Real-Time Workflows Oriented Hybrid Scheduling Approach With Balancing Host Weighted Square Frequencies in Clouds

CHEN LI\(^1\), LISU HUO\(^2\), AND HUANGKE CHEN\(^2\)

\(^1\)Graduate School, National University of Defense Technology, Changsha 410073, China
\(^2\)College of Systems Engineering, National University of Defense Technology, Changsha 410073, China

Corresponding author: Lisu Huo (huolisu09@nudt.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 71801218 and Grant 61603404.

ABSTRACT  High energy consumption in cloud data centers has become one of the main obstacles to green cities, and an urgent problem to be solved. So far, a large number of scheduling algorithms have been developed to reduce energy consumption for executing workflows. However, most existing algorithms have obvious defects in energy and resource efficiency, because they schedule workflow tasks to hosts directly and ignore that the host is so powerful that a single workflow task cannot make full use of its resources. To resolve the issue, a new scheduling architecture is designed for cloud data centers. Then, two principles are derived: one suggests that hybrid scheduling tasks from different workflows is helpful for improving resource utilization; the other one deduces that balancing host weighted square frequencies can minimize total power consumption of active hosts under a given resource requirement. On the basis of the scheduling architecture and the two principles, an online scheduling algorithm, namely NEAT, is proposed to schedule dynamic workflows with deadlines. Furthermore, three strategies for dynamically adjusting the available virtual machines (VMs) and active hosts are proposed and integrated into NEAT to improve the energy and resource efficiency for cloud data centers. Finally, the proposed NEAT is compared with three existing algorithms using real-world workflow traces to demonstrate its superior performance with respect to energy and resource efficiency while guaranteeing the timing requirements of workflows. Compared with baseline algorithms, NEAT is capable of reducing energy consumption for cloud data centers by an average of 40% in the range from 34.0% to 45.67%.

INDEX TERMS  Cloud computing, energy efficiency, real-time, workflow, scheduling, balancing.

I. INTRODUCTION  With the soaring expansion of cloud computing services, the number of hosts for supporting these services in cloud data centers is increasing sharply [1]. The number of hosts in a cloud data center is up to tens of thousands, or even hundreds of thousands.\(^1\) It is inevitable that such large size of hosts expend massive electric energy, which comprises around 1.5% of the global energy consumption [2], [3]. Nowadays, the increasing energy costs [4] and negative environmental impacts have pressurized cloud providers to improve energy efficiency for their cloud data centers.

Contrasting to high energy consumption, the resource utilization in cloud data centers is very low, and the average value is around 15% [5], [6]. What is worse, hosts in low resource utilization still have very high energy powers, even some idle hosts dissipate more than 60% of their peak energy powers [6], [7]. Thus, a considerable proportion of energy in cloud data centers can be saved by improving hosts’ resource utilization and shutting down idle hosts.

The reason for such low resource utilization of hosts is threefold. Firstly, it is unusual for a single workflow task to fully use the resources of a powerful host, whose average resource utilization is less than 40% when executing a workflow task [8]. Then, the workflow applications in cloud data centers typically comprise of a great deal of tasks with data dependencies [8]–[10], e.g., Montage and CyberShake.

\(^1\)https://github.com/google/cluster-data
The data dependencies among tasks inevitably result in a large number of idle time slots on resources, which further reduce resource utilization. Thirdly, workflow applications are submitted by a wide range of end-users dynamically, the data centers’ workloads will fluctuate over the time. For many cloud data centers, their peak workloads are often many times their average workloads [6], [7], thus their resources’ over-provisioning is inevitable at non-peak time.

Motivation: Most existing works often neglect that a single workflow task cannot fully use a host’s resource, and directly allocate workflow tasks to hosts for execution, causing significant low resource efficiency. What is worse, the existing works rarely take full advantage of the idle time slots on resources. In this paper, we strive to incorporate the advantages of virtualization technology and the elasticity of clouds into energy-efficient scheduling approach. Specifically, with the virtualization technology, a host is split into several independent units (i.e., VMs) to execute multiple workflow tasks simultaneously, and the available VMs can be scaled up/down dynamically to adjust the number of active hosts. Besides, the fact that tasks from different workflows have not data dependencies is explored to further improve resource utilization.

Contributions: The main technical contributions are as follows:
- A novel reactive scheduling architecture is designed to transform an online scheduling problem to multiple static scheduling processes. This scheduling architecture also enables each static scheduling process to reschedule all the waiting workflow tasks by taking full advantage of all available information at each scheduling time.
- We derive two principles: 1) executing tasks from different workflows alternately at the same VMs is helpful for improving VMs’ resource utilization; 2) the total power consumption is minimized by balancing weighted square frequencies of hosts, when the resource requirements and active hosts are given.
- Based on the scheduling architecture and the two principles, we propose an energy-efficient online scheduling Algorithm, NEAT, to improve the energy and resource efficiency for executing dynamic workflows with deadlines.
- We conduct three groups of experiments using real-world workflow traces to compare the proposed NEAT with other three existing algorithms.

The paper is organized as follows. Section II reviews the relevant literatures. Section III introduces the scheduling architecture and problem formulation, followed by detailing the proposed algorithm in Section IV. Section V reports the experimental results. Section VI concludes this paper.

II. RELATED WORK
Reducing energy consumption for executing workflows in clouds has become one of the hottest directions. In past years, many energy-aware workflow scheduling algorithms have been reported, and these existing algorithms can be roughly classified as 1) population-based evolutionary algorithms and 2) heuristic algorithms.

The population-based evolutionary algorithms have become an attractive optimization technique for resolving the workflow scheduling problems. For instance, Mezmaz et al. employed genetic algorithm to make a trade-off between finish time and energy consumption for workflow scheduling in clouds [11]. Yao et al. employed a multi-swarm algorithm to simultaneously optimize makespan, cost, and energy consumption for workflows in clouds [12]. Zhang et al. proposed a bi-objective genetic algorithm to schedule workflows in clouds, to minimize energy consumption and enhance system reliability [13]. However, these population-based evolutionary approaches have much higher time complexity, and are not suitable for scheduling dynamic workflows in clouds.

