Int j simul model 19 (2020) 3, 505-516
ISSN 1726-4529 Original scientific paper

ENERGY-SAVING CLOUD WORKFLOW SCHEDULING BASED ON OPTIMISTIC COST TABLE

Lin, T.[*,**]; Wu, P.[*,***,#]; Gao, F. M.[**] & Wu, T. S.[****]
[*] School of Automation, Chongqing University, Chongqing 400044, China
[**] Chongqing College of Electronic Engineering, Chongqing 401331, China
[***] Chongqing Chuanyi Automation Co., Ltd., Chongqing 401121, China
[****] College of Computer Science, Chongqing University, Chongqing 400044, China
E-Mail: lintaoemail@126.com, pwu@cqu.edu.cn, gfmemail@126.com, 723412117@qq.com
([#] Corresponding author)

Abstract
In recent years, intelligent flow sensors have been applied to many fields. Cloud operation is a design method to further improve the intelligence of such sensors. However, the cloud workflows of intelligent flow sensors consume too much energy, making it imperative to schedule cloud workflows. With the growing awareness of energy conservation, it is a hot topic to design an energy-efficient workflow scheduling algorithm. Therefore, this paper puts forward the predict minimum energy consumption (PMEC) algorithm, a cloud workflow scheduling algorithm that strikes a balance between energy consumption and execution time. Firstly, the optimistic cost table (OCT) was adopted to rank the tasks by priority. Then, the resources, i.e. virtual machines, were assigned statically to the tasks, in the light of task priority and energy consumption. After that, the workflow was scheduled according to the assignments. Simulation results show that the PMEC is much more energy efficient than traditional list-based scheduling algorithms.
(Received in April 2020, accepted in June 2020. This paper was with the authors 1 month for 1 revision.)

Key Words: Energy Consumption, Workflows, Scheduling Algorithm, Sensors

1. INTRODUCTION

Intelligent flow sensors are capable of processing massive flow information in real time. In recent years, intelligent flow sensors have been applied to more and more fields. Many design methods have been adopted to further improve the intelligence, namely, group measurement, networking, and cloud operation. However, high energy consumption of the cloud workflows of such sensors must be controlled in the design of the flow measurement system.

Scientific workflows [1] can fully utilize the abundant resources on cloud sensors. For cloud sensors, the resources come from datacentres distributed in various places. The datacentres are heavy consumers of energy, owing to the largescale management and service provision. In 2011, cloud sensor datacentres consumed 3 % of the electricity of the entire grid in the US, incurring an electricity cost far greater than server cost. Thus, the provision of diverse computing services is realized at the cost of energy efficiency and the environment. The computing resources should be reasonably utilized to reduce energy consumption.

The cloud sensor service platform can provide platform as a service (PaaS), software as a service (SaaS), and infrastructure as a service (IaaS) [2]. Cloud workflows refer to the scientific workflows based on the IaaS of cloud sensors, which is provided as virtual machines. The optimization of cloud workflows, i.e. cloud workflow scheduling, aims to determine the execution sequence of tasks and the optimal mapping to virtual machines [3, 4].

The cloud workflow scheduling algorithms can be evaluated on physical platforms. But this evaluation method is very inconvenient. Therefore, most scholars choose to simulate cloud workflow scheduling algorithms [5]. This paper proposes the predict minimum energy consumption (PMEC) algorithm, an energy-saving cloud workflow scheduling algorithm for

intelligent flow sensors based on optimistic cost table (OCT), and simulates the proposed algorithm on WorkflowSim. Extended from CloudSim, this platform not only dynamically provides cloud resources, but also supports the simulation of scientific workflows.

