With the advent of the information age, information technology has been widely used in various fields of human life, such as medical big data analysis in the medical field [1]. And the resulting application services have also been growing explosively. To provide an environment for an effective service running environment, cloud computing platforms based on virtualization technology emerge. Cloud computing refers to applications and services running on a distributed network using virtualized resources [2–4]. Virtualization is used to build virtual hosts running different operating systems on the same physical machine, while applications and services run on these different virtual machines as needed [5, 6]. In the early days of cloud computing [7], it was common to build cloud computing clusters based on traditional virtual machine clusters. As Docker container technology [8, 9] is maturing day by day, the way of cloud platform construction has gradually changed into the shaping of a Docker cluster through the integration of multiple Docker physical nodes [10]. Compared with the traditional virtualization architecture [11–14], the container has the characteristics of low resource consumption, fast startup speed, high deployment efficiency, and good scalability, which can ensure the reliability and timeliness of the elastic scaling of resources for the cluster. However, the following problems still exist in the elastic scaling:

(1) The container cluster has the characteristics of large number of resource indicators and complex

relationship between indicators, so it is difficult to analyze the relationship between indicators quantitatively. At this time, if the performance model and evaluation function [15–17] cannot be reasonably built according to user demand, system resource consumption, and other indicators, it is easy to have unreasonable allocation of resources

(2) The number of tasks in the Internet environment has the characteristics of mutation, that is, the number of tasks arriving is irregular and sudden. When the elastic scaling algorithm cannot allocate resources according to the number of tasks in time, the task may be lost due to the long average waiting time of tasks or the resource waste caused by the cluster idling due to the insufficient number of tasks

Therefore, it is urgent to solve the problem of how to quantitatively describe the relationship among number of tasks, average waiting time of tasks, and comprehensive resource utilization rate of cluster, for the purpose of ensuring that the comprehensive resource utilization rate of cluster is always at a high level on the basis of controllable average waiting time of tasks. In order to solve the problem, this paper proposes the ACEA. The main contributions of this algorithm are as follows:

(1) According to the state information of the container cluster and the application characteristics it carries, building a self-defined QoS constraint relationship and feasible solution space to provides computing constraints for the resources elastic scaling

(2) The paper studies the problem of resource elastic scaling in container cluster, introduces the calculation method of task effective arrival rate, and proposes an adaptive elastic capacity expansion framework and performance model based on a queuing model. This paper uses M/M/s/K to describe the relationship among the number of tasks, the average waiting time of tasks, and the comprehensive utilization of cluster resources and solves the problem of building the performance model and evaluation function of a container cluster

(3) Taking the cluster evaluation function as the fitness function, through the particle swarm optimization algorithm to search the feasible solution space, improves the dynamic optimization and convergence timeliness of ACEA algorithm, and achieves the goal of improving the accuracy of resource elastic scaling under the condition of ensuring the convergence timeliness of the algorithm

The structure of this paper is as follows. Section 2 summarizes the research work on the elastic scaling of container cluster. Section 3 discusses the overall design of the ACEA algorithm. Section 4 discusses the M/M/s/K performance model design and adaptive scaling strategy of the ACEA algorithm. Section 5 validates the overall design of Section 3 and the performance model and cluster scheduling strategy of Section 4 through experiments. Section 6 is the conclusions.

2. Related Work

At present, the research mainly provides appropriate resources for the task under the premise of ensuring the shortest task execution time [18–20]. For example, [18] proposes a cloud environment task scheduling algorithm based on the multipriority queue and memory algorithm (MPQMA). The basic idea of this method is to improve the convergence speed with the advantages of MA. [19] provides a comprehensive multiobjective optimization task scheduling model to minimize execution time, delivery time, and execution cost. However, the scheduling model has conflicts of objective function parameters, and there may be the issue of timeliness in multiobjective optimization. [20] proposes a method for automatically testing the entire cloud environment using containers, which serves as the foundation of distributed cloud monitoring. In [21], the cluster elastic scaling technique is divided into reaction scaling and prediction scaling. Reaction scaling refers to the dynamic scaling of a cluster when a burst task request occurs. Prediction scaling refers to predicting task size based on historical data and prediction algorithms and dynamically scale before the change takes place. [22] proposes a layer-by-layer elastic scaling technique for applications. [23] discusses the application of optimization algorithms in load balancing and elastic scheduling. [24] proposes the application of the integrated MOPSO algorithm in task scheduling and optimizes the total task time and average task time. [25–28] introduce three resource scheduling methods provided by Docker Swarm: spread strategy, binpack strategy, and random strategy. The spread strategy is the default strategy. Docker Swarm prefers nodes with the fewest resources (such as CPU and memory) to ensure uniform use of all node resources in the cluster. The binpack strategy is the opposite of spread strategy, and its purpose is to use one node as much as possible to ensure enough idle nodes; the random strategy is a random selection strategy, that is, the task is completely randomly assigned to the existing nodes. Based on the theoretical analysis, the corresponding advantages and disadvantages of the scheduling algorithm are discussed. Among them, [25, 26] introduce Docker Swarm, the most widely used Docker cluster management tool, and provide the spread strategy as the default scheduling strategy. Docker Swarm selects the least quantity of resources to consume according to the number of CPU cores of a node and the unallocated memory; [28] proposes a task scheduling technique based on the genetic algorithm, which effectively allocates cloud computing resources and minimizes the overall response time. [29–31] detail the container orchestration tool Kubernetes and its elastic scaling function. The elastic scaling of Kubernetes can dynamically adjust the number of Pod copies for purpose of scaling the container according to the number of tasks. The number of Pod copies is adjusted by periodically querying the status of the Pod to obtain the monitoring data of the Pod, and then, the average usage rate and target usage rate of the existing Pod is compared to determine the number of scaling s. The