With respect to the heuristic algorithms, they can be further divided into three categories: list-based, clustering-based, and duplication-based approaches. For list-based workflow scheduling algorithms, they mainly include two stages: assigning weights to tasks and schedule tasks in descending order of their weights. For example, Topcuoglu et al. extended the classical list-based workflow scheduling algorithm HEFT [14] to search for Pareto solutions with respect to finish time and energy consumption [15]. Tao et al. designed a parallel intelligent algorithm to maximize the reliability, minimize the execution time, cost and energy [16]. Schranzhofer et al. developed an algorithm to map tasks to processing resources, striving to minimize the expected average power consumption [17]. Regarding clustering-based scheduling algorithms, they firstly group all the tasks in a workflow into many clusters, and then schedule all tasks in each cluster [18], [19]. The fact that there exists no data transmission among tasks on the same computing resources is exploited by duplication-based algorithms to improve resource efficiency and workflows’ finish time. For example, Zong et al. developed two tasks duplication based scheduling strategies to reduce makespan and energy consumption for executing workflows on clusters [20]. However, these existing works focused on scheduling a single workflow, and directly scheduling workflow tasks to hosts for execution. In this work, tasks from different workflows having no data dependencies is explored to schedule tasks from many workflows in a hybrid way. Besides, the varied resource requirements of workflow tasks and the changing workload of cloud data centers are also considered in this work.

So far, we have published several papers on workflow scheduling in clouds [21], [22]. Compared with the prior works, this paper derives two principles to improve resource utilization, minimize total power consumption. Then the scheduling algorithm is developed with three novel scale down strategies. Moreover, the time complexity of the proposed algorithm is detailed.

III. MODELING AND PROBLEM FORMULATION
In this section, the models of workflows, scheduling architecture and workflow scheduling are introduced.
A. WORKFLOW MODELING

The dynamic workflows in a cloud data center can be characterized by \( W = \{w_1, w_2, \ldots, w_m\} \), and \( w_i \) (\( i = 1, 2, \ldots, m \)) stands for the \( i \)-th workflow. The workflow \( w_i \) can be formulated as \( w_i = \{a_i, d_i, G_i\} \), where \( a_i \), \( d_i \), and \( G_i \) denote \( w_i \)'s arrival time, deadline and structure, respectively. For workflow \( w_i \), its structure \( G_i \) can be further described by a directed acyclic graph (DAG) \( G_i = (T_i, E_i) \), where \( T_i = \{t^i_1, t^i_2, \ldots, t^i_{|T_i|}\} \) represents the task set in workflow \( w_i \); \( E_i \subseteq T_i \times T_i \) indicates the data dependencies among workflow tasks. An edge \( e_{pj}^i \in E_i \) means the start of task \( t^i_j \) depends on the output data of executing task \( t^i_p \), then \( t^i_j \) and \( t^i_p \) call the other side as immediate successor and predecessor. All the immediate successors of task \( t^i_j \) is modeled as \( \text{pred}(t^i_j) \), and all its successors are denoted as \( \text{succ}(t^i_j) \).

B. SYSTEM MODELING

One cloud data center typically consists of large-scale heterogeneous hosts, i.e., \( H = \{h_1, h_2, \ldots, h_n\} \), where \( n \) denotes the host count. For a host \( h_k \in H \), it can be modeled as \( h_k = \{m_k, sc_k, n_k, p^\text{max}_k, (f_k, v_k), VM_k\} \), where \( m_k \), \( sc_k \), \( n_k \), and \( p^\text{max}_k \) denote \( h_k \)'s memory size, storage capacity, network bandwidth, and peak energy power, respectively; the discrete set of voltages-frequency pairs of host \( h_k \) is described as \( (f_k, v_k) = \{(f^1_k, v^1_k), \ldots, (f^\text{max}_k, v^\text{max}_k)\} \); the VMs on host \( h_k \) is modeled as \( VM_k = \{\text{vm}_{k,1}, \text{vm}_{k,2}, \ldots, \text{vm}_{k,|VM_k|}\} \), and \( \text{vm}_{k, l} \) stands for the \( l \)-th VM on host \( h_k \). For VM \( \text{vm}_{k, l} \), it can be further depicted as \( \text{vm}_{k, l} = \{f_{k,l}, m_{k,l}, s_{k,l}\} \), where \( f_{k,l}, m_{k,l} \) and \( s_{k,l} \) represent the requirements of \( \text{vm}_{k, l} \)'s CPU frequency, memory and storage capacity, respectively. With virtualization technology, a host’s hardware resources can be separately shared by many VMs, and these VMs can migrate among different physical hosts.

In this work, we propose a new reactive scheduling architecture for deploying workflows on clouds, which is shown in Fig. 1. Similar to traditional multiprocessor systems, the scheduling architecture in a cloud data center can be divided into three layers: applications, scheduling and resources. Throughout the architecture, the scheduling layer bridges the application layer and resource layer. However, differing from traditional systems, the resource layer in clouds consists of two layers: VM layer and host layer. Moreover, the available VMs in VM layer can be scaled up/down dynamically based on the workload of data centers.

The main focus of this work is the scheduling layer, which consists of the following four components: 1) a task pool (TP) which contains all the waiting workflow tasks; 2) a schedulability analyzer whose function is to produce the mappings between workflow tasks and VMs, and the plan of adjustment for the hosts and VMs; 3) a task controller that implements the plan of schedulability analyzer, and allocates tasks to corresponding VMs; 4) a resource controller which executes the adjustment plan of resources. The novelty of this scheduling architecture mainly embodies that all the waiting workflow tasks are queuing in the TP rather than on the VMs directly, and only tasks that are ready to be executed will be allocated to VMs. This architecture makes it easy for the clouds to obtain all waiting tasks upon the arrival of new workflows, and reschedule those tasks, together with the tasks from the new workflows, for optimal scheduling solutions. With this architecture, the online scheduling problem is simplified into a sequence of static scheduling problems.