2. LITERATURE REVIEW

Cloud workflow scheduling can be realized by heuristic or metaheuristic scheduling algorithm [5, 6]. Many scholars have proposed heuristic and metaheuristic scheduling algorithms to minimize the makespan. For example, Liu and Dessouky [7] created a mixed integer programming model for joint passenger and freight train scheduling problem and solved the joint scheduling problem with a heuristic two-step decomposition algorithm. Borisovsky et al. [8] presented a scheduling strategy for a large-scale industrial plant operating in a continuous mode, which couples a decomposition approach, a genetic algorithm (GA), and a constructive mixed-integer linear programming (MILP)-based heuristic algorithm, which greatly outperforms the pure decomposition algorithm in solution quality and solving time. Abazari et al. [9] developed a systematic method that considers both the security demand for tasks and the interaction between secure tasks in the cloud and solved the scheduling problem with a heuristic algorithm. Hosseinioun et al. [10] designed an improved metaheuristic scheduling algorithm based on the invasive weed optimization and culture (IWO-CA) evolutionary algorithm, which optimizes the dynamic scheduling of cloud computing tasks. Alzidani and Dao [11] presented a new hybrid approach to optimize the sequence of parts in each cell and the exceptional elements, thereby minimizing the makespan.

Besides heuristic and metaheuristic algorithms, artificial intelligence (AI) algorithms have been adopted for multi-objective scheduling of workflows. For instance, Singh et al. [12] designed a workflow scheduling algorithm called PPDPS to solve the supply chain scheduling problem. Entezari-Maleki et al. [13] put forward a genetic-based scheduling algorithm to appropriately dispatch programs of a workflow application to the resources distributed within a grid computing environment. Ambursa et al. [14] developed a multi- quality-of-service (QoS) workflow scheduling strategy based on particle swarm optimization (PSO) and a look-ahead heuristic (LAPSO); under this strategy, the best scheduling solutions are selected based on the proposed constraint-handling strategy to improve the quality of the best solutions. Chirkin et al. [15] proposed a solution that considers the complexity and the stochasticity of the workflow components and their runtime, which solves various problems from the task level to the workflow level. Sathish and Reddy [16] combined the GA and ant colony optimization (ACO) into a hybrid algorithm that satisfies all the defined constraints and user parameters. Kintsakis et al. [17] leveraged machine learning (ML) to predict workflow task runtime and the probability of failure of task assignments to execution sites, and proposed to make scheduling decisions with the ML model as part of a workflow management system (WMS); the WMS can significantly improve its scheduling performance by independently learning over time, without external intervention or reliance on any specific heuristic or optimization technique. Shirvani [18] presented a hybrid discrete particle swarm optimization (HDPSO) algorithm and proved that the HDPSO excels over the existing metaheuristics in scheduling length ratio (SLR), speedup, and efficiency by 10.47 %, 14.48 %, and 3 %, respectively. To rationalize resource allocation and task scheduling of cloud computing, Dubey et al. [19] created a new scheduling algorithm called the ideal distribution algorithm (IDA), which can enhance the system ability under deadline constraints and improve the monetary cost.

Time and energy consumption are two important dynamic parameters in workflow scheduling. Rezaeian et al. [20] pointed out that the current workflow technique cannot meet user demand in cloud applications, and developed a metaheuristic optimization algorithm, i.e. the PSO strategy, in view of the elasticity and heterogeneity of computing resources. Deldari et

al. [21] abstracted the time-cost optimization of directed acrylic graph (DAG) applications into the trade-off between discrete time and cost, and thus introduced discrete intelligent algorithms to the optimization process. Stavrinides and Karatza [22] invented an energy-efficient, QoS-aware, and cost-effective scheduling strategy for real-time workflow applications in cloud computing systems; this approach utilizes per-core dynamic voltage and frequency scaling (DVFS) on the underlying heterogeneous multi-core processors, plus approximate computations. Barika et al. [23] proposed a pluggable dynamic scheduling technique that accepts user-defined algorithms to handle stream workflow runtime changes, and also presents three different plug-in algorithms and methods to enable auto-scaling of this workflow in a multi-cloud environment; experimental results manifest that the proposed technique is more efficient than baseline and dynamic fair-share techniques in changing runtime changes. Stavrinides et al. [24] investigated the workflow scheduling problem in largescale distributed systems and provided a scheduling approach that optimizes the system performance under tardiness bounds and data locality. Arabnejad and Barbosa [25] put forward a scheduling algorithm that manages concurrent workflows in a dynamic environment, in which jobs are submitted by users at any moment in time, on shared heterogeneous resources, and constrained to a specified budget and deadline for each job; the scheduling algorithm can increase the profit achieved by the provider by defining task priorities based on the budget available for each job.