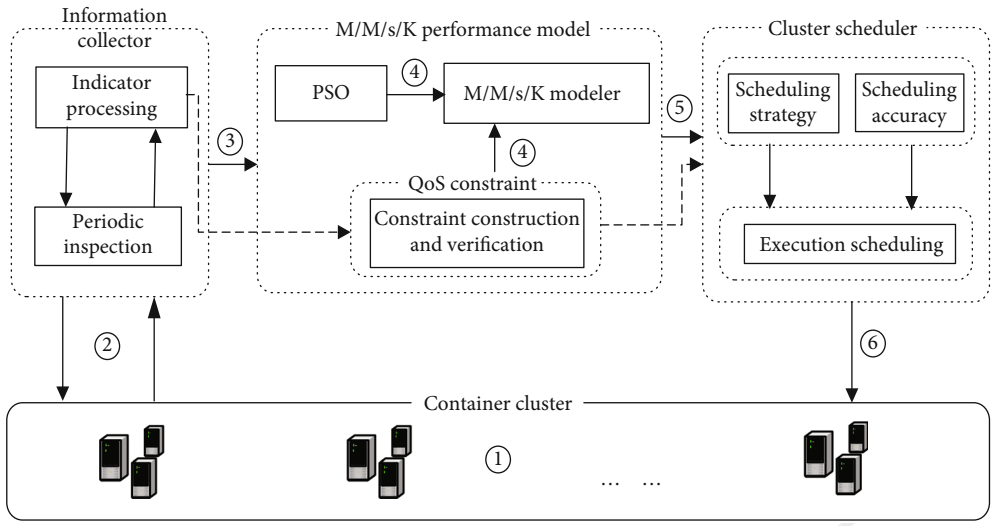

Figure 1: ACEA framework.

fuzzy system is famous for the good balance between approximation accuracy and interpretability. [32] uses the fuzzy system for data preprocessing to improve the accuracy of the algorithm. In this paper, whether we can use the above ideas for reference to classify and deal with tasks is found.

In summary, the existing container cluster dynamic scaling algorithm mainly provide resources for tasks under the premise of ensuring the minimum task execution time [18–20], without considering quantitatively the relationship among the sudden change of the number of tasks, average waiting time of tasks, and comprehensive resource utilization rate of cluster in the Internet environment.

3. Design of ACEA Algorithm

The overall design is shown in Figure 1. The algorithm ACEA consists of three modules: the information collector, M/M/s/K performance model, and cluster scheduler. The information collector is used to obtain the status of the current cluster and provide input data for M/M/s/K; the M/M/s/K performance model is the core of ACEA, which can be divided into three functional components: the QoS constraint verifier, M/M/s/K modeler, and PSO. The module mainly completes the construction of the cluster performance model, evaluation function and QoS constraints, verification of QoS constraints, and dynamic optimization and provides input data for the cluster scheduler. The cluster scheduler completes the specific cluster scheduling function according to the optimal number of containers, and the cluster state provided by the M/M/s/K performance model. The three modules of the ACEA algorithm are executed in sequence and form a closed loop.

3.1. The Process of ACEA

(1) The container cluster is responsible for receiving and processing tasks, which is the processing object of elastic scaling of the algorithm

(2) The information collector is responsible for obtaining the current status of the cluster, such as CPU resources R_{usedCPU}, memory resources R_{usedMEN}, IO resources R_{usedIO}, network resources R_{usedNET}, and number of tasks n, etc. used by each container

(3) After the M/M/s/K performance model obtains obtaining the data of step (2), firstly, the M/M/s/K modeler is used to quantitatively describe the relationship among the various indicators using the hybrid multiserver queuing model M/M/s/K and to build the performance model for the container cluster and the evaluation function which is also the fitness function of the PSO (see Section 3.2.1 for details)

(4) Firstly, the QoS constraint verifier is used to construct QoS constraints. Then, under the constraints of the verifier, the cluster evaluation function in step (3) is used as the fitness function to solve the dynamic optimization problem and obtain the optimal number of containers in the current cluster and achieve the goal of dynamic optimization of the performance of the cluster (see Section 3.2.2 and Section 3.2.3 for details)

(5) After the cluster scheduler obtains the optimal number of containers output by the M/M/s/K performance model module, the cluster scheduling strategy is determined according to the optimal numbers of containers and the state of the cluster.

(6) Complete the cluster scheduling. After the scheduling is completed, go to step (2) to continue execution and form a closed loop.

3.2. M/M/s/K Performance Model. The M/M/s/K performance model is the core of ACEA. It consists of three functional components: the QoS constraint verifier, M/M/s/K modeler, and PSO. This section mainly discusses the components of the M/M/s/K performance model according to the overall design of the ACEA algorithm.

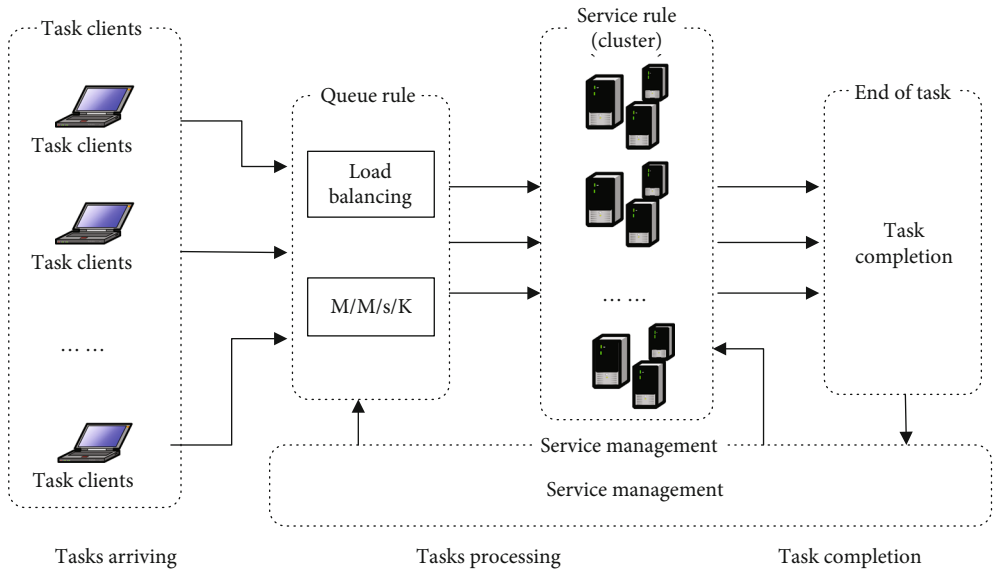

Figure 2: The model of the M/M/s/K modeler.