C. ENERGY MODEL OF HOSTS

Assuming that \( p^\text{max}_k \) and \( s_k \) are the maximal energy power and the ratio of static energy power for host \( h_k \). Host \( h_k \)'s energy power can be described as [7]:

\[
p_k = s_k \cdot p^\text{max}_k \cdot y_k^3 + \frac{(1 - s_k) \cdot p^\text{max}_k}{(f^\text{max}_k)^3} \cdot (f_k)^3,
\]

(1)
where \( y_k^t \in \{1, 0\} \) means whether \( h_k \) is active at time \( t \); \( f_{k, \text{max}}^t \) and \( f_k^t \) are \( h_k \)'s maximum CPU frequency and working CPU frequency at time \( t \), respectively.

Then, the energy consumption (EC) of \( n \) hosts from time instant \( st \) to \( et \) is computed as follows:

\[
EC = \sum_{i=1}^{n} \int_{st}^{et} (s_k \cdot p_k^{\text{max}} \cdot y_k^t + \frac{(1 - s_k) \cdot p_k^{\text{max}}}{(f_{k, \text{max}}^t)^3} \cdot (f_k^t)^3) \, dt, \tag{2}
\]

where parameters \( y_k^t \) and \( f_k^t \) are time-dependent variables, and other parameters are fixed values for host \( h_k \).

### D. PROBLEM FORMULATION

We use the record variable \( x_{j, kl}^t \) to indicate the mapping relationship between task \( t_j^t \) and VM \( vm_{k, l} \) on host \( h_k \). The parameter \( x_{j, kl}^t \) is 1 if task \( t_j^t \) is mapped to VM \( vm_{k, l} \), otherwise, \( x_{j, kl}^t \) equals to 0, i.e.,

\[
x_{j, kl}^t = \begin{cases} 1, & \text{if } t_j^t \text{ is mapped to } vm_{k, l}, \\ 0, & \text{otherwise}. \end{cases} \tag{3}
\]

The data dependencies among tasks in each workflow comes to the following constraint:

\[
ft_{p, kl}^t + t_{p, j}^l \leq st_{j, kl}^t, \quad \forall (p, j) \in E_t. \tag{4}
\]

where \( ft_{p, kl}^t \) denotes task \( t_p^t \)'s finish time, and \( t_{p, j}^l \) stands for the data transmission time between task \( t_p^t \) and task \( t_j^t \).

Once all the tasks in \( w_i \) are scheduled to VMs, \( w_i \)'s finish time \( ft_i \) refers to the maximum finish time of all these tasks.

\[
ft_i = \max_{t_j^t \in T_i} (ft_{j, kl}^t). \tag{5}
\]

Another constraint in workflow scheduling comes from meeting workflows’ deadlines, which is formulated as follows.

\[
ft_i \leq d_i, \quad \forall w_i \in W. \tag{6}
\]

In addition, the limited resource of a host also restricts the number of VMs on it, which causes the following constraint.

\[
f_{k, \text{max}} - \sum_{l=1}^{[VM]} f_{k, l}^t \geq 0, \quad \forall h_k \in H; \tag{7}
\]

\[
m_k - \sum_{l=1}^{[VM]} m_{k, l} \geq 0, \quad \forall h_k \in H. \tag{7}
\]

Satisfying the constraints in formulas (4), (6) and (7), the first optimization objective is to minimize the energy consumption for completing the workflow set \( W \), i.e.,

\[
\min \sum_{k=1}^{n} \int_{st}^{et} (s_k \cdot p_k^{\text{max}} \cdot y_k^t + \frac{(1 - s_k) \cdot p_k^{\text{max}}}{(f_{k, \text{max}}^t)^3} \cdot (f_k^t)^3) \, dt. \tag{8}
\]

where \( n \) denotes the host count; \( st \) and \( et \) are the start time and the end time for executing the workflow set \( W \), respectively.

Apart from the energy consumption, resource utilization is another important metric to be optimized. Thus, we try to maximize hosts’ average resource utilization:

\[
\max \left( \sum_{i=1}^{m} \sum_{j=1}^{[T]} cpu_{j, i} \cdot ct_{j, i} \right) \left( \sum_{k=1}^{n} f_{k, \text{max}} \cdot wt_{k} \right), \tag{9}
\]

where \( m \) and \( [T] \) represent the workflow count of \( W \) and the task count of workflow \( w_i \), respectively; \( cpu_{j, i} \) and \( ct_{j, i} \) are task \( t_j^t \)'s CPU frequency requirement and computation time; \( n \) denotes the number of hosts in clouds; and \( wt_{k} \) represents host \( h_k \)'s active time.

### IV. ALGORITHM DESIGN

Since workflow scheduling on multiple machines belongs to typical NP-complete problem, its optimal solutions can be gained only for the problems with small size. Therefore, we propose a reactive heuristic method to make schedulability analysis for dynamic workflows with deadlines, followed by the algorithm description and its time complexity.

### A. PRELIMINARIES

Before scheduling workflows, how to rank the their tasks is a key issue. In this paper, we define the latest start time \( lst_j^t \) of each task, and use it to rank the workflow tasks. The \( lst_j^t \) of task \( t_j^t \) is defined as the latest time, before which \( t_j^t \) should be started; otherwise, the finish time \( ft_i \) of workflow \( w_i \) will be delayed and may miss its deadline.

**Definition 1**: For task \( t_j^t \), its latest start time \( lst_j^t \) is defined as:

\[
lst(t_j^t) = \begin{cases} d_i - ct_j^t, & \text{if } succ(t_j^t) = \phi, \\ \min_{t_j^t \in succ(t_j^t)} \{lst(t_j^t) - t_j^t - ct_j^t\}, & \text{otherwise}. \end{cases} \tag{10}
\]

where \( succ(t_j^t) \) represents a task set including all the immediate successor tasks of task \( t_j^t \). Then, task \( t_j^t \)'s latest finish time \( ftt_j^t \) is defined as: \( ftt_j^t = lst_j^t + ct_j^t \).

Since tasks in a single workflow have precedence constraints, a large number of idle time slots will be left over in VMs when executing workflows. It is worth noting that there is no data dependency among tasks from different workflows. If tasks in different workflows are executed in proper sequences at the same VMs, the idle time slots in VMs can be reduced or even removed. Therefore, we can get the following principle.