3. PROBLEM DESCRIPTION

In the cloud environment, the cloud service provider offers computing services under the pay-as-you-go model. The user rents the server from the cloud service provider, and then submits the scientific workflows to be calculated to the datacentre for calculation. The datacentre has multiple computing nodes and storage nodes. When multiple virtual machines in the datacentre are utilized, a period will be consumed by the data transmission between virtual machines. Once the computing service completes, the cloud service provider will charge a fee according to the rental time and the type of server rented by the user. Taking Amazon Elastic Compute Cloud (Amazon EC2) for example, the rental time is counted in hours. If the rental time is less than one hour, it will be treated as one hour during the charging. Fig. 1 abstracts a tire manufacturing process into cloud workflows.

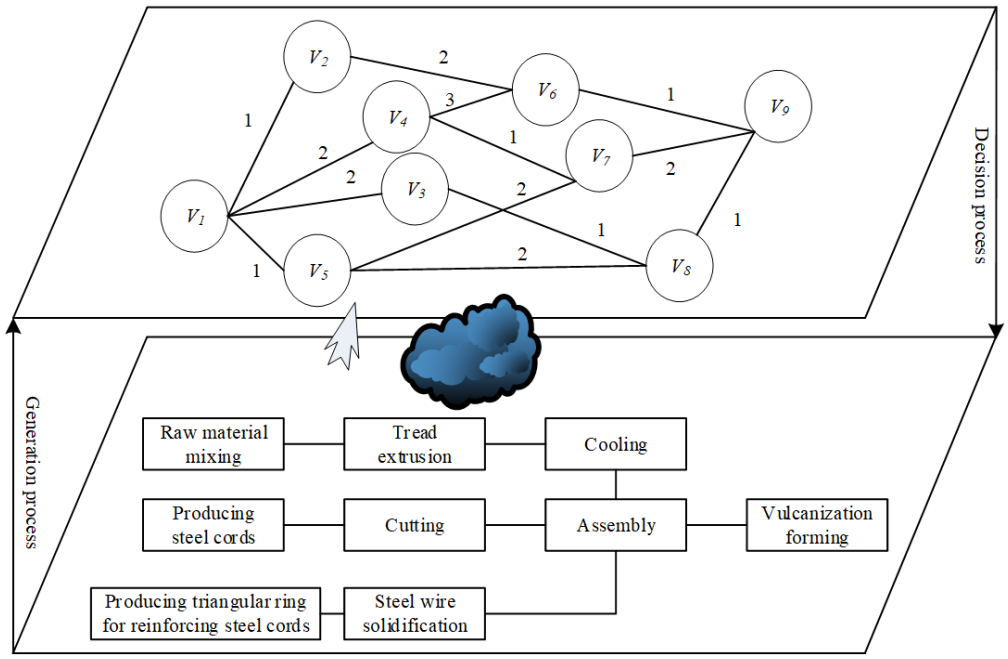

Figure 1: The abstraction of the tire manufacturing process.

3.1 Workflow application model

Task scheduling can be divided into static scheduling and dynamic scheduling. Without knowing all the needs of the entire workflow, dynamic scheduling cannot easily optimize the workflow schedule globally. By contrast, static scheduling, knowing the task dependences and optimization objectives, can maximize the scheduling effect, because the resource allocation plan has been generated before the execution of the workflow.

This paper focuses on the static scheduling of a single workflow in the cloud sensor environment. The workflow can be simplified as a DAG $G = (V, E)$, where V is the set of nodes; E is the set of the directed edges between the nodes. Each node $v_i \in V$ represents a task in the workflow; each directed edge $e(i, j) \in E$ represents the dependence between tasks n_i and n_j. Since n_i must be completed before n_j, n_i is called the predecessor of n_j, and n_j is called the successor of n_i.

Each task v can be further expressed as a 4-tuple $v = (id, len, fin, fout)$, where id is the serial number of the task; len is the number of instructions to be executed to complete the task; fin and $fout$ are the set of input files and the set of output files to complete the task, respectively. Each file f can be expressed as a 2-tuple $f = (id, len)$, where id is the serial number of the file; len is the size of the file. The precedence of n_i over n_j can be expressed as $n_i.fout \cap n_j.fin \neq \phi$. Finally, the number of tasks can be expressed as |V|.