3.2.1. M/M/s/K Modeler. As shown in Figure 2, the ACEA algorithm compares tasks and containers providing services to customers and servers, respectively. When the container cluster processing capability cannot meet the QoS constraints, the number of containers in the cluster can be dynamically adjusted according to the number of tasks to enhance the service processing capability for the purpose of a flexible supply of resources.

The M/M/s/K modeler uses the principle of Figure 2 to quantitatively describe the relationships among the number of tasks, the average waiting time of tasks, and the comprehensive resource utilization rate of cluster using the hybrid multiserver queuing model M/M/s/K, so as to solve the construction problem of cluster performance model and evaluation function. The evaluation function is used as the fitness function of dynamic optimization. The reasons for choosing the queuing model M/M/s/K are as follows [33]:

(1) The arrival of tasks and the processing time have relatively stable frequencies, while the task has discreteness and independence, which satisfies the condition of exponential distribution

(2) There is an upper limit for the tasks that the servers can handle, which is consistent with the concept of "system space" in the queuing model. The upper limit is defined as K

(3) When the number of tasks to be processed in the servers reach K, the newly arrived task cannot be effectively processed in accordance with the requirements of the quality standard, which results in the loss of tasks. This is consistent with the principle of "when K locations have been occupied by customers, the newly arrived customers leave automatically" in the queuing model

(4) When the number of tasks to be processed in the system is lower than K, the newly arrived task enters the queue and waits for the service, and the principle of "newly arrived customers enter the system to wait in line when the system has a free position" is consistent with the queuing model

3.2.2. QoS Constraint Verifier. The QoS constraint verifier mainly includes two functions: one is to construct QoS constraints and the other is to determine the solution space of the algorithm according to the QoS constraints. QoS constraints define the constraints among number of tasks, average waiting time, comprehensive resource utilization rate of cluster, and the cluster running indicators, which is the basis of constructing the QoS constraints verifier. They specifically include the following contents:

(1) Maximum number of running containers: the maximum number of containers that the cluster hardware resources can support is denoted by K. In the production environment, the number of containers running in the cluster should be less than the maximum number of running containers. Otherwise, the container cannot be started due to insufficient hardware resources of the cluster, so there is $s < K$

(2) Average waiting time of tasks: the mathematical expectation of the maximum waiting time that the user can withstand from the time when the task is issued to the time when the cluster starts responding. If waiting time exceeds the average waiting time, the task will be lost. This paper assumes that the average waiting time of tasks does not exceed 50 ms, that is, $W_q < 50$ ms

(3) Queue length n: the number of tasks in the current queue. If $s > n$, which means the number of

containers is greater than the number of tasks, then there are free containers, resulting in waste of cluster resources. If $n > K$, which means the cluster is overloaded with tasks, it cannot process additional tasks effectively, resulting in the loss of these tasks; therefore, there must be $s < n \leq K$

(4) Threshold constraint

Define $f_{\text{used}\,i}$ as the weighted sum of resources consumed by the ith container in the presence of a task, and the following relationship exists.

$$f_{\text{used}\,i} = aR^i_{\text{usedCPU}} + bR^i_{\text{usedMEN}} + c\text{R}^i_{\text{usedIO}} + dR^i{}_{\text{usedNET}}. \quad (1)$$

Define $f_{\text{total}\,i}$ as the weighted sum of the resources assigned to the ith container by the system, and the following relationship exists.

$$f_{\text{total}\,i} = aR^i{}_{\text{totalCPU}} + bR^i_{\text{totalMEM}} + cR^i{}_{\text{totalIO}} + dR^i{}_{\text{totalNET}}. \quad (2)$$

$R^i{}_{\text{usedCPU}}$ represents the CPU resource used by the ith container, and $R^i{}_{\text{totalCPU}}$ indicates the total CPU resources allocated by the system for the ith container. Other resources are similar. a, b, c, and d are the weights of each resource in the total resources and are determined by the task attributes processed by the container, satisfying the relationship of $a + b + c + d = 1$; this paper assumes $a = b = c = d = 0.25$.

Define U_i as the comprehensive resource utilization rate of a single container, derived from Equation (1), (2).

$$U_i = \frac{f_{\text{used}\,i}\left(R^i_{\text{usedCPU}}, R^i_{\text{usedMEN}}, R^i_{\text{usedIO}}, R^i{}_{\text{usedNET}}\right)}{f_{\text{total}\,i}\left(R^i_{\text{totalCPU}}, R^i_{\text{totalMEM}}, R^i_{\text{totalIO}}, R^i_{\text{totalNET}}\right)}. \quad (3)$$

U_{down} is defined as the lower limit of the comprehensive resource utilization rate of cluster, which means that the comprehensive resource utilization rate of cluster is the lowest. If the utilization rate is lower than this value, it needs to shrink. U_{up} is defined as the upper limit of the comprehensive resource utilization rate of cluster, which means that the comprehensive resource utilization rate of cluster is the highest. If it is higher than this value, it needs to be scaling. According to the requirements for the comprehensive utilization of the cluster, the following constraint is obtained from Equation (3).

$$U_{\text{down}} < \sum_{i=1}^{s} \frac{U_i}{s} * 100\% < U_{\text{up}}. \quad (4)$$

3.2.3. Particle Swarm Optimization. Particle swarm optimization (PSO) algorithm makes use of an individual's sharing of information in the swarm, so that the swarm can evolve from disorder to order in the solution space to obtain the optimal solution. Due to its simple operation and fast convergence, PSO has been widely used in many fields such as function optimization, image processing, and geodetic survey [24]. The reasons for choosing PSO are as follows:

(1) PSO has many mature applications in function optimization

(2) PSO has a fast convergence rate and meets the timeliness requirements of the algorithm

(3) PSO is easy to operate for improving computational efficiency

In the dynamic optimization solution, PSO uses the cluster evaluation function built by the M/M/s/K modeler in the solution space determined by the QoS constraint verifier to search for the optimal solution for the fitness function and obtain the obtained solution. The optimal solution is provided as input data to the cluster scheduler.

4. ACEA Performance Model and Scheduling Strategy Design

This section focuses on the design of the M/M/s/K performance model based on the overall design of ACEA and implements the processing flow and scaling conditions of each module through pseudocode.