**Principle 1**: Assume task \( t_j^t \) and task \( t_j^t \) in workflow \( w_i \) share the same VM \( vm_{k, l} \), and the finish time of task \( t_j^t \) is less than the start time of task \( t_j^t \), i.e., \( ft_{j, kl}^t < st_{j, kl}^t \). Let the start time \( st_{p, kl}^t \) of task \( t_p^t \) in workflow \( w_p \) be bigger than the start time \( st_{j, kl}^t \) of task \( t_j^t \) on VM \( vm_{k, l} \), i.e., \( st_{p, kl} < st_{j, kl} \). If task \( t_p^t \) satisfies \( \max (ft_{p, kl}^t + st_{p, kl}^t + et_{p, kl}^t) \leq st_{j, kl}^t \), then allocating task \( t_p^t \) between \( t_j^t \) and \( t_j^t \) will reduce the idle time slot of VM \( vm_{k, l} \).
The total energy power of host set is a variable representing the frequency of host active hosts, and the total CPU frequency of host set is minimum if turned on/off when scaling up/down the computing resources. For the first issue, we give the Principle 2.

Principle 2: Let \( H_a = \{ h_1, h_2, \ldots, h_{|H_a|} \} \) be the set of active hosts, and the total CPU frequency of host set \( H_a \) is supposed to be a fixed value, i.e., \( F = \sum_{k=1}^{|H_a|} f_k \), where \( f_k \) is a variable representing the frequency of host \( h_k \in H_a \). The total energy power of host set \( H_a \), denoted as \( \sum_{k=1}^{|H_a|} p_k \), is minimum if \( c_1 f_1^2 = c_2 f_2^2 = \cdots = c_{|H_a|} f_{|H_a|}^2 \), where \( c_k = \frac{(1-s_k) \cdot p_k^{max}}{f_k^{max}} \) is a constant for host \( h_k, 1 \leq k \leq |H_a| \).

Proof: Based on Eq. (1), the power \( p_k \) of active host \( h_k \) can be rewritten as:

\[
p_k = s_k \cdot p_k^{max} \cdot y_i^j + \frac{(1-s_k) \cdot p_k^{max}}{f_k^{max}} \cdot (f_k^j)^3
\]

Define the Lagrange function of formula (11) as:

\[
L(f, \lambda) = \sum_{k=1}^{|H_a|} p_k + \lambda \left( \sum_{k=1}^{|H_a|} f_k - F \right)
\]

\[
= \sum_{k=1}^{|H_a|} s_k \cdot p_k^{max} + \sum_{k=1}^{|H_a|} c_k \cdot (f_k^j)^3 + \lambda \left( \sum_{k=1}^{|H_a|} f_k - F \right)
\]

where \( j = \{ f_1, f_2, \ldots, f_{|H_a|} \} \) is a vector, and \( \lambda \) is a constant.

Its optimal conditions are:

\[
\nabla_L L(f, \lambda) = \sum_{k=1}^{|H_a|} f_k - F = 0;
\]

\[
\nabla_k L(f, \lambda) = 3 \cdot c_k \cdot f_k^2 + \lambda = 0, \quad k = 1, 2, \ldots, |H_a|.
\]

Thus, if and only if \( c_1 f_1^2 = c_2 f_2^2 = \cdots = c_{|H_a|} f_{|H_a|}^2 = \frac{\lambda}{3} \), the total power \( \sum_{k=1}^{|H_a|} p_k \) is minimized. \( \square \)

Note that at one moment, the number of virtual machines in the data center is fixed, and the CPU frequency of each virtual machine is also fixed. Then, the total frequency that the data center needs to provide is fixed. This total CPU frequency is the assumption in Principle 2.

Principle 2 suggests that making \( c_k \cdot f_k^2 \) as balanced as possible among all the active hosts in a data center is able to minimize its total power.

B. PROPOSED ALGORITHM

Definition 2: Task \( t^j_i \) is defined as ready if it has no predecessor task, i.e., \( pred(t^j_i) = \emptyset \); or all its predecessors have been scheduled to VMs.

On the basis of the scheduling architecture in Fig. 1 and Principle 1 and Principle 2, we develop an online workflow scheduling algorithm, \( NEAT \), to deploy dynamic workflows with deadlines for execution in cloud data centers, while dynamically adjusting the available VMs and active hosts to achieve energy saving. Regarding traditional scheduling algorithms, once workflow tasks are scheduled, they will be dispatched to VMs or hosts immediately. Unlike those algorithms, \( NEAT \) only allocates workflow tasks that are ready to be executed to VMs, and keeps all the other waiting tasks in a task pool and their scheduling decisions can be reproduced to improve the resource and energy efficiency. The pseudocode of \( NEAT \) is shown in Algorithm 1.

Algorithm 1 Pseudocode of \( NEAT \)

1. \( taskPool \leftarrow \emptyset \); \n2. \( \text{foreach new workflow } w_j \text{ arrives do} \)
3. \( \text{Cancel the schedule decisions that are not executed;} \)
4. \( \text{Update status information for all the available VMs;} \)
5. \( \text{Calculate } lst^j_i \text{ for each task in } w_j \text{ using Eq. (10);} \)
6. \( \text{Add all the tasks of } w_j \text{ into } taskPool; \)
7. \( \text{while } taskPool \text{ exists unscheduled tasks do} \)
8. \( \text{RT} \leftarrow \text{get the ready tasks from } taskPool; \)
9. \( \text{Sort } RT \text{ in an increasing order according to } lst^j_i; \)
10. \( \text{foreach } t^j_i \in RT \text{ do} \)
11. \( \text{selVM } \leftarrow \emptyset; \quad \text{minWR } \leftarrow +\infty; \)
12. \( \text{canVMs } \leftarrow \text{all the VMs meeting } f_j, i \geq \text{cpu}_j; \)
13. \( \text{foreach } vm_{k,l} \in \text{canVMs} \text{ do} \)
14. \( \text{Calculate the finish time } f_{j,k,l}^i \text{ and waste resource } wr_{j,k,l}^i \text{ of task } t^j_i \text{ on } vm_{k,l}; \)
15. \( \text{if } f_{j,k,l}^i \leq lst^j_i \text{ and } wr_{j,k,l}^i < \text{minWR} \text{ then} \)
16. \( \text{selVM } \leftarrow vm_{k,l}; \quad \text{minWR } \leftarrow wr_{j,k,l}^i; \)
17. \( \text{if selVM } = \text{NULL} \text{ then} \)
18. \( \text{selectedVM } \leftarrow \text{ScaleUpResources}(); \)
19. \( \text{Schedule task } t^j_i \text{ to selVM;} \)