3.2 Workflow resource model

This paper adopts a cloud sensor resource model similar to Amazon EC2. In the cloud environment, the host was divided into several virtual machines with different CPUs, memories, hard-disk spaces, and input/outputs (I/Os). Each virtual machine was treated as the smallest instance of computing resources. It is assumed a virtual machine can only process one task at a time. The virtual machine response time to task request is so small as to be negligible. For simplicity, our model assumes that there is no delay in the response.

The host can be expressed as a 4-tuple $h = (id, mips, vml, ld)$, where id is the serial number of the host; $mips$ is the number of instructions that the host can execute per second; vml is the list of virtual machines running on the host; ld is the load of the host.

Each virtual machine can also be expressed as a 4-tuple $vm = (id, hid, mips, stat)$, where id is the serial number of the virtual machine; $mips$ is the number of instructions that the virtual machine can execute per second; $stat$ is the state of the virtual machine (the virtual machine is occupied if $stat = 1$, and available if $stat = 0$). Let VM be the set of all virtual machines. Then, the total number of virtual machines can be expressed as |VM|.

3.3 Workflow energy consumption model

The running of the virtual machines on the host needs to consume some energy. The energy consumption of each type of host are available on the website of Standard Performance Evaluation Corporation (SPEC). The energy consumption of a specific load can be obtained through linear interpolation. If the host load $h.ld$ remains constant in period t, the energy consumption $E(h, t)$ of the host h in period t can be calculated by:

$$\begin{cases} E(h,t) = \int_{ST}^{ET} e(h.ld) \\ h.ld = \dfrac{\sum_{vm \in h.vml} vm.mips}{h.mips} \end{cases} \tag{1}$$

where, $t = [ST, ET]$ (ST and ET are the start and end times of the period that the host load remains constant, respectively); $e(h.ld)$ is the linearly interpolated energy consumption of the host at the load of $h.ld$.

Fig. 2 shows the three kinds of changes of the host load when a new virtual machine vm_3 is activated for scheduling.

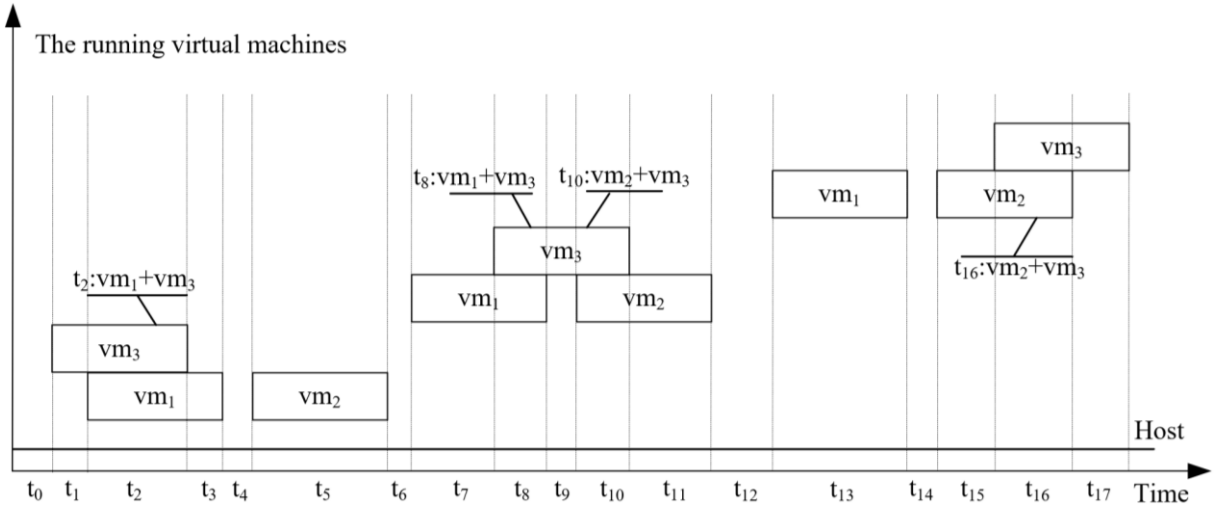

Figure 2: The variations of host load after adding virtual machine vm_3.