4.1. M/M/s/K Performance Model. According to the existing cluster state and task attributes, ACEA firstly obtains the mathematical distribution parameters of task arrival and processing time by using statistical principle, then passes the obtained parameters into the M/M/s/K performance model, and finally solves the optimal expansion strategy under the constraints of QoS. The task processing flow is shown in Figure 3.

L_q is defined as the task average queue length: the mathematical expectation of the number of tasks to be processed in the queued model. According to the Equations (1) and (3) constraints in Section 3.2.2, the following relationships exist:

$$L_q = \sum_{n=1}^{K} (n-s)p_n = \frac{p_0\rho^s\rho_s}{s!(1-\rho_s)^2}\left[1-\rho_s^{K-s+1} - (1-\rho_s)(K-s+1)\rho_s^{K-s}\right], \quad (5)$$

where $\rho = \lambda/\mu$, $\rho_s = \lambda/s\mu$ indicating service intensity, reflecting the busy degree of the system.

$$p_0 = \left(\sum_{n=0}^{s-1}\frac{\rho^n}{n!} + \frac{\rho^s\left(1-\rho_s^{K-s+1}\right)}{s!(1-\rho_s)}\right)^{-1}, \quad (6)$$

$$p_n = \frac{\rho^n}{s!s^{n-s}}p_0. \quad (7)$$

From Equations (5)–(7), W_q can be obtained.

$$W_q = \frac{L_q}{\lambda_e}. \quad (8)$$

$\lambda_e = \lambda(1-p_k)$ indicates the effective arrival rate of tasks. The reason for the effective arrival rate of the task request is

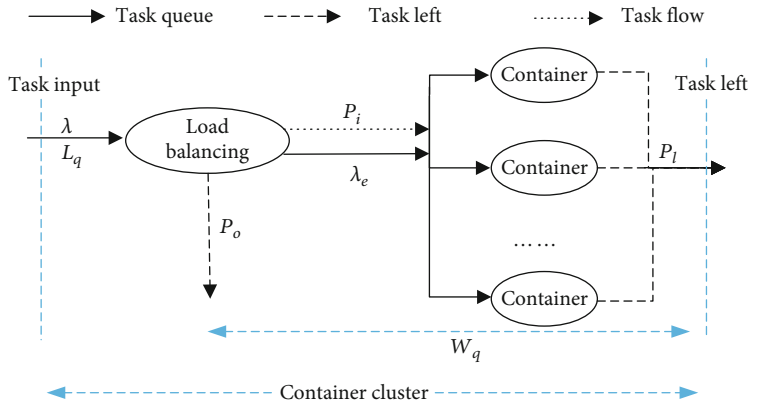

Figure 3: ACEA task processing flow.

that the part of the task failed to process properly during the cluster service. The reason for this is that in the process of cluster service, tasks with probability P_o cannot be handled properly, and tasks with probability P_i can be handled properly. Therefore, the arrangement Equation (8) can be obtained.

$$W_q = \frac{L_q}{\lambda_e} = \frac{\sum_{n=1}^{K}(n-s)p_n}{\lambda P_i} = \frac{\sum_{n=1}^{K}(n-s)p_n}{\lambda(1-P_o)}. \quad (9)$$

Because of the existence of QoS constraints, Equation (9) must satisfy the $W_q < W_{q\,\max}$, that is, there are the following relationships:

$$\mathrm{W_q} = \frac{L_q}{\lambda_e} = \frac{\sum_{n=1}^{K}(n-s)p_n}{\lambda P_i} = \frac{\sum_{n=1}^{K}(n-s)p_n}{\lambda(1-P_o)} < \mathrm{W_{qmax}}, \quad (10)$$

$$E\left[\mathrm{W_q} \mid s=s\right] < \mathrm{W_{qmax}} \text{ and } E\left[\mathrm{W_q} \mid s=s-1\right] \geq \mathrm{W_{qmax}}, \quad (11)$$

$$\mathrm{U_{down}} < E\left[\sum_{i=1}^{s}\frac{\mathrm{U}_i}{s} \mid s=s\right] < \mathrm{U_{up}}, \quad (12)$$

$$\mathrm{U_{down}} > E\left[\sum_{i=1}^{s}\frac{\mathrm{U}_i}{s} \mid s=s+1\right] \text{ or } E\left[\sum_{i=1}^{s}\frac{\mathrm{U}_i}{s} \mid s=s-1\right] > \mathrm{U_{up}}. \quad (13)$$

Equations (10)–(13) show that W_q is only related with s, K, λ, and μ, and since λ, μ, and K are relatively independent, s that conforms to the QoS constraints Section 3.2.2 can be regarded as a set of feasible solutions, and all the set of feasible solutions is the solution space. Therefore, the problem of obtaining the optimal index of cluster turns into the problem of obtaining the best feasible solution. For this reason, PSO is introduced to search for the optimal solution in the solution space to achieve the goal of dynamic optimization.

4.2. Adaptive Scaling Strategy Design. This section implements the overall design and cluster scheduling strategy of ACEA in pseudocode. The pseudocode is shown in Algorithm 1. Line 1 defines the model parameters of the algorithm as global variables, including the distribution function parameters, the number of particles, the maximum number of iterations, and the adaptation degree of the task arrival and processing in the queuing model M/M/s/K. Line 2 defines the threshold of the system's comprehensive resource utilization rate and average waiting time of tasks. Line 3 defines the configuration file function of information collection service, which is used to obtain the cluster's state parameters, including CPU usage, memory usage, IO resources, network resources, etc. Lines 4-5 construct the particle fitness function (performance model evaluation function) and particle initialization and particle swarm algorithm according to the parameters of Line 1. Lines 6-12 particles search the feasible solution space for dynamic optimization. Lines 14-16 define the condition for scaling, that is, if the average waiting time of tasks satisfies the requirement, and the fitness value is lower than the lower threshold, contraction() function is performed. Lines 17-19 define the condition for scaling, that is, if the average waiting time of tasks satisfies the requirement and the fitness value is higher than the upper threshold, expand() function is performed. Lines 20-22 defines the condition for stability, that is, if the average waiting time of tasks satisfies the requirement, and the fitness value is between the upper and lower thresholds, scheduling is not performed, and only the current results will be visually managed. There are three main scheduling strategies for Algorithm 1.

