An Energy-Efficient Scheduling Method for Real-Time Multi-workflow in Container Cloud

Zaixing Sun, Zhikai Li, Chonglin Gu(✉), and Hejiao Huang

School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China
guchonglin@hit.edu.cn

Abstract. Cloud computing has a powerful ability to handle a large number of tasks. Correspondingly, it also consumes a lot of energy. Reducing the energy consumption of cloud service platforms while ensuring the quality of service has become a crucial issue. In this paper, we propose a heuristic energy-saving scheduling algorithm named Real-time Multi-workflow Energy-efficient Scheduling (RMES) with the aim to minimize the total energy consumption in container cloud. RMES executes tasks as parallel as possible to enhance the resource utilization of the running machines in cluster, therefore reducing the time of the global process, saving energy as a result. RMES takes advantage of the affinity between containers and machines to meet the resource quantity and performance requirements of containers during scheduling. In order to follow the change of the system state overtime, we introduce the re-scheduling mechanism, which can automatically adjust the scheduling decisions of the tasks that have not yet been executed in the scheduling scheme. The experimental results show that RMES has obvious advantages over other scheduling algorithms in terms of energy consumption and success ratio.

Keywords: Multi-workflow scheduling · Real time · Container cloud · Energy minimization

1 Introduction

Cloud service platform (CSP) have powerful ability to handle large-scale scientific applications. These applications are submitted to CSP in real time in the form of workflow. The different users' workflow requests with various structures are mixed into a multi-workflow for CSP to process. Each workflow has its Quality of Service (QoS, such as deadline) needs. Different workflows consist of tasks with various resource requirements. CSP provides consumers with on-demand compute and storage resources [2]. Container, a new virtualization technique, is better suited for this multi-workflow scenario than classic virtualization technology [14]. It has three advantages including less memory, faster startup speed and lower management overhead [17]. In container cloud, users can specify the affinity between containers for applications, which facilitates the container orchestration on clusters [11], such that the special resource requirements of the tasks can be met.

CSP has a large number of physical machines, which consume massive energy. Data centres are reported [6] to spend around $13 billion a year on electricity. Massive energy usage not only increases the expense of the data center, but also causes some damage to the environment. However, the main challenge is that we should not only minimize the energy consumption of cloud service center, but also ensure all the tasks to be completed on time. At present, many researches have focused on the scheduling algorithm to reduce energy consumption in data centers. In [12,15], the authors proposed energy-saving scheduling algorithms on the traditional cloud. Since the complex affinity relationship between containers and physical machines brings more constraints to energy-saving scheduling decisions, these methods can not be directly used in container cloud scenarios. In [8], an energy-saving scheduling algorithm based on Q-Learning is proposed. However, the algorithm does not consider the dependencies between tasks. The workflow scheduling problem is an NP-hard problem [16]. These algorithms require a lot of computation to make decisions, which may not be suitable for real-time scheduling scenarios that require rapid response.

The heuristic method is to set some scheduling rules to make the task scheduling results on the cluster reach an approximately optimal state. Heuristic approaches are faster than other methods at producing scheduling decisions because they are based on empirical design rules. Therefore, this method is more suitable for real-time scheduling scenarios. In [9], principles from generational garbage collection (GC) reduce energy consumption in homogeneous clusters and ensure that all requests do not violate deadline constraints as much as possible. In [10], the authors adjust the task scheduling decision by balancing energy consumption and task execution time in the real-time scenario. However, the above methods either do not consider the real-time constraints, or ignore some special conditions of resource constraints in container cloud, such as their affinity.

In view of the above shortcomings, we propose a real-time multi-workflow energy-efficient scheduling (RMES) algorithm to solve real-time multi-workflow scheduling. The objective is to minimize the energy consumption of the cluster while completing as many workflows on time as possible. The main contributions of this paper are as follows:

- We build a real-time multi-workflow scheduling model on heterogeneous clusters considering the affinity constraints between tasks and machines.
- We propose a heuristic scheduling algorithm called RMES, which decreases the base energy consumption of cluster by compressing the time of the global process through executing tasks in parallel.
- RMES evaluates the current running status of the physical machines in cluster, and shuts down the physical machines with low utilization in time, thus reducing unnecessary energy consumption of the cluster.
- The performance of scheduling algorithm is verified by using workflow in the real world. Compared with the existing algorithms, RMES reduces more energy consumption for CSP while meeting the affinity constrains between container and physical machine.

The rest of this paper is organized as follows. Section 2 introduces the real-time multi-workflow scheduling and energy consumption model. In Sect. 3, a heuristic energy-saving scheduling algorithm is proposed. Section 4 presents the experimentation and evaluation. Finally, Sect. 5 concludes the paper.