3.4 Relevant definitions of the workflow

Definition 1: the execution time $w_{i,j}$ of task v_i on virtual machine vm_j. The value of $w_{i,j}$ can be calculated by:

$$w_{i,j} = \frac{v_i.len}{vm_i.mips} \tag{2}$$

Definition 2: the set $pred(v_i)$ of immediately predecessors of task v_i in the DAG. Any task without any predecessor is called an entry task v_{entry}. To simplify the workflow scheduling, if there are multiple entry tasks, a virtual predecessor is added before every entry task to serve as the new entry task.

Definition 3: the set $succ(v_i)$ of immediately successors of task v_i in the DAG. Any task without any successor is called an exit task v_{exit}. If there are multiple exit tasks, a virtual successor is added after every exit task to serve as the new exit task. The completion time of the virtual exit task is taken as the completion time of the entire workflow.

Definition 4: the mean transmission time $\overline{c_{i,j}}$ between tasks n_i and n_j. The value of $\overline{c_{i,j}}$ can be calculated by:

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{B}} \tag{3}$$

where, $\overline{L}$ is the mean response time of a virtual machine to task request ($\overline{L} = 0$ because there is no delay in the response); $\mathrm{data_{i,j}} = \sum_{\mathrm{f}\in(\mathrm{n_i.fout}\cap\mathrm{n_j.fin})} \mathrm{f.len}$ is the amount of data to be transmitted from task v_i to task v_j; $\overline{B}$ is the mean bandwidth. If tasks v_i and v_j are executed on the same host, $\overline{c_{i,j}} = 0$.

Definition 5: the earliest start time $EST(v_i, vm_j)$ of task v_i on virtual machine vm_j. The value of $EST(v_i, vm_j)$ can be calculated by:

$$EST(v_i, vm_j) = \max\left\{T_{avl}(vm_j), \max_{v_p \in pred(v_i)}\{AFT(v_p) + c_{p,i}\}\right\} \tag{4}$$

where, $T_{avl}(vm_j)$ is the earliest available time of virtual machine vm_j; $AFT(v_p)$ is the actual completion time of task v_p; $c_{p,i}$ is the arrival time of the data needed for task v_i at virtual machine vm_j. Note that $\forall vm, EST(v_{entry}, vm) = 0$.

Definition 6: the earliest completion time of task v_i on virtual machine vm_j. The value of $EFT(v_i, vm_j)$ can be calculated by:

$$EFT(v_i, vm_j) = EST(v_i, vm_j) + w_{i,j} \tag{5}$$

3.5 Workflow generation algorithm (WFGA)

Any task submitted to the cloud datacentre for calculation must be divided into different workflows. The WFGA can be formally described as follows:

Algorithm 1. The WFGA

Inputs: The set of task T; the set of task dependences E
Output: Workflow diagram G
1: Create a virtual entry task and add it to G
2: Add the actual entry task v_0 to G as a subtask of the virtual entry task
3: Delete the actual entry task v_0 from T
4: while T != NULL
5: if (task v in T && task v's predecessor v_p in G)
6: Delete task v from T
7: Add task v to G as a subtask of task v_p
8: end while
9: return G

4. PMEC ALGORITHM

This paper proposes a list-based scheduling algorithm called the PMEC, with the aim to minimize the energy consumption of workflow scheduling. The scheduling algorithm can be divided into two phases: task prioritization and resource allocation.

4.1 Task prioritization

The heterogeneous earliest finish time (HEFT) is the most classic list-based heuristic workflow scheduling algorithm. The HEFT prioritizes tasks by calculating the longest path $rank_u(v_i)$ from task v_i to the exit task v_{exit}. Nevertheless, this method does not consider the successors, but only the current task. Then, the priorities of some successors might be suboptimal. To solve the problem, it is necessary to consider the impact of the periodization of the current task on its subtask. For this purpose, a new concept called the OCT was introduced. The OCT can be expressed as a |V|×|VM| matrix.