(1) Container Contraction. The main reason for the container contraction is that the number of tasks is reduced or the task processing is completed, causing the decrease of various monitoring indicators to different extents. The decrease of $\sum f_{\mathrm{used}\,i}$ will make the resource utilization rate lower than the MIN threshold. In this case, the container cluster needs to be reduced

(2) Container Expand. The main reason for the expansion of the container is that the number of tasks increases, causing the increase the monitoring indicators to different extents. The increase of $\sum f_{\mathrm{used}\,i}$ will make the resource utilization rate higher than the upper threshold or the average waiting time of tasks failing to meet the QoS constraints. In this case, the container cluster needs to be expanded

Input: achievement rate of historical tasks $\lambda_{History}$, task processing interval μ, number of containers in the current cluster s.
Output: indicator information for container cluster, volume of autoscaling, mode of adaptive scaling.
1. global var W_q, Fitness; //Fitness: real-time comprehensive resource utilization rate of cluster.
2. const var U_{up}, U_{down}, $W_{limit}U_{down}$;// W_{limit}: maximum waiting time of tasks.
3. define func getConfig(): L_{CPU}, L_{MEM}, L_{IO}, L_{NET}, L_{LIMIT};
4. Input
5. par[Pnum]=
6. PSOInit(Pnum): ParticleInit(Pnum), ParticleEvaluate();
7. Fitness = Wq(getUall(s), getUlimit());
8. PSORun():
9. For each par.
10. ParticleUpdate();
11. ParticleEvaluate();
12. End For;
13. For Iteration times do.
14. If ($W_q < W_{limit}$ &&U_{down} > Fitness):
15. Contract(service);
16. End if;
17. If(($W_q < W_{limit}$ &&Fitness > U_{up})|| $W_q < W_{limit}$):
18. Expand(service);
19. End if;
20. If($W_q < W_{limit}$ &&U_{down} < Fitness < U_{up}):
21. PSOshowresult();
22. End if;
23. End For;

Algorithm 1: The algorithm of ACEA.

(3) Stable State. The main reason for the stability of the cluster is that the number of tasks is stable and there is no sudden change, so that the average waiting time of tasks and resource utilization rate are in line with the custom QoS constraints

5. Results and Discussion

The algorithm ACEA has been prototyped in the Docker virtualized cluster. This section will verify the effectiveness of the algorithm in the case of a sudden change in the number of tasks and compare with the existing elastic scaling algorithm to verify the accuracy of the system's model. The objects for comparison are two general algorithms in the field of elastic scaling.

The incremental scheduling algorithm (ISA) is an algorithm that periodically checks the state of the cluster through the polling service. When the task average waiting time or the comprehensive resource utilization rate of cluster does not meet the QoS constraints, the cluster scheduler or operation and maintenance personnel, based on historical experience, will quantitatively determine the number of containers to be adjusted in a certain interval to cope with the current task. The quantitative determination of the number of containers required to be adjusted within a certain interval is an increment.

Kubernetes HPA [29] (Kubernetes horizontal Pod autoscaling, Kubernetes Pod) obtains information about resource usage by periodically polling the Pod state during the operation of the container cluster and then compares the average usage rate of the existing Pod with the target usage rate to determine the number of Pod copies, and finally through the horizontal dynamic adjustment of the number of Pod copies to achieve the purpose of scaling. The formula for calculating the elastic scaling of Pod is as follows.

$$\text{ExpansionPods} = \frac{\text{Ceil}(\text{Sum}(\text{CurrentUtilization}))}{\text{TargetUtilization}}. \quad (14)$$

Among them, ExpansionPods indicates the number of containers required; Target Utilization indicates the user-defined resource usage threshold; CurrentUtilization indicates the average resource utilization of the current Pod, and the calculation formula is as shown in Equation (15); Sum() is a summation function for the sum of current utilization; Ceil() is the integer function used to return the smallest integer greater than or equal to the specified expression.

$$\text{CurrentUti lization} = \frac{\text{Average value of used resources}}{\text{Resources allocated to Pod by the system}}. \quad (15)$$

5.1. Experimental Threshold. The experimental threshold settings in this section are as follows:

(1) The upper threshold of the average waiting time of tasks is set to 50 ms, that is $W_q \leq 50$ ms

(2) The upper threshold of the comprehensive resource utilization rate of cluster is 80%, and the lower

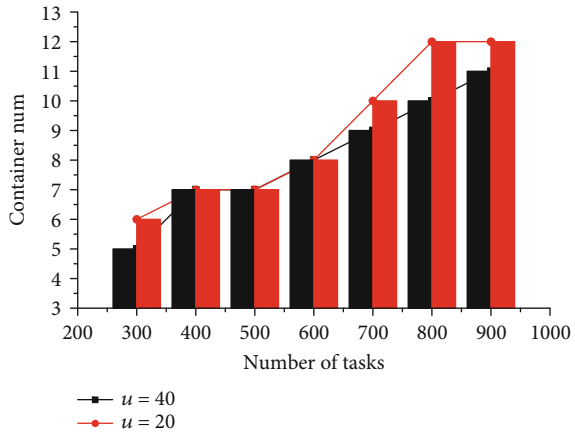

Figure 4: Adaptive figure of the number of containers.

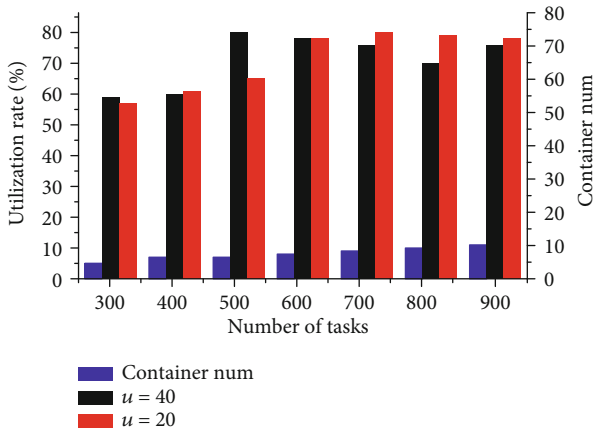

Figure 5: Changing figure of the comprehensive resource utilization rate.

threshold is 55%. Equation (4) gives the relationship of cluster resource utilization as shown in Equation (16)

$$55\% < \sum_{i=1}^{s} \frac{f_{\text{used}\,i}\left(R^i_{\text{usedCPU}}, R^i_{\text{usedMEN}}, R^i_{\text{usedIO}}, R^i_{\text{usedNET}}\right)}{f_{\text{total}\,i}\left(R^i_{\text{totalCPU}}, R^i_{\text{totalMEM}}, R^i_{\text{totalIO}}, R^i_{\text{totalNET}}\right)} * 100 < 80\%. \quad (16)$$

5.2. Experimental Analysis. The purpose of this experiment is to verify the coordination and effectiveness of the information collector, the M/M/s/K performance model, cluster scheduler in ACEA, and the feasibility of ACEA's scaling strategy. The experimental results are shown in Figures 4–6.