The proposed \( NEAT \) in Algorithm 1 works in a reactive fashion, once a new workflow arrives, it will be triggered to schedule all the new workflow tasks and the waiting workflow tasks in the task pool. The specific steps are as follows. Once a new workflow arrives the cloud data center (Line 2), the scheduling decisions that were previously generated and not executed will be cancelled (Line 3). The scheduling decisions include the mappings of waiting tasks in \( taskPool \) to VMs and the plan of scaling up/down resources. Then, the status information of all the VMs will be updated (Line 4). Before adding into the \( taskPool \) (Line 5), each task in the new
workflow will obtain a weight (i.e., \( l_{st} \)) (Line 5). After that, all the tasks in \( taskPool \) will be scheduled to VMs in a hybrid way to improve resource and energy efficiency (Lines 7-19). Particularly, in each scheduling iteration, all the ready tasks from different workflows in \( taskPool \) are selected and sorted (Lines 4-5). To conserve energy consumption, for each ready task \( t_j \), an initiated VM can meet its resource requirement (Line 12) and complete it before its latest finish time \( f_j \) with the minimum wasted resources \( w_j^{t_j,k,i} \) is selected for task \( t_j \) (Lines 13-16). If the strategy above cannot select feasible VM (Line 17), the function \( ScaleUpResources() \) will be called to add a new VM (Line 18).

The pseudocode for function \( ScaleUpResources() \) is detailed in Algorithm 2.

Algorithm 2 Function \( ScaleUpResources() \)

1. Select a VM \( vmt \) whose CPU freq. is just larger than \( \text{cpu}_{j}^{t_j} \);
2. \( AH \leftarrow \text{active hosts with unused CPU freq. larger than } \text{f}_{j} \);
3. \( destHost \leftarrow \text{NULL}; \minValue \leftarrow +\infty; \)
4. foreach \( h_k \in AH \) do
   5. if \( c_k \cdot f_{k}^{t_j} < \minValue \) then
      6. \( destHost \leftarrow h_k; \minValue \leftarrow c_k \cdot f_{k}^{t_j}; \)
7. if \( destHost = \text{NULL} \) then
   8. Initiate the VM \( vmt_{k,i} \) on \( destHost \); return \( vmt_{k,i} \);
else
   10. \( OH \leftarrow \text{all the off hosts in the data center}; \)
    11. foreach \( h_k \in OH \) do
       12. if \( f_{k}^{\max} > \text{cpu}_{j}^{t_j} \) & \( p_{k}^{\max} / f_{k}^{\max} < \minValue \) then
          13. \( destHost \leftarrow h_k; \minValue \leftarrow p_{k}^{\max} / f_{k}^{\max}; \)
14. Turn on host \( destHost \), then initiate VM \( vmt_{k,i} \) on it; return \( vmt_{k,i} \);

Function \( ScaleUpResources() \) in Algorithm 2 firstly chooses a VM template \( vmt \) that meets task \( t_j \)'s CPU frequency requirement (Line 1). To reduce the total energy power according to Principle 2, the host \( h_k \) that can accommodate the VM template \( vmt \) and has the minimal \( c_k \cdot f_{k}^{t_j} \) is chosen to deploy the new VM with template \( vmt \) (Lines 2-6). If the above mentioned step is infeasible, an off host \( h_k \) with the minimal energy-frequency ratio (i.e., \( p_{k}^{\max} / f_{k}^{\max} \)) that can contain \( vmt \) is turned on for the new VM (Lines 9-15).

Theorem 1: The time complexity of algorithm \( NEAT \) to schedule a workflow set \( W \) is \( O(|W| \cdot \max\{|T_i|\} \cdot (\max\{|E_i|\} + N_{vm} + N_{ho})) \), where \( |W| \) represents the workflow count in workflow set \( W \), \( \max\{|T_i|\} \) and \( \max\{|E_i|\} \) denote the maximum count of tasks and edges of a workflow in \( W \), \( N_{vm} \) is the number of VMs, \( N_{ho} \) denotes the host count.

Proof: It takes \( O(|T_i| \cdot |E_i|) \) to calculate the latest start time for all the tasks in workflow \( w_i \) (Line 5, Algorithm 1). Checking if a VM can execute the ready task costs \( O(N_{vm}) \) (Lines 13-16, Algorithm 1). In Algorithm 2, the complexity of selecting the candidate active hosts is \( O(N_{ho}) \) (Line 2, Algorithm 2). It takes \( O(N_{vm}) \) to search a destination host for a VM template (Lines 4-6, Algorithm 2), where \( N_{ho} \) is the count of active hosts. Looking for a turned-off host to contain a VM, the time complexity is \( O(N_{ho}) \) (Lines 10-13, Algorithm 2), where \( N_{ho} \) is the count of turned-off hosts. Thus, the time complexity of function \( ScaleUpResources() \), as shown in Algorithm 2, is \( O(N_{vm} + N_{ho}) \) to schedule a ready task to a VM. Moreover, it takes \( O(|T_i| \cdot |E_i| + |T_i| \cdot (N_{vm} + N_{ho})) \) to schedule all the tasks in workflow \( w_i \). In summary, the time complexity of scheduling a workflow set \( W \) is calculated as \( O(|W| \cdot \max\{|T_i|\} \cdot (\max\{|E_i|\} + N_{vm} + N_{ho})) \).

Definition 3: Idle VM: a VM is defined as idle if it has completed all the tasks mapped to it.

Since a wide range of end-users submit workflow applications to cloud platforms dynamically, their workload certainly fluctuate sharply over the time. When the local peak workloads over, there always appear some idle VMs and hosts in a cloud data center. To improve resource and energy efficiency, we devise a scale down strategy to reduce the number of used hosts by consolidating active VMs dynamically, which is outlined in Algorithm 3.