Definition 7: the maximum $OCT(v_i, vm_j)$ of the shortest distances from the subtask of task v_i to the exit task v_{exit}, after task v_i has been executed on virtual machine vm_j. The value of $OCT(v_i, vm_j)$ can be calculated by:

$$OCT(v_i, vm_j) = \max_{v_c \in succ(v_i)} [\min_{vm_k \in VM} \{OCT(v_c, vm_k) + w_{c,k} + \overline{c_{i,c}}\}] \tag{6}$$

where, $\overline{c_{i,c}}$ is the mean transmission time $\overline{c_{i,c}} = 0$, if task v_i is executed on virtual machine vm_k; $OCT(v_i, vm_j)$ is the optimal processing time of the subtask of task v_i. Since the subtask could be handled on virtual machines, the total processing time can be shortened by reducing the transmission time. The *OCT* is calculated by recursion. For the exit task v_{exit} = 0, $OCT(v_{exit}, vm_j) = 0$, $vm_j \in$ VM.

To define the priority of each task, the mean *OCT* of each task can be defined as:

$$rank_{oct(v_i)} = \frac{\sum_{j=1}^{|\mathrm{VM}|} OCT(v_i, vm_j)}{|\mathrm{VM}|} \tag{7}$$

4.2 Virtual machine selection

To assign the suitable virtual machine to each task, the sum $OE(v_i, vm_j)$ of the *OCT* and the energy consumption increment of the host after task v_i has been executed on virtual machine vm_j can be calculated by:

$$\begin{cases} OE(v_i, vm_j) = \Delta E + OCT(v_i, vm_j) \\ \Delta E(h,t) = \int_{ST_{v_i,vm_j}}^{ET_{v_i,vm_j}} e(h.ld_0 + vm_j.mips) - e(h.ld_0) \end{cases} \tag{8}$$

where, $t = [ST_{v_i,vm_j}, ET_{v_i,vm_j}]$ (ST_{v_i,vm_j} and ET_{v_i,vm_j} are the start and end times of task v_i being executed on virtual machine vm_j, respectively); the integral part is the energy consumption increment of the host after the increase of host load.

Besides considering the energy consumption of the host, the *OCT* was added as the regular term to prevent the workflow execution from being too long. Due to the addition, the execution time of the current task not necessarily the shortest. However, the successors can be executed in shorter time.

4.3 Details of the PMEC algorithm

Considering the priority of each task, the proposed PMEC algorithm selects a virtual machine with relatively low energy consumption to execute the task and controls the workflow execution time within the allowable range, reducing the energy consumption of the host. The PMEC algorithm can be formally described as follows:

Algorithm 2. PMEC algorithm.

1: Calculate the *OCT* list and the priority of each task $rank_{oct}$
2: Create an empty list *list* and take the entry task v_{entry} as the initial task
3: while *list* is not empty do
4: $v_i \leftarrow$ the task with the highest priority $rank_{oct}$ in the *list*
5: for all virtual machines vm_j in |VM| do
6: $OE(v_i, vm_j) = \Delta E + OCT(v_i, vm_j)$
7: end for
8: Assign task v_i to the virtual machine vm_j that minimizes $OE(v_i, vm_j)$
9: Update *list*
10: end while

In Line 1, the algorithm calculates the *OCT* list and $rank_{oct}$. Then, an empty list *list* is created, and the entry task v_{entry} is placed on top of the list. Lines 3-9 are a while loop, in which the task with the highest $rank_{oct}$ is selected for scheduling in each iteration; then, the *OE* value of the task is calculated relative to each virtual machine. During virtual machine assignment, the objective is explained by the *OE* values: selecting the virtual machine with relatively low energy consumption to execute each task, while keeping the execution time within the tolerance. In Line 8, the virtual machine with the minimal $OE(v_i, vm_j)$ is selected to execute task v_i.

In terms of time complexity, the PMEC algorithm needs to calculate the *OCT* table, which requires $O(m \cdot (e+v))$. During the virtual machine assignment, the time complexity is $O(v^2 \cdot m)$. Thus, the total time complexity equals $O(m \cdot (e+v) + v^2 \cdot m))$. For a dense DAG, the number of

edges e is roughly v^2. Hence, the total time complexity is $O(v^2 \cdot m)$, where v=|V| is the number of tasks, and m=|VM| is the number of virtual machines.

5. NUMERICAL EXAMPLE

The scheduling and optimization process of the PMEC algorithm is explained with an example workflow in this section.