As shown in Figure 4, the number of containers in the cluster changes simultaneously when the number of tasks changes, and the trend of change is consistent with the number of tasks, which verifies the feasibility of container cluster scheduling strategy. The following validation experiments are based on the experimental data to verify the performance of the average waiting time of tasks and the comprehensive resource utilization rate of cluster.

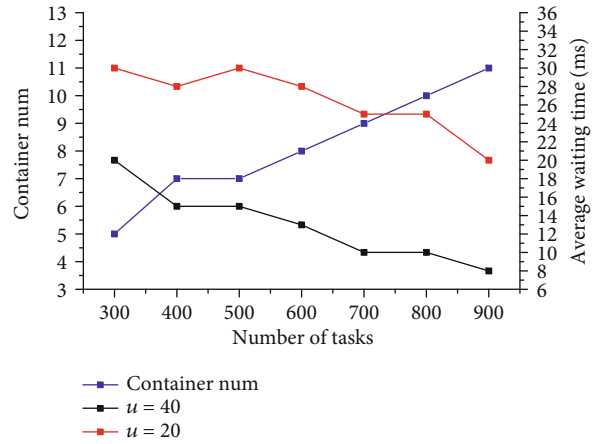

Figure 6: Changing figure of the average waiting time of tasks.

As shown in Figure 5, when the number of tasks changes, the number of containers in the cluster and the comprehensive resource utilization rate of cluster change accordingly. Corresponding to the left axis of the figure above, the number of tasks and the number of containers have the same trend. Corresponding to the right axis of the figure above, although the number of tasks that the cluster can handle per unit time, i.e., the average service rate μ, is different, the comprehensive utilization rate of cluster resources calculated by Equations (3) and (4) is always between 55% and 80%, which meets the threshold requirement of the comprehensive resource utilization rate of cluster.

As shown in Figure 6, when the number of tasks changes, the number of containers in the cluster and the average waiting time of tasks change accordingly. Corresponding to the left axis of the figure above, the number of tasks and the number of containers have the same trend. Corresponding to the right axis of the figure above, although the number of tasks that the cluster can handle per unit time, i.e., the average service rate μ, is different, the average waiting time of tasks increases synchronously and is always lower than 35 ms, which satisfies the threshold requirement of average waiting time of tasks and does not affect the effective processing of tasks.

It can be seen from the above experimental results that the average waiting time of tasks is always within 35 ms, and the comprehensive resource utilization rate of cluster is always maintained between 55% and 80%, which satisfies the requirement for the threshold set by the user, indicating that the algorithm satisfies the average waiting time of tasks. In the case of time requirements, the comprehensive resource utilization rate of cluster is also guaranteed.

5.3. Comparison with ISA Algorithm. The biggest feature of this algorithm is easy to implement, no need for plugins, and easy operation and maintenance personnel. Compared with the algorithm ACEA, the performance is shown in Figures 7 and 8.

As shown in Figure 7, with the running of the system, when the number of tasks changes, the number of containers in the cluster and the average waiting time of tasks change accordingly. Corresponding to the left axis of the figure

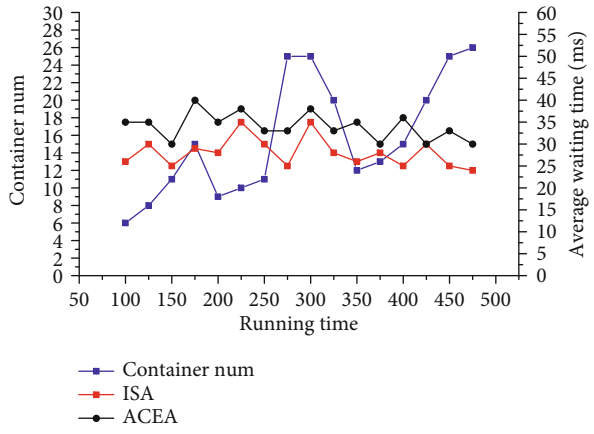

Figure 7: Comparison of average waiting time of tasks.

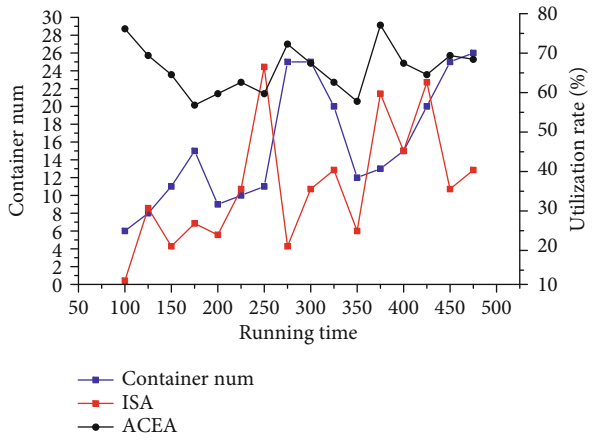

Figure 8: Comparison of the comprehensive resource utilization rate.

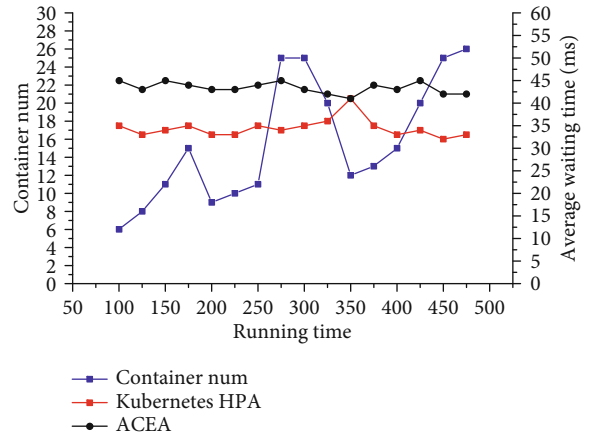

Figure 9: Comparison of average waiting time of tasks.