2 Problem Formulation

2.1 Workflow Modeling

In cloud, the system needs to schedule the workflow applications submitted dynamically by users in real time. These workflows are composed of many requests which can be denoted as $\mathcal{W} = \{w_1, w_2, ..., w_m\}$. A single request can be described as a directed acyclic graph (DAG). We model a request $w_m \in \mathcal{W}$ as $w_m = \{w_m^{at}, w_m^{trt}, w_m^{d}, G_m\}$, where w_m^{at}, w_m^{trt}, w_m^{d} and G_m represent the arrival time, tolerable running time (the maximum time a user can tolerate for a request to be execute), deadline and structure of w_m, respectively. w_m^d can be calculated as $w_m^d = w_m^{at} + w_m^{trt}$. $G_m = (\mathcal{T}_m, E_m)$, where $\mathcal{T}_m$ is the set of tasks and E_m represents the dependency between tasks. $\mathcal{T}_m = \{t_{m1}, t_{m2}, \cdots, t_{m|\mathcal{T}_m|}\}$, where $T_{mi}(0 < i \leq |\mathcal{T}_m|)$ represents the ith task of w_m and $|\mathcal{T}_m|$ is the total number of tasks contained in w_m. E_m is the 0–1 matrix of $T_m \times T_m$. $e_{u,v}^m = 1$ means that there is a data dependence between t_{mu} and t_{mv}, where t_{mu} is the immediate predecessor of t_{mv}. For the t_{mi}, we further model it as $t_{mi} = \{t_{mi}^I, t_{mi}^{image}, t_{mi}^{type}, t_{mi^d}\}$. t_{mi}^I is the number of instructions contained in t_{mi}; t_{mi}^{image} is the image of container executing t_{mi}; t_{mi}^{type} is the t_{mi}'s type; t_{mi}^d is the sub-deadline of t_{mi}.

A task is executed within a container and then deployed on physical machine (PM). Container bundles the software configuration of a specific workflow into a container image and is the smallest execution unit in the resource scheduling system. We model container c_j as $c_j = \{c_j^{type}, c_j^{cpu}, c_j^{mem}, c_j^{cache}, c_j^{taints}\}$, where $c_j \in \mathcal{C}$ $(0 < j \leq |\mathcal{C}|)$. $\mathcal{C}$ is the set of all containers in the cluster. c_j^{type} is the type of task that container c_j runs; c_j^{cpu} is the number of CPU cores required by container c_j; c_j^{mem} is memory size required by container c_j; c_j^{cache} is the set of tasks that have been assigned to container c_j; c_j^{taints} is taint nodes (the set of PMs that cannot run container c_j).

Since a single container can only process one task at a time, the tasks in the cache can be divided into two types: waiting and executing. The taint node is a set of PMs that cannot run the container. This set is defined by the user. The reasons why a PM cannot run the container include that there are no specific devices required by the container, the performance of some devices cannot meet the minimum requirements for container operation, and so on. There is a one-to-one correspondence between containers and tasks. The same type container can only run the same type task, and tasks of the same type can only run on the same type of container. A container is created based on the images contained in the corresponding task.

2.2 Service Instance Modeling

A cloud service provider can provide a variety of cloud service instances, such as virtual machines and PMs. In this paper, we only consider the case of PM $\mathcal{P} =$

$\{p_1, p_2, \ldots, p_{|\mathcal{P}|}\}$. We use $p_k(0 < k \le |\mathcal{P}|)$ to represent the kth PM and $\mathcal{P}$ is the set of PMs in this CSP. We model a PM as $p_k = \{p_k^{t\_cpu}, p_k^{t\_mem}, p_k^{\bar{t},u\_cpu}, p_k^{\bar{t},u\_mem}, p_k^c, p_k^{e\_total}, p_k^{e\_base}, p_k^{ips}\}$. $p_k^{t\_cpu}$ is the Number of CPU cores of the p_k; $p_k^{t\_mem}$ is the memory resources of the p_k; $p_k^{\bar{t},u\_cpu}$ is the Number of CPU cores used in p_k at $\bar{t}$ moment; $p_k^{\bar{t},u\_mem}$ is the memory resources used in p_k at $\bar{t}$ moment; p_k^c is the container set which contains container runs in p_k; $p_k^{e\_total}$ is the power of full load operation of p_k; $p_k^{e\_base}$ is the basic power of no-load operation of p_k; p_k^{ips} is the number of instructions that a single core of p_k can process per second. $p_k^{\bar{t},u\_cpu}, p_k^{\bar{t},u\_mem}$ can be calculate by Eqs.(1–2), where $x_{j,k}^{\bar{t}} \in \{0, 1\}$, $x_{j,k}^{\bar{t}} = 1$ means that the container c_j is running in p_k at $\bar{t}$ moment.