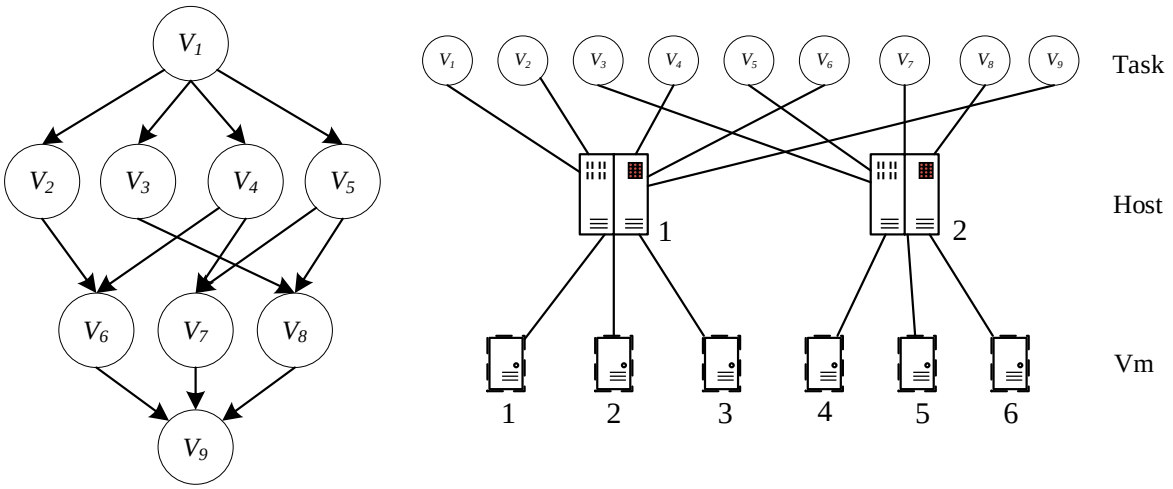

Figure 3: The example workflow.

The workflow to be scheduled is shown in Fig. 3, where the numbers next to the edges between tasks represent the task transmission time between hosts. Tables I and II list the execution time of each task on each virtual machine, and the transmission time of each task between hosts. The $rank_{OCT}$ value of each task was obtained by Eqs. (6) and (7) and recorded in Table III.

For simplicity, three models of virtual machines were considered. The load ratios of the three models to the host are 2: 3: 5, respectively. Table IV displays the energy consumption of the host at different load conditions.

According to the rank values, the tasks should be scheduled in the following sequence: $V_1 \rightarrow V_3 \rightarrow V_5 \rightarrow V_2 \rightarrow V_4 \rightarrow V_6 \rightarrow V_7 \rightarrow V_8 \rightarrow V_9$.

Table I: The execution time of each task on each virtual machine.

	$Host_1$			$Host_2$		
Task	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6
V_1	10	8	6	10	8	6
V_2	8	7	5	8	7	5
V_3	4	3	2	4	3	2
V_4	9	8	6	9	8	6
V_5	8	7	6	8	7	6
V_6	7	5	4	7	5	4
V_7	6	5	3	6	5	3
V_8	10	9	8	10	9	8
V_9	7	5	3	7	5	3

6. SIMULATION

Our simulation was carried out on WorkflowSim, an extension of CloudSim platform. The extended platform supports the simulation of workflows, allowing the addition of user-defined functions.

Table II: The transmission time of each task between hosts.

Task	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
V_1	0	1	2	2	1	-	-	-	-
V_2	1	0	-	-	-	2	-	-	-
V_3	2	-	0	-	-	-	-	1	-
V_4	2	-	-	0		3	2	-	-
V_5	1	-	-	-	0	-	1	2	-
V_6	-	2	-	3	-	0	-	-	1
V_7	-	-	-	2	1	-	0	-	2
V_8	-	-	1	-	2	-	-	0	1
V_9	-	-	-	-	-	1	2	1	0

Table III: The $rank_{OCT}$ value of each task.

Task	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
$rank_{OCT}$	17	7	11	7	11	3	3	3	0

Table IV: The energy consumption of the host at different load conditions.