Algorithm 3 Function \( ScaleDownResources() \)

1. \( AH \leftarrow \text{all the active hosts in the data center}; \)
2. Sort \( AH \) by hosts' \( f_{k}^{\max} / p_{k}^{\max} \) in an increasing order;
3. foreach \( h_k \in AH \) do
   4. \( canH \leftarrow AH - h_k; \text{migPlan} \leftarrow \text{NULL}; \)
    5. foreach VM \( vmt_{k,i} \) on host \( h_k \) do
       6. if \( h_p \in canH \) then
          7. if \( h_p \) can contain \( vmt_{k,i} \) & \( c_p \cdot f_{p}^{2} < mV \) then
             8. \( selH \leftarrow h_p; \text{migPlan} \leftarrow \text{migPlan} \cup \{vmt_{k,i}, \text{selectedHost}\}; \)
          else
             10. \( \text{migPlan} \leftarrow \text{migPlan} \cup \{vmt_{k,i}, \text{selectedHost}\}; \)
             11. \( \text{migPlan} \leftarrow \text{NULL}; \text{break}; \)
        if \( \text{Tag} = \text{TRUE} \) then
           12. \( \text{migPlan} \leftarrow \text{migPlan} \cup \{vmt_{k,i}, \text{selectedHost}\}; \)
        else
           13. \( \text{migPlan} \leftarrow \text{NULL}; \text{break}; \)
    14. Execute the migration plan \( \text{migPlan} \);
    15. Turn off host \( h_k \), and remove it from \( AH \);

If the system workload becomes light and there exist some VMs that are idle. These idle VMs will be deleted, and idle hosts will also be shut down if all the VMs on them are deleted. Then, the function \( ScaleDownResources() \), as shown in Algorithm 3, will be called to reduce the count of used
hosts by consolidating the VMs and turning off the idle hosts. Firstly, all the active hosts are sorted by their $f_k^{max}/p_k^{max}$ in an increasing order (Line 2). After that, this function starts from the active hosts with lower $f_k^{max}/p_k^{max}$ (Line 3), and then checks if the VMs on the selected host can be migrated to other hosts (Lines 5-13). Moreover, to improve system’s energy efficiency, this function strives to maintain the balance of $c_p \cdot f_k^2$ of all the active hosts in $canH$ (Lines 8-9). Notably, if there exist some virtual machines on the selected host cannot find any target hosts, the plan of migrating other VMs for the selected host will be given up (Line 13). Otherwise, migrate all the VMs on selected host to corresponding target hosts, and then turn off the selected host after migrations (Lines 14-16).

Theorem 2: Function $ScaleDownResource()$ takes $O(N_a^2)$ to scale down the resources, where $N_a$ is the active host count.

Proof: It takes $O(N_a \log(N_a))$ to sort active hosts (Line 2, Algorithm 3). Reallocation of all the available VMs takes $O(N_a |VM_a| N_a)$ (Lines 3-13, Algorithm 3), where $|VM_a|$ denotes the count of VMs on host $h_k$. Since the $|VM|$ is not a large integer, $O(N_a |VM| N_a) = O(N_a^2)$. In conclusion, the complexity of Algorithm 3 is $O(N_a \log(N_a)) + O(N_a^2) = O(N_a^2)$.

\section*{V. EXPERIMENTS}

In this section, we conduct comparative experiments to verify the superiority of algorithm \textit{NEAT}. These comparing algorithms are: \textit{EASA} [23], \textit{HEFT} [14], and \textit{ESFS} [24].

\textit{EASA:} it models a workflow as a set of precedence constrained virtual machines (VMs), and each VM represents a workflow task. Algorithm \textit{EASA} firstly sorts all the VMs in a decreasing order based on their distances to the exit VM, then schedules VMs to hosts so as to optimize both makespan and host utilization, after that \textit{EASA} tries to relocate VMs on low-utilized active hosts to high-utilized active hosts.

\textit{HEFT:} this algorithm firstly assigns priority to each task according to its maximal length to the exit task. Then, the task with higher priority will be preferentially scheduled to a host with minimal energy consumption.

\textit{ESFS:} this algorithm first utilizes algorithm \textit{HEFT} to build an initial schedule of workflow tasks, and then gradually adjusts hosts’ CPU frequency for each workflow task with the goal of reducing energy consumption while guaranteeing the deadline constraint of a workflow.

In addition, the energy consumption (EC) as formula (8), resource utilization (RU) as formula (9) and the count of used hosts, are chosen to compare \textit{NEAT} with these three algorithms.

\section*{A. EXPERIMENTAL SETUP}

We choose ten types of real hosts\footnote{http://www.spec.org/power_ssj2008/results/} to simulate the host configurations in a cloud data center, and assume the count of hosts with each type to 1000. The main parameters of these hosts are shown in Table 1. The parameters Max Fre., $p_k^{idle}$, and $p_k^{max}$ refer to the variables $f_k^{max}$, $s_k \cdot p_k^{max}$, and $p_k^{max}$ in formulas (1) and (8), respectively. Ten types of VM templates are used in this experiment, and the CPU frequency requirements of these templates range from 200 to 2000 with the step size of 200. The network bandwidth among VMs is set to 1.0 Gbps. In addition, the time delays of turning on a host and creating a new VM are set to one minute and half a minute.

Four kinds of widely-used workflows (i.e., CyberShake, LIGO, SIPHT, and Montage) are selected, and three different sizes of each workflow: type one (about 30 tasks), type two (about 50 tasks) and type three (about 100 tasks), are used in this experiment, and the CPU frequency requirements of these templates range from 200 to 2000 with the step size of 200. The network bandwidth among VMs is set to 1.0 Gbps. In addition, the time delays of turning on a host and creating a new VM are set to one minute and half a minute.

The arrival of these workflows is assumed to be subject to the Poisson distribution with parameter $\lambda$. In addition, the symbol, \textit{arrival rate} $= 1/\lambda$, is utilized to control the average number of new arrival workflows per second.

In addition, the parameter \textit{deadlineBase} is used to reflect workflows’ timing requirements, and the workflow deadline becomes looser with the increase of \textit{deadlineBase}.

For each group of experimental setting, each algorithm is tested 30 times independently. Then, the average, minimum and maximum of experimental results are presented.