$$p_k^{\bar{t},u\_cpu} = \sum_{c_j \in \mathcal{C}} x_{j,k}^{\bar{t}} c_j^{cpu}, \tag{1}$$

$$p_k^{\bar{t},u\_mem} = \sum_{c_j \in \mathcal{C}} x_{j,k}^{\bar{t}} c_j^{mem}, \tag{2}$$

Based on the model widely used [4,8] in cloud computing energy analysis, we model the relationship between the power of the p_k and the CPU utilization at $\bar{t}$ moment by Eq. (3), where $\mathbf{P}_k^{\bar{t}}$ represent the power of p_k at $\bar{t}$ moment.

$$\mathbf{P}_k^{\bar{t}} = \begin{cases} 0, & p_k \, is \, off, \\ \frac{p_k^{\bar{t},u\_cpu}}{p_k^{t\_cpu}} (p_k^{e\_total} - p_k^{e\_base}) + p_k^{e\_base}, & p_k \, is \, on. \end{cases} \tag{3}$$

2.3 Workflow Scheduling Model

In this paper, workflow scheduling aims to minimize the total energy consumption for executing workflows. The total energy consumption for a CSP can be expressed by Eq. (4), where T is total running time of the system.

$$\mathbb{E} = \sum_{\bar{t}=0}^{T} \sum_{k=1}^{|\mathcal{P}|} \mathbf{P}_k^{\bar{t}}, \tag{4}$$

Task Dependency Constraints. Due to the dependency between tasks, all tasks can be executed only when their predecessors are completed or there are no predecessors. t_{mu}^{EST} and t_{mv}^{FT} mean earliest start time of t_{mu} and finish time of t_{mv}, respectively. $Pred(t_{mu})$ is a set of immediate predecessors of t_{mu}, where $Pred(t_{mu}) = \{t_{mv} | e_{v,u}^m = 1, \forall \; t_{mv} \in \mathcal{T}_m\}$.

$$\max_{t_{mv} \in Pred(t_{mu})} t_{mv}^{FT} \le t_{mu}^{EST}, \tag{5}$$

Task Completion Time Constraint. When the system makes scheduling decisions, we should ensure that the real-time tasks submitted by users can be completed on time.

$$\max_{t_{mv} \in \mathcal{T}_m} t_{mv}^{FT} \le w_m^d. \tag{6}$$

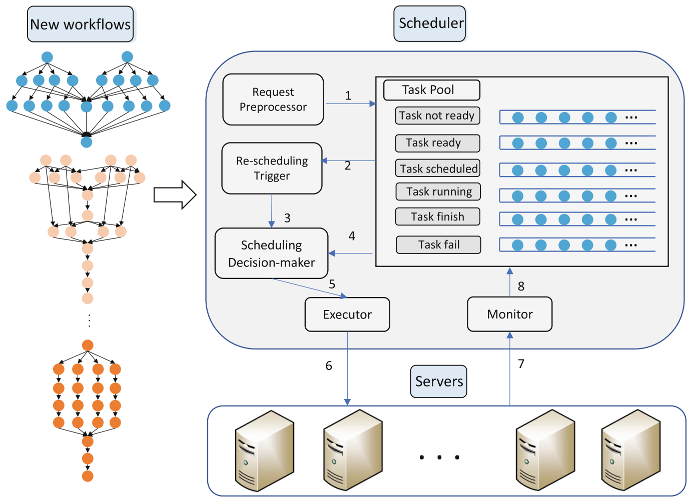

Fig. 1. Schedule System

Task and Container Placement Constraint. A task can only be deployed on a single container, and its type should be the same as the task type that the container can handle. In Eq. (7), $y^{\bar{t}}_{mi,j} \in \{0,1\}$ and $y^{\bar{t}}_{mi,j} = 1$ means t_{mi} is deployed on c_j, otherwise we set $y^{\bar{t}}_{mi,j}$ to 0. When we deploy containers on PMs, we need to meet some resource level constraints. Equation (9) and Eq. (10) mean the resources occupied by the deployed container on the PM cannot exceed the total resources of the PM. Equation (11) means a container can only be deployed on one PM and this PM can't be taint node of c_j.

$$\sum_{j=1}^{|\mathcal{C}|} y^{\bar{t}}_{mi,j} = 1. \tag{7}$$

$$t^{type}_{mi} = c^{type}_j, y^{\bar{t}}_{mi,j} = 1. \tag{8}$$

$$\sum_{c_j \in \mathcal{C}} x^{\bar{t}}_{j,k} c^{cpu}_j \le p^{t\_cpu}_k, \tag{9}$$

$$\sum_{c_j \in \mathcal{C}} x^{\bar{t}}_{j,k} c^{men}_j \le p^{t\_mem}_k, \tag{10}$$

$$\sum_{j=1}^{|\mathcal{C}|} x^{\bar{t}}_{j,k} = 1,\ p_k \notin c^{taints}_j. \tag{11}$$