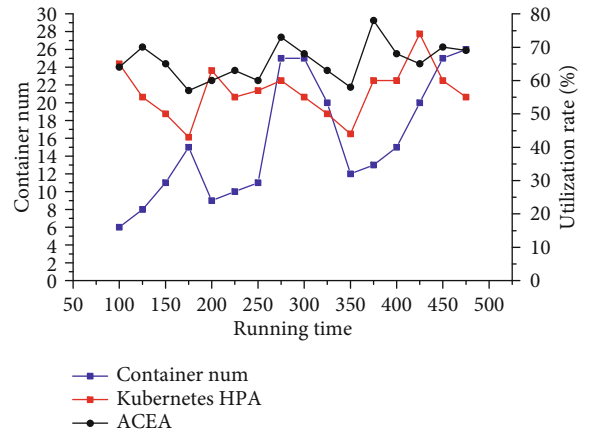

Figure 10: Comparison of the comprehensive resource utilization rate.

above, with the system running, in order to meet the threshold requirements, the number of containers in the cluster changes when the number of tasks changes. Corresponding to the right axis of the figure above, the average waiting time of ACEA algorithm is higher than that of ISA algorithm, and the difference is less than 12 ms. Meanwhile, although the average waiting time of ACEA algorithm fluctuates, it is always lower than 45 ms, which satisfies the threshold requirement of average waiting time of tasks and does not affect the effective processing of tasks.

As shown in Figure 8, with the running of the system, when the number of tasks changes randomly, the comprehensive resource utilization rate and the number of containers in the cluster will change accordingly. For effective comparison of the performance of the algorithm, the change rule of the number of tasks here is consistent with Figure 7. Corresponding to the left axis of the figure above, with the running of the system, in order to meet the threshold requirements, the number of containers in the cluster changes when the number of tasks changes. Corresponding to the right axis of the figure above, when the running time is 250, the comprehensive resource utilization rate of ISA algorithm is higher than that of ACEA algorithm. The reason is that the number of capacity expansion determined by experience, i.e. the increment, matches the number of tasks at present, but overall, the comprehensive resource utilization rate of ACEA algorithm is higher than that of ISA algorithm, and the comprehensive resource utilization rate of ACEA algorithm is always between 55% and 80%, which achieves the goal of ACEA algorithm to improve the comprehensive resource utilization rate on the basis of ensuring the average waiting time of tasks.

5.4. Comparison with Kubernetes HPA. From the perspective of theoretical analysis, Kubernetes HPA does not provide the function of automatically scaling the Pod according to the constraint of average waiting time of tasks. ACEA makes up for this part of the blank and ensures the balance between the average waiting time of tasks and the comprehensive resource utilization rate of cluster by using the dynamic optimization algorithm. From an experimental point of view, the performance is shown in Figures 9 and 10 compared to ACEA.

As shown in Figure 9, with the running of the system, when the number of tasks changes, the number of containers in the cluster and the average waiting time of tasks change accordingly. Corresponding to the left axis of the figure

above, with the system running, in order to meet the threshold requirements, the number of containers in the cluster changes when the number of tasks changes. Corresponding to the right axis of the figure above, the average waiting time of ACEA algorithm is higher than that of Kubernetes HPA, and the difference is less than 15 ms. Meanwhile, although the average waiting time of ACEA algorithm fluctuates, it is always lower than 45 ms, which satisfies the threshold requirement of average waiting time of tasks and does not affect the effective processing of tasks.

As shown in Figure 10, with the running of the system, when the number of tasks changes randomly, the comprehensive resource utilization rate and the number of containers in the cluster will change accordingly. For effective comparison of the performance of the algorithm, the change rule of the number of tasks here is consistent with Figure 9. Corresponding to the left axis of the figure above, with the running of the system, in order to meet the threshold requirements, the number of containers in the cluster changes when the number of tasks changes. Corresponding to the right axis of the figure above, when the running time is 200 and 425, the comprehensive resource utilization rate of Kubernetes HPA is higher than that of ACEA algorithm. The reason is due to the influence of Kubernetes HPA resource monitoring transmission delay and Equation (10) average strategy, but overall, the comprehensive resource utilization rate of ACEA algorithm is higher than that of Kubernetes HPA, and the comprehensive resource utilization rate of ACEA algorithm is always between 55% and 80%, which achieves the goal of ACEA algorithm to improve the comprehensive resource utilization rate on the basis of ensuring the average waiting time of tasks.

6. Conclusions

This paper proposes the ACEA algorithm for balancing the average waiting time of tasks and the comprehensive resource utilization rate of cluster in elastic scaling. This algorithm uses a hybrid multiserver queuing model M/M/s/K to build a container cluster performance model and evaluate functions and QoS constraints and uses a QoS constraint validator to determine the feasible solution space of the algorithm, and the feasible solution space is searched by PSO to achieve dynamic optimization of the algorithm. The experimental results show that the proposed algorithm can ensure the comprehensive resource utilization rate of cluster while guaranteeing that the average waiting time of tasks is satisfied. However, we believe that there is still some follow-up work worth extending, including the following:

(1) For the characteristics of sudden changes of the number of tasks, it may be considered to introduce a multiple QoS authentication strategy or a smoothing algorithm to avoid invalid scaling and reduce the jitter of cluster scaling

(2) Try to apply ACEA to the microservice architecture container cluster

(3) At present, there are some improved clustering algorithms. We can try to cluster tasks on the basis of the improved clustering algorithm and use the task category attribute to improve the efficiency of task forwarding in load balancing

Data Availability

The data used to support the findings of this study are included in the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Kui Li did the conceptualization, methodology, writing—original draft, and writing—review and editing. Yi-mu Ji did the funding acquisition and project administration. Shang-dong Liu found resources and did the validation. Hai-chang Yao did the visualization. Hang Li and Shuai You acquired the software. Si-si Shao did the data curation and investigation.