Load ratio	20 %	30 %	50 %	70 %	80 %	100 %
Energy consumption	2	3	5	7	8	10

6.1 Simulation environment

The hardware of the simulation is a personal computer (PC) with an Intel® Core™ i5-7200U Processor (up to 2.5 GHz) and a memory of 8 Gb. Ten servers were deployed in the simulation environment. Referring to the configuration of Amazon EC2, the servers were divided into large, medium, and small categories, whose processing capacities are 2,500, 2,000, and 1,500 Mips, respectively.

6.2 Results analysis

The proposed PMEC algorithm was compared with two common algorithms, namely, HEFT, and dominance-based HEFT (DHFT), on several typical scientific workflows.

The CyberShake workflow, a workflow in seismic wave physics, is used by the Southern California Earthquake Center to characterize earthquake hazards in a region. The task set of the workflow has four common sizes: 50, 100, 150, and 200. As shown in Fig. 4, with the growing number of tasks, the energy consumption of the PMEC algorithm increased at a much slower rate than that of the other two algorithms.

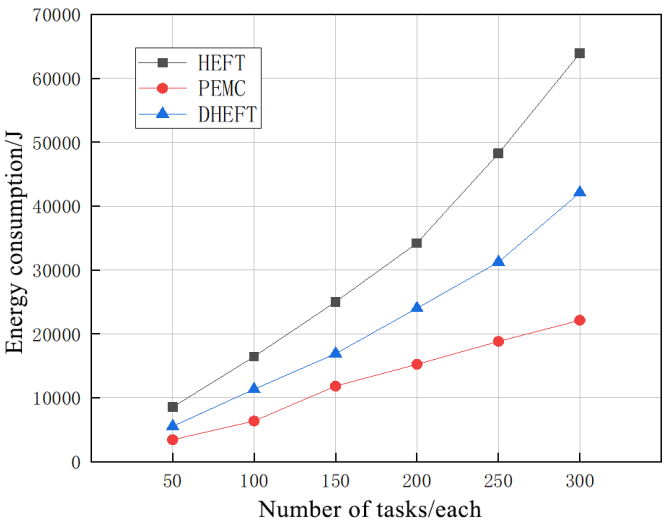

Figure 4: The energy consumption on CyberShake workflow.

The Montage workflow is a computationally intensive workflow in astronomy. The task set of the workflow also has four common sizes: 50, 100, 150, and 200. As shown in Fig. 5, with the growing number of tasks, the HEFT and DHEFT consumed more energy than PMEC, but their growth rates of energy consumption were not significantly higher than the growth rate of the PMEC.

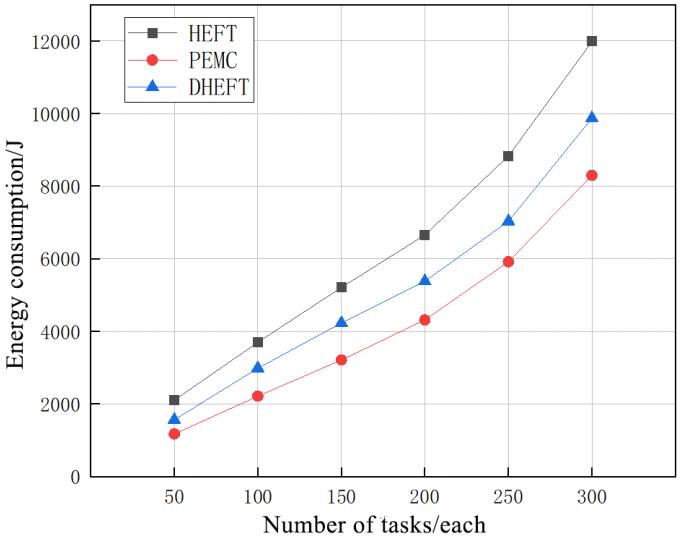

Figure 5: The energy consumption on Montage workflow.

The Sipht workflow is a biological workflow model used by Harvard University to find non-coding RNA (ncRNA) from viruses or bacteria. The task set of the workflow also has four common sizes: 50, 100, 150, and 200. As shown in Fig. 6, with the growing number of tasks, the three algorithms can be ranked as HEFT, DHEFT, and PEMC in descending order of energy consumption.