3 Real-Time Multi-workflow Energy-Efficient Scheduling Algorithm

3.1 Scheduling Architecture

The real-time multi-workflow scheduling architecture is shown in Fig. 1. The architecture can be divided into three parts: end user, instance cluster and scheduler. End users can submit workflow requests to the system at any time. The cloud service platform provides instance clusters to handle the requests submitted by users. The scheduler is to arrange the workflow submitted by users into the instance cluster reasonably, so that the whole system can operate efficiently. Schedulers are mainly divided into several components: request preprocessor, task pool, rescheduling trigger, scheduling decision maker, executor and monitor. After the workflow request is accepted by the scheduler, the request preprocessor first decomposes the workflow submitted by the user into tasks and sets the deadline and priority for each task. These tasks will be placed in the task pool (Step 1). The rescheduling trigger will receive the status in the task pool (Step 2) and inform the scheduling decision-maker whether to perform general scheduling or rescheduling (Step 3). After receiving instructions, the scheduler extracts the task information (Step 4) to be dispatched from the task pool and sends the generated scheduling decisions to the executor (Step 5). During the execution period, adjust the running state of the instance cluster according to the received instructions (Step 6). The monitor will constantly monitor the status of cluster (Step 7) and update the information in the task pool (Step 8).

3.2 Request Preprocessor

This component sets the sub-deadline for the tasks contained in the request submitted by the user, and sorts them according to the priority. The sub-deadline setting of each task is related to the topology level of the task in request. The topological level of a t_{mi} is defined as Eq. (12). For each level, we calculate the task with the largest number of instructions in the level as the critical task of the level. We take the duration of the task on the fastest machine in the system as the execution time of this level ($level_l^{time}$).

$$Lev(t_{mi}) = \begin{cases} 1, & Pred(t_{mi}) = \emptyset \\ \underset{t_{mj} \in Pred(t_{mi})}{max} Lev(t_{mj}) + 1, other. \end{cases} \tag{12}$$

$$Level_l^t = \{t_{mi} | Lev(t_{mi}) = l)\} \tag{13}$$

$$level_l^{time} = c_j^{load} + \frac{\hat{t}_l}{\hat{p}^{ips} \cdot c_j^{cpu}}, c_j^{type} = \hat{t}_l^{type}, \hat{p} = \underset{p_k \in \mathcal{P}}{\arg\max}(p_k^{ips}), \hat{t}_l = \underset{t_{mi} \in Level_l^t}{\arg\max}(t_{mi}^I), \tag{14}$$

where $\hat{t}_l$, $\hat{p}$ and c_j^{load} represent the task with maximum number of instructions in level l, the fastest single core PM and the preparation time before the container is able to handle tasks after deploying, respectively. The estimated processing time for w_m can be model as Eq. (15), where L denotes the maximum level contained in the w_m. After that, we can set the sub-deadline of t_{mi} as Eq. (16).

$$w_m^{et} = \sum_{l=1}^{L} level_l^{time}, \tag{15}$$

$$t_{mi}^{d} = \frac{level_l^{time}}{w_m^{et}} \cdot w_m^{trt} + w_m^{at},\ Lev(t_{mi}). \tag{16}$$

Our priority ranking of tasks is mainly calculated according to the number of subsequent tasks related to the task which include all immediate and mediate successors. We defined $d(t_{mu})$ as the set of tasks which are dependent on t_{mu}.

$$d(t_{mu}) = (\bigcup_{t_{mv} \in Sub(t_{mu})} d(t_{mv})) \cup t_{mu}, \tag{17}$$

$$Rank(t_{mu}) = |d(t_{mu})|, \tag{18}$$

$Sub(t_{mv}) = \{t_{mu} | e_{u,v}^{m} = 1, \forall\ t_{mu} \in \mathcal{T}_m\}$ represents the set of immediate successors of t_{mv}. A task with higher rank means the task has higher scheduling priority than other tasks at the same topology level.

3.3 Task Pool

Task pool is a mapping of task states in the system. In the cluster, tasks are mainly divided into the following types: *Task not ready:* Tasks whose predecessors have not been completed and have not entered the executable state; *Task ready:* Tasks whose predecessors have completed but are not scheduled by the system; *Task scheduled:* Tasks that have been scheduled to the container but have not started to execute; *Task running:* Tasks being processed by the container; *Task finished:* Tasks that have been completed on time; *Task fail:* Tasks that have timed out.