Acknowledgments

This work was supported by the National Key R&D Program of China (2020YFB2104000, 2020YFB2104002), Natural Science Foundation of Jiangsu Province (Higher Education Institutions) (BK20170900, 19KJB520046, and 20KJA520001), Innovative and Entrepreneurial talents projects of Jiangsu Province, Jiangsu Planned Projects for Postdoctoral Research Funds (No. 2019K024), Six talent peak projects in Jiangsu Province (JY02), Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX19_0921, KYCX19_0906), Zhejiang Lab (2021KF0AB05), and NUPT DingShan Scholar Project and NUPTSF (NY219132).

References

[1] L. Wang, Y. Zhang, M. Jiang et al., "Toward a normalized clinical drug knowledge base in China - applying the RxNorm model to Chinese clinical drugs," *Journal of the American Medical Informatics Association*, vol. 25, no. 7, pp. 809–818, 2018.

[2] Z. H. Zhan, X. F. Liu, Y. J. Gong, J. Zhang, H. S. H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–33, 2015.

[3] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: a survey," *Journal of Cloud Computing*, vol. 5, no. 7, article 7, 2016.

[4] J. V. B. Bibal and D. Dejey, "An auto-scaling framework for heterogeneous Hadoop systems," *International Journal of Cooperative Information Systems*, vol. 26, no. 4, article 1750004, 2017.

[5] M. Abdullah, K. Lu, P. Wieder, and R. Yahyapour, "A heuristic-based approach for dynamic VMs consolidation in cloud data centers," *Arabian Journal for Science & Engineering*, vol. 42, no. 8, pp. 3535–3549, 2017.

[6] Y. T. Lin, M. L. Wen, M. Jou, and D. W. Wu, "A cloud-based learning environment for developing student reflection abilities," *Computers in Human Behavior*, vol. 32, pp. 244–252, 2014.

[7] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[8] "Docker—build, ship, and run any app, anywhere," January 2019, http://www.docker.com.

[9] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 239, 2014.

[10] F. Paraiso, S. Challita, Y. Al-Dhuraibi, and P. Merle, "Model-driven management of docker containers," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 718–725, San Francisco, CA, USA, 2016.

[11] Q. Wu, "Making Facebook's software infrastructure more energy efficient with autoscale," *Facebook Engineering Blog*, 2014, https://engineering.fb.com/2014/08/08/production-engineering/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale.

[12] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, pp. 342–346, Ghaziabad, India, 2015.

[13] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 955–962, Taipei, Taiwan, 2017.

[14] J. A. Aroca, A. F. Anta, M. A. Mosteiro, C. Thraves, and L. Wang, "Power-efficient assignment of virtual machines to physical machines," *Future Generation Computer Systems*, vol. 54, pp. 82–94, 2016.

[15] P. R. Desai, "A survey of performance comparison between virtual machines and containers," *International Journal of Computer Sciences and Engineering*, vol. 4, no. 7, pp. 55–59, 2016.

[16] Y. Kouki and T. Ledoux, "SCAling: SLA-driven cloud auto-scaling," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*, pp. 411–414, Coimbra Portugal, 2013.

[17] C. Liu, B. T. Loo, and Y. Mao, "Declarative automated cloud resource orchestration," in *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*, Cascais, Portugal, 2011.

[18] B. Keshanchi and N. J. Navimipour, "Priority-based task scheduling in the cloud systems using a memetic algorithm," *Journal of Circuits, Systems and Computers*, vol. 25, no. 10, article 1650119, 2016.

[19] F. Ramezani, J. Lu, and F. Hussain, "Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization," in *Service-Oriented Computing. ICSOC 2013*, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds., vol. 8274 of Lecture Notes in Computer Science, pp. 237–251, Springer, Berlin, Heidelberg, 2013.

[20] S. Dhakate and A. Godbole, "Distributed cloud monitoring using Docker as next generation container virtualization technology," in *2015 Annual IEEE India Conference (INDICON)*, pp. 1–5, New Delhi, India, 2015.

[21] D. Wang, S. Zhu, T. Li, and Y. H. Gong, "Comparative document summarization via discriminative sentence selection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 3, pp. 1–18, 2012.

[22] K. Lerman and R. McDonald, "Contrastive summarization: an experiment with consumer reviews," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers on - NAACL '09*, pp. 113–116, Boulder, Colorado, 2009.

[23] X. Wan, H. Jia, S. Huang, and J. G. Xiao, "Summarizing the differences in multilingual news," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11*, pp. 735–744, Beijing, China, 2011.

[24] M. Abdullah, E. A. Al-Muta'a, and M. A. Sanabani, "Integrated MOPSO algorithms for task scheduling in cloud computing," *Journal of Intelligent & Fuzzy Systems*, vol. 36, no. 2, pp. 1823–1836, 2019.

[25] N. Naik, "Applying computational intelligence for enhancing the dependability of multi-cloud systems using docker swarm," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–7, Athens, Greece, 2016.

[26] N. Naik, "Building a virtual system of systems using Docker Swarm in multiple clouds," in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–3, Edinburgh, UK, 2016.

[27] J. W. Xu, W. B. Zhang, T. Wang, and T. Huang, "A genetic algorithm based adaptive strategy for image backup of virtual machines," *Chinese Journal of Computers*, vol. 39, no. 2, pp. 351–362, 2016.

[28] M. Agarwal and G. M. S. Srivastava, "A genetic algorithm inspired task scheduling in cloud computing," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 364–367, Greater Noida, India, 2016.

[29] "Kubernetes plugin," January 2019, http://docs.getcloudify.org/4.1.0/plugins/kubernetes.

[30] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing*, vol. 5, article 4, 2016.

[31] L. Nikolaos, "D1.1 requirements analysis report. Cloud 4SOA project deliverable," January 2019, https://pdfs.semanticscholar.org/20fb/57b26982a404138a32ff756e73d26c29a6f2.pdf.

[32] Y. Zhang, F. L. Chung, and S. Wang, "Takagi-Sugeno-Kang fuzzy systems with dynamic rule weights," *Journal of Intelligent & Fuzzy Systems*, vol. 37, no. 6, pp. 8535–8550, 2019.

[33] C. de Alfonso, A. Calatrava, and G. Moltó, "Container-based virtual elastic clusters," *Journal of Systems and Software*, vol. 127, pp. 1–11, 2017.