\section*{B. PERFORMANCE IMPACT OF WORKFLOW DEADLINE}

This group of experiments focus on studying the impact of workflow deadline on the performance of the four algorithms.
hosts. In addition, NEAT tasks, and turns on hosts in a random way when increasing consideration of the heterogeneity of hosts when scheduling workflow consumes the most energy. The reason is that it does not consider a single task cannot fully utilize the host resources, it always consumes the most energy. The reason is that it does not consider the heterogeneity of hosts when scheduling workflow tasks, and turns on hosts in a random way when increasing hosts. In addition, NEAT consumes less energy than EASA, HEFT and ESFS on average by 45.64%, 39.23% and 35.98%, respectively. This experimental result indicates that NEAT is efficient with energy conservation for executing workflows in virtualized cloud data centers.

The resource utilization (RU) of active hosts is shown in Fig. 3(b). With the increase of deadlineBase, the RUs of algorithms NEAT, EASA, HEFT and ESFS fluctuate slightly around 69.32%, 53.14%, 37.06% and 29.08%, respectively. The interpretation for such high RU of the proposed NEAT is that it schedules workflow tasks coming from different workflows in a hybrid way to reduce the idle time slots on resources, and also adjusts the available resources dynamically.

According to Fig. 3(a) and Fig. 3(b), we find an unusual phenomenon that EASA produces much higher resource utilization than HEFT and ESFS, but EASA still consumes much more energy than them, which seems contrary to the common sense that the higher resource utilization means the lower energy consumption. This can be explained by the fact that EASA does not consider the host heterogeneity in cloud data centers, and does not turn on/off the hosts with high/low energy efficiency when scaling up/down computing resources.

To compare the performance of the four algorithms with respect to the number of used hosts, we fix the workflow count, arrival rate, and deadlineBase as 10000, 0.5 and 1.2 to observe the number distribution of the used hosts for the four algorithms over time, which is illustrated in Fig. 3(c). This experimental result shows that the number of used hosts of NEAT is far lower than that of the other three algorithms, especially about 50% lower than that ESFS. The explanation for this result can be referred to that of Fig. 3(a).

C. PERFORMANCE IMPACT OF ARRIVAL RATE

In this group of experiments, the arrival rate is increased from 0.1 to 0.5 with an increment of 0.1. The workflow count and deadlineBase are fixed to 10,000 and 1.8, respectively. Fig. 4 illustrates the comparing results of the four algorithms NEAT, EASA, HEFT and ESFS.

Fig. 4(a) reveals that the EC of the four tested algorithms increase slightly with the increase of arrival rate. The explanation for the phenomenon is that the higher arrival rate of workflows means the cloud data center needs to execute more workflows in each unit time, which requires more hosts, resulting in the use of more hosts with low energy efficiency. Besides, the advantage of the proposed NEAT is consistent with that in Fig. 3(a). We can also observe that NEAT consumes the least energy consumption, whereas EASA is the most energy consumption. This dues to that EASA does not consider the heterogeneity of hosts in the data center. In addition, NEAT consumes less energy than EASA, HEFT and ESFS on average by 45.60%, 36.83% and 34.00%, respectively. The above result is consistent with that in Fig. 3(a).

Fig. 4(b) shows that the RU of NEAT decreases from 74.79% to 70.16% with arrival rate increases from 0.1 to 0.5. This can be explained by that with the increase of arrival rate, a cloud data center has to start more hosts, which also leads to higher heterogeneity of used hosts, thereby increasing the difficulties in scaling up/down computing resources. On average, NEAT outperforms EASA, HEFT and ESFS by 26.27%, 49.95% and 59.78%, respectively. The reasons are similar to Fig. 3(b).

Fig. 4(c) provides the count distribution of hosts used over time for algorithms NEAT, EASA, HEFT and ESFS, when the workflow count, arrival rate, and deadlineBase are set to be 10000, 0.5 and 1.8, respectively. Fig. 4(c) shows that NEAT
turns on less hosts than \textit{EASA}, \textit{HEFT} and \textit{ESFS} roughly by 50\%, 32\% and 30\%, respectively. This result demonstrates the advantage of the proposed \textit{NEAT} in reducing the host count. Besides, the advantage of the proposed \textit{NEAT} is consistent with that in Fig. 3(a). It can be found that the count of used hosts for the four algorithms are elastic according to the request demand. Comparing with other three algorithms, algorithm \textit{NEAT} boots up the least hosts all the time. Fig. 4(c) also shows that \textit{NEAT} turns on less hosts than \textit{EASA}, \textit{HEFT} and \textit{ESFS} roughly by 50\%, 32\% and 30\%, respectively.

D. PERFORMANCE IMPACT OF CCR

The parameter \textit{CCR} of a workflow refers to the ratio of its computational time and its communication time [23]. In this subsection, we attempt to study the impact of \textit{CCR} on the four algorithms, and vary the \textit{CCR} 0.5 to 5.5. The experimental results are shown in Fig. 5.

Fig. 5(a) shows that the \textit{EC} of \textit{EASA}, \textit{HEFT} and \textit{ESFS} increases when increasing the \textit{CCR}. This result can be contributed to the fact that larger \textit{CCR} means higher communication time among workflow tasks, which further results in longer idle time slots on VMs. In addition, these three algorithms have not employ effective strategies to compress the idle time slots, resulting in more energy consumption. It is worth noting that the \textit{EC} of \textit{NEAT} basically keeps unchanged with the variance of the \textit{CCR}. This can be contributed to the strategy in \textit{NEAT} that it deploys tasks coming from different workflows to run on the same VMs alternately, which is helpful of decreasing the idle time slots.

Fig. 5(b) gives the first impression that \textit{NEAT} is far superior to \textit{EASA}, \textit{HEFT} and \textit{ESFS} in terms of \textit{RU}. This result again reveals the advantage of the proposed \textit{NEAT} in improving \textit{RU} while meeting workflows’ timing requirements. Fig. 5(b) also depicts that when \textit{CCR} increases, the resource utilizations of algorithm \textit{EASA}, \textit{HEFT} and \textit{ESFS} decrease correspondingly. This is because with the increase of \textit{CCR}, the idle time slots on computing resources become larger, thus the resource utilization shows a descending trend. However, the resource utilization of \textit{NEAT} shows a slight upward trend with the increase of \textit{CCR}. This phenomenon demonstrates that the strategy, that scheduling tasks from different workflows to the same VMs, of \textit{NEAT} is effective with respect to resource utilization.