3.4 Re-scheduling Trigger

In most scheduling strategies, the system only schedules tasks once, which often falls into local optimization in real-time scenarios. In the real-time system, the request will arrive at the cloud platform at any time, thus the state of the task in the system will fluctuate with time. For tasks that have been scheduled before but the container has not started to execute, there may be a better scheduling decision in the current new system state. However, if every new task arrives, rescheduling all the tasks in the system will greatly increase the scheduling cost of the system and affect the quality of service. To trade off this decision, we use $\theta_t = \frac{|Newtask_t|}{|Alltask_t|}$ to describe the state of unprocessed tasks in the system at t moment. $Newtask_t$ represents the set of new executable tasks at t moment and $Alltask_t$ represents the set of tasks that can be executed but not started. $|Newtask_t|$ and $|Alltask_t|$ mean the number of elements in corresponding set. α is re-schedule factor, $(0 < \alpha < 1)$. If $\theta_t > \alpha$, this means that the task state in the system has changed greatly, and rescheduling decision will be a better choice.

3.5 Scheduling Decision-Maker

After the above stages, the system has completed the screening and sorting of the tasks to be scheduled. We will schedule all tasks once according to the priority of the tasks. The selection of target container and machine should consider the scheme with the lowest energy consumption as far as possible on the basis of ensuring that the task can be completed on time, and also consider the universality of the machine. If too many containers are deployed on a machine with higher versatility (the machine is used as taint with fewer tasks), more picky tasks (tasks with many Taints) may not have enough resources to deploy in the cluster. Therefore, the selection of scheduling objectives should be comprehensively determined by weighing the new energy consumption caused by the deployment of the machine and the universality of the target machine.

The calculation of new energy consumption after deployment can be divided into the following situations: $\mathcal{A}$: There is a deployed container, and the running time of the container after deployment will not exceed the maximum running time of the container deployed on the PM to which the container is deployed. $\mathcal{B}$: There is a deployed container, and the running time of the container after deployment will exceed the maximum running time of the container deployed on the PM to which the container is deployed. $\mathcal{C}$: A new container needs to be deployed on the PM that has been powered on, and the running time of the PM will not be extended. $\mathcal{D}$: A new container needs to be deployed on the PM that has been powered on, which will cause the increment of running time of the PM. $\mathcal{E}$: Need to open a new PM to deploy the container. $\mathcal{F}$: The cluster does not have enough resources to complete the task on time. Considering the above situations, the new energy consumption caused by the deployment tasks can be calculated by Eq. (19).

$$\Delta E = \begin{cases} \frac{c_j^{cpu}}{p_k^{cpu}} \cdot (p_k^{e\_total} - p_k^{e\_base}) \cdot \bar{t}_r, & \mathcal{A} \\ \frac{c_j^{cpu}}{p_k^{cpu}} \cdot (p_k^{e\_total} - p_k^{e\_base}) \cdot \bar{t}_r + \bar{t}_e \cdot p_k^{e\_base}, & \mathcal{B} \\ \frac{c_j^{cpu}}{p_k^{cpu}} \cdot (p_k^{e\_total} - p_k^{e\_base}) \cdot (\bar{t}_r + \bar{t}_p), & \mathcal{C} \\ \frac{c_j^{cpu}}{p_k^{cpu}} \cdot (p_k^{e\_total} - p_k^{e\_base}) \cdot (\bar{t}_r + \bar{t}_p) + \bar{t}_e \cdot p_k^{e\_base}, & \mathcal{D} \\ \frac{c_j^{cpu}}{p_k^{cpu}} \cdot (p_k^{e\_total} - p_k^{e\_base}) \cdot (\bar{t}_r + \bar{t}_p) + (\bar{t}_r + \bar{t}_p + \bar{t}_s) \cdot p_k^{e\_base}, & \mathcal{E} \end{cases} \tag{19}$$

$$t_r = \frac{t_{mi}^I}{p_k^{ips} \cdot c_j^{cpu}}, \tag{20}$$

where c_j, p_k are the container and PM for task t_{mi} plan to deployment, respectively. $\bar{t}_r$, $\bar{t}_e$, $\bar{t}_p$ and $\bar{t}_s$ are task execution time, extended execution time of PM, start time of container and start time of PM. For each PM in the platform, the system calculates the universality of each PM (p_k^u, (21)) according to the taints node information submitted by the user. $\mathcal{C}_{avail}$ is a set of containers that can be deployed to p_k. The higher p_k^u means that p_k has higher versatility.

$$p_k^u = \frac{|\mathcal{C}_{avail}|}{|\mathcal{C}|}, \tag{21}$$

In order to meet the scheduling objectives, the system needs to allocate tasks to lower energy consumption and more "exclusive" PMs. $Q_{i,j,k} \in \mathcal{Q}$ is calculated by Eq. (22), which denote the energy consumption of deploying t_i to container c_j and PM p_k. The system will deploy the task with a higher Q scheme.

$$Q_{i,j,k} = \beta \Delta E_{i,j,k} + (1-\beta) p_k^u, \tag{22}$$

where β is the weight of energy consumption in scheduling decision, $(0 < \beta < 1)$.

The detailed pseudocode of the our algorithm can be found in Appendix A.

4 Performance Evaluation

4.1 Experimental Setup

We use five well-known workflows widely used in previous work to evaluate the algorithm: Montage, LIGO, Epigenomics, CyberShake and SIPHT[1]. Montage is I/O intensive, LIGO and CyberShake are CPU intensive containing task with high memory requirements. Epigenomics and SIPHT are CPU intensive. Details of these workflows are described in [13]. Similar to [7], we use the rule that the arrival interval of any two requests in the actual scenario obeys the Poisson distribution to generate a real-time workflow.

We establish a simulation platform in Python which generates the request workflow according DAX format file. The platform runs on a Ubuntu 20.04.2 LTS a 64bit PC with i5-9500 3.0 GHz CPU and 32 GB RAM, python 3.8.5. In the experiment, we use eight types of PMs (see Table 1.), and the relevant configuration parameters of these PMs are obtained by [1]. We set re-schedule factor α to 0.05. We set w_m^{min} as the time it takes for the longest critical path in the workflow to run on the fastest machine in the cluster. We select deadline factor γ according to the uniform distribution $[2, 8]$ and then assign the deadline calculated by $\gamma \cdot w_m^{min}$ to workflows.

Table 1. Real-world PM types

Type	CPU cores	Mem (GB)	basic power (w)	full load power (w)
PM1	4	4	43	115
PM2	4	8	63	115
PM3	8	8	89.4	173
PM4	8	16	155	269
PM5	8	18	173	334
PM6	8	32	226	294
PM7	16	16	299	521
PM8	32	32	260	748

[1] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

We constructed five homogeneous scenarios (with only one workflow type) and one heterogeneous scenario (with all workflow types) using the above workflows. In each scenario, the experimental parameters are composed of three parameters, including arrival rate λ, workflow scale *scale* and compatibility δ. Similar to previous works [3,7], λ includes the arrival rate of four poisson distribution of 1 workflows/s, 5 workflows/s, 10 workflows/s and 15 workflows/s. Workflow scale represents the workflow intensity of three requests, *small*, *medium* and *large*. They are composed of a mixture of multiple workflows with a total of 1000, 2000 and 3000 tasks. The workflow contains more tasks, if it is in large workflow scale. We set compatibility to 0.2, which represent 20% of the PMs in the cluster as taints of the task. This represents the degree to which the task is picky about the PMs in the cloud service provider cluster. For each workflow structure, we conducted experiments on the different values of the above three parameters, and a total of 24 groups of experiments were conducted.

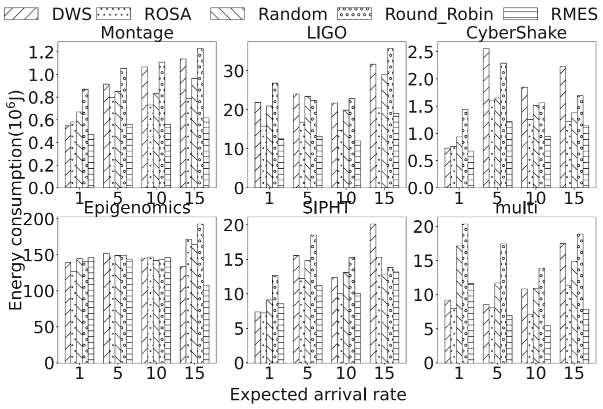

Fig. 2. The energy consumption of each workflow with DWS, ROSA, Round-Robin, Random and RMES

4.2 Comparison Algorithm

To verify the effectiveness of our proposed algorithm, we compare it with four existing algorithms: Random and Round-Robin, ROSA [5] and DWS [3]. Random is a random scheduling strategy. After disassembling the arriving workflow into sub tasks, the system randomly assigns the tasks to the container and schedules the container to run on the random machine. Round-Robin is one of the default scheduling strategies of Kubernetes. The algorithm schedules tasks to appropriate containers and PMs according to the polling rules. ROSA is an uncertainty-aware online scheduling algorithm to schedule dynamic and multiple workflows with deadlines. The algorithm first estimates the completion time of the task, and then schedules the task to minimize the cost. DWS is an online heuristic algorithm, which aims to minimize the cost of renting service instances under the deadline. When the new workflow arrives, the system sets heuristic rules according to the cost deadline to schedule the task to a more reasonable instance. To ensure the fairness of the experiment, we equivalently replace the optimization objective function in ROSA and DWS with the same energy consumption objective function as RMES.