Fig. 5(c) depicts the change of the count of used hosts with time, when the workflow count, arrival rate, \textit{deadlineBase} and \textit{CCR} are set to be 10000, 0.5, 1.8 and 0.5, respectively. It is evident that the performance of the proposed \textit{NEAT} remains consistently superior to that of its competitors with respect to the count of hosts used. Fig. 5(c) clearly shows that the hosts used of these four algorithms can be roughly classified into three categories: the count of hosts used by \textit{NEAT} fluctuates between 3000 and 4000, and that of \textit{EASA} and \textit{HEFT} are around 5200, while \textit{ESFS} initiates up to 7000 hosts. This result once again demonstrates our proposed \textit{NEAT} can effectively reduce the count of hosts used by cloud data centers. This experimental result is consistent with that in 3(c), and the explanation for it is similar to that for Fig. 3(c).

VI. CONCLUSION AND FUTURE WORK

The focus of this paper is on reducing energy consumption for cloud data centers while guaranteeing the deadlines of dynamic workflows. We first propose a novel scheduling architecture to translate dynamic scheduling into static scheduling, then derive two principles to address the workflow scheduling issues. After that, we propose the algorithm \textit{NEAT} to improve the energy and resource efficiency for cloud platforms while meeting timing requirements of workflows. Finally, three groups of experiments on real-world workflow traces are conducted to validate the performance of the proposed \textit{NEAT}. Compared with three existing algorithms, \textit{NEAT}
shows better performance in terms of energy consumption, resource utilization and the count of used hosts.

When executing data intensive applications in industrial clouds, the network congestion is inevitable, which can also evidently impact the platforms’ performance. Thus, we intend to develop novel contention-aware scheduling approaches for data-intensive applications in the future. Besides, it is reported that task execution times and data transmission times in cloud environments appear significant uncertainties, which may severely shake the performance of the algorithms. Thus another future study is to extend the NEAT to handle the uncertainties of task execution times and data transmission times.

REFERENCES

[1] F. Tao, Y. Cheng, L. D. Xu, L. Zhang, and B. H. Li, “CCIoT-CMfg: Cloud computing and Internet of Things-based cloud manufacturing service system,” *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1435–1442, May 2014.

[2] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, “Energy-aware scheduling of MapReduce jobs for big data applications,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2720–2733, Oct. 2015.

[3] Y. Wu, X. Tan, L. Qian, D.-K. Tsang, W.-Z. Song, and L. Yu, “Optimal pricing and energy scheduling for hybrid energy trading market in future smart grid,” *IEEE Trans. Ind. Informat.*, vol. 11, no. 6, pp. 1585–1596, Dec. 2015.

[4] M. H. Y. Moghaddam, M. Moghaddassian, and A. Leon-Garcia, “Autonomous two-tier cloud-based demand side management approach with microgrid,” *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1109–1120, Jun. 2017.

[5] X. Li, P. Garraghan, X. Jiang, Z. Wu, and J. Xu, “Holistic virtual machine scheduling in cloud datacenters toward minimizing total energy,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1317–1331, Jun. 2018.

[6] H. Yan, H. Wang, X. Li, Y. Wang, D. Li, Y. Zhang, Y. Xie, Z. Liu, W. Cao, and F. Yu, “Cost-efficient consolidating service for Aliyun’s cloud-scale computing,” *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 117–130, Jan/Feb. 2019.

[7] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, “Real-time tasks oriented energy-aware scheduling in virtualized clouds,” *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Apr./Jun. 2014.

[8] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vihi, “Characterizing and profiling scientific workflows,” *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[9] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, “Uncertainty-aware online scheduling for real-time workflows in cloud service environment,” *IEEE Trans. Services Comput.*, to be published.

[10] G. Xie, J. Jiang, Y. Liu, R. Li, and K. Li, “Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems,” *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1068–1078, Jun. 2017.

[11] M. Mezmnz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuytten, “A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems,” *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1497–1508, 2011.

[12] G. Yao, Y. Ding, Y. Jin, and K. Hao, “Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system,” *Soft Comput.*, vol. 21, no. 15, pp. 4309–4322, 2016.

[13] L. Zhang, K. Li, C. Li, and K. Li, “Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems,” *Inf. Sci.*, vol. 379, pp. 241–256, Feb. 2017.

[14] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[15] J. J. Durillo, V. Nae, and R. Prodan, “Multi-objective energy-efficient workflow scheduling using list-based heuristics,” *Future Gener. Comput. Syst.*, vol. 36, pp. 221–236, Jul. 2014.

[16] F. Tao, Y. J. LaiLi, L. D. Xu, and L. Zhang, “FC-PACO-RM: A parallel method for service composition optimization selection in cloud manufacturing system,” *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2023–2033, Nov. 2013.

[17] A. Schranzhofer, J.-J. Chen, and L. Thiele, “Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms,” *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 692–707, Nov. 2010.

[18] S. Abrihami, M. Naghibzadeh, and D. H. J. Epema, “Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds,” *Future Generat. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.

[19] P. Xiao, Z.-G. Hu, and Y.-P. Zhang, “An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters,” *J. Comput. Sci. Technol.*, vol. 28, no. 6, pp. 948–961, 2013.

[20] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, “EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters,” *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

[21] H. Chen, X. Zhu, D. Qiu, H. Guo, L. T. Yang, and P. Lu, “EONS: Minimizing energy consumption for executing real-time workflows in virtualized cloud data centers,” in *Proc. 45th Int. Conf. Parallel Process. Workshops (ICPPW)*, Aug. 2016, pp. 385–392.

[22] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, “Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2674–2688, Sep. 2017.

[23] V. Ebrahimirad, M. Goudarzi, and A. Rajabi, “Energy-aware scheduling for precedence-constrained parallel virtual machines in virtualized data centers,” *J. Grid Comput.*, vol. 13, no. 2, pp. 233–253, 2015.

[24] I. Pietri and R. Sakellarios, “Energy-aware workflow scheduling using frequency scaling,” in *Proc. 43rd Int. Conf. Parallel Process. Workshops (ICPPW)*, Sep. 2014, pp. 104–113.