4.3 Simulation Results

Arrival Rate. Figure 2 shows the energy consumption result of the algorithms under different workflow structures. The arrival rate is increased from 1 to 15 in the case of medium workflow scale. We can see that in most cases, the experimental results of RMES are excellent, which is significantly improved compared with other algorithms. When the arrival rate increases, our algorithm performs better. When the arrival rate is 1, our algorithm improves -1%, -10%, 18.45% and 34.11% respectively compared with DWS, ROSA, Random and Round-Robin. When arrival rate reaches 15, RMES increases to 40.57%, 19.42%, 27.89% and 39.40%. With the increasing arrival rate, the proportion of newly arrived tasks in the task pool will increase, resulting in large changes in the status of the task pool. Due to the setting of rescheduling mechanism in RMES, when the state of task pool changes greatly, the scheduler can reschedule the unexecuted tasks in the system in time, so that the scheduling results of tasks in the task pool are more in line with the current state of task pool.

Table 2. The energy consumption of each workflow with DWS, ROSA, Round-Robin, Random and RMES

Workflow, size	Random	Round-robin	DWS	ROSA	RMES
Montage, S	0.47	0.79	0.64	0.52	**0.37**
Montage, M	0.83	1.10	1.06	0.73	**0.55**
Montage, L	1.62	1.77	1.72	**1.17**	1.22
LIGO, S	13.12	18.96	9.95	**9.16**	9.64
LIGO, M	19.86	22.81	21.68	14.71	**11.98**
LIGO, L	23.17	24.69	27.85	19.74	**17.46**
CyberShake, S	0.85	0.95	0.80	0.75	**0.67**
CyberShake, M	1.51	2.55	1.84	1.25	**0.94**
CyberShake, L	1.85	2.44	2.17	1.60	**1.30**
Epigenomics, S	208.48	**205.63**	216.23	216.22	206.24
Epigenomics, M	**141.98**	144.15	145.72	147.13	146.19
Epigenomics, L	163.81	162.89	**160.84**	172.21	162.00
SIPHT, S	12.92	13.14	17.49	12.08	**11.18**
SIPHT, M	13.09	15.20	12.37	**10.02**	10.08
SIPHT, L	30.89	46.99	44.93	28.90	**23.99**
multi, S	6.21	8.06	6.01	4.44	**3.86**
multi, M	10.95	13.91	10.85	10.95	**5.51**
multi, L	27.49	34.94	24.80	17.12	**12.44**

Workflow Scale. Table 2 shows the energy consumption result of the algorithm under different sizes and different workflow structures when the arrival rate is 10 workflows/s. In Table 2, 'Montage, S' means Montage workflow small-scale test example. In addition to the two workflow structures with fluctuating completion rates, RMES has obvious advantages over other algorithms. Under the three workflow structures of multi, SIPHT and CyberShake, the improvement of RMES and the comparison algorithm with the best performance increases from 13.06%, 7.45% and 10.66% on a small scale to 27.33%, 16.98% and 18.75% on a large scale. We find that RMES algorithm has a greater improvement under the condition of large-scale workflow.

5 Conclusion

In this paper, we focus on the real-time energy-saving multi-workflow scheduling on container cloud. Firstly, we establish a cloud-based workflow scheduling model, which considers resource quantity and performance constraints of container deployment. Then we propose an real-time multi-workflow energy-efficient scheduling (RMES) algorithm. By executing tasks in parallel on the running PM, RMES can compress the time of the global process to reduce the base energy consumption. Furthermore, RMES introduces rescheduling mechanism, so that the task scheduling decision can be adjusted with the change of system state. Finally, we conduct several groups of experiments under the actual workflow conditions. Compared with other algorithms, RMES significantly reduces the energy consumption generated by CSP.

Acknowledgements. This work is supported by the Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001, and No. JCYJ20210324132406016.

Appendix for "An Energy-Efficient Scheduling Method for Real-Time Multi-workflow in Container Cloud"

A Detailed Pseudocode of the Proposed Algorithm

The detailed procedure is given in Algorithm 1. $task_{ready}$ and $task_{scheduled}$ represent task sets of type task ready and task scheduled in the task pool, respectively. $<Q_{n,i,j}, c_i, p_j>$ means assign $task_n$ to c_i running in p_j. Firstly, we evaluate the task state in the system and judge whether to reschedule the scheduled tasks according to the current task state of the system, as shown in lines 2–10 of Algorithm 1. Then, for the task to be scheduled, the system calculates the Q value of the task deployed on the existing container, and deploys the task to the container with the lowest Q value, as shown in lines 11–24 of Algorithm 1. For a task without a suitable container to run, the system will create a new container for it, calculate the Q value of the container deployed to each PM in the cluster, and select the PM with the lowest Q value to run the container, as shown in lines 25–41 of Algorithm 1.

Algorithm 1: RMES

```
   Input: $task_{ready}, task_{scheduled}, \mathcal{C}, \mathcal{P}$
   Output: $\{x_{n,j}\}$ ,$\{y_{i,j}\}$
1  $scheduletask \leftarrow \emptyset$;
2  Calculate $\theta_t$;
3  if $\theta_t > \alpha$ then
4      foreach $t_n \in task_{scheduled}$ do
5          $scheduletask \leftarrow scheduletask \cup \{t_n\}$;
6      end
7  end
8  foreach $t_n \in task_{ready}$ do
9      $scheduletask \leftarrow scheduletask \cup \{t_n\}$;
10 end
11 foreach $t_n \in scheduletask$ do
12     $target \leftarrow \emptyset$;
13     foreach $c_i \in \mathcal{C}$ do
14         if $c_i^{type} = task_n^{type}$ then
15             Calculate $Q_{n,i,j}$ according Eq. (22);
16             $target \leftarrow < Q_{n,i,j}, c_i, p_j >$;
17         end
18     end
19     if $target \neq \emptyset$ then
20         select $c_i$ with minimum $Q$;
21         $x_{n,j} \leftarrow 1$;
22         $scheduletask \leftarrow scheduletask - \{task_n\}$;
23     end
24 end
25 if $scheduletask \neq \emptyset$ then
26     foreach $t_n \in scheduletask$ do
27         $target \leftarrow \emptyset$;
28         create container $c_i$;
29         foreach $pm_j \in \mathcal{P}$ do
30             Calculate $Q_{n,i,j}$ according Eq. (22);
31             $target \leftarrow < Q_{n,i,j}, c_i, p_j >$;
32         end
33         if $target \neq \emptyset$ then
34             select $c_i$ with minimum $Q$;
35             $x_{n,j} \leftarrow 1$;
36             $y_{i,j} \leftarrow 1$;
37             $scheduletask \leftarrow scheduletask - \{task_n\}$;
38             $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_i\}$;
39         end
40     end
41 end
```

References

1. Third quarter 2021 specpower_ssj2008 results (2021). www.spec.org/power_ssj2008/results/res2021q3/
2. Al-Dulaimy, A., Taheri, J., Kassler, A., HoseinyFarahabady, M.R., Deng, S., Zomaya, A.: Multiscaler: a multi-loop auto-scaling approach for cloud-based applications. IEEE Trans. Cloud Comput. **10**(4), 2769–2786 (2022)
3. Arabnejad, V., Bubendorfer, K., Ng, B.: Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds. Futur. Gener. Comput. Syst. **100**, 98–108 (2019)
4. Beloglazov, A., Buyya, R., Lee, Y.C., Zomaya, A.: A taxonomy and survey of energy-efficient data centers and cloud computing systems. In: Advances in Computers, vol. 82, pp. 47–111 (2011)
5. Chen, H., Zhu, X., Liu, G., Pedrycz, W.: Uncertainty-aware online scheduling for real-time workflows in cloud service environment. IEEE Trans. Serv. Comput. **14**(4), 1167–1178 (2018)
6. Cheng, M., Li, J., Nazarian, S.: DRL-cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: 2018 23rd Asia and South Pacific Design Automation Conference, pp. 129–134 (2018)
7. Deng, F., Lai, M., Geng, J.: Multi-workflow scheduling based on genetic algorithm. In: 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 300–305. IEEE (2019)
8. Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., Zeng, J.: Q-learning based dynamic task scheduling for energy-efficient cloud computing. Futur. Gener. Comput. Syst. **108**, 361–371 (2020)
9. Havet, A., Schiavoni, V., Felber, P., Colmant, M., Rouvoy, R., Fetzer, C.: Genpack: a generational scheduler for cloud data centers. In: 2017 IEEE International Conference on Cloud Engineering (IC2E), pp. 95–104 (2017)
10. Hu, B., Cao, Z., Zhou, M.: Scheduling real-time parallel applications in cloud to minimize energy consumption. IEEE Trans. Cloud Comput. **10**(1), 662–674 (2022)
11. Hu, Y., Zhou, H., de Laat, C., Zhao, Z.: Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Futur. Gener. Comput. Syst. **102**, 562–573 (2020)
12. Hussain, M., Wei, L.F., Lakhan, A., Wali, S., Ali, S., Hussain, A.: Energy and performance-efficient task scheduling in heterogeneous virtualized cloud computing. Sustain. Comput. Inform. Syst. **30**, 100517 (2021)
13. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst. **29**(3), 682–692 (2013)
14. Merkel, D., et al.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
15. Sun, Z., Huang, H., Li, Z., Gu, C., Xie, R., Qian, B.: Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud. Expert Syst. Appl. **228**, 120401 (2023)
16. Sun, Z., Zhang, B., Gu, C., Xie, R., Qian, B., Huang, H.: ET2FA: a hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud. IEEE Trans. Serv. Comput. **16**(3), 1807–1821 (2023)
17. Zhang, F., Tang, X., Li, X., Khan, S.U., Li, Z.: Quantifying cloud elasticity with container-based autoscaling. Futur. Gener. Comput. Syst. **98**, 672–681 (2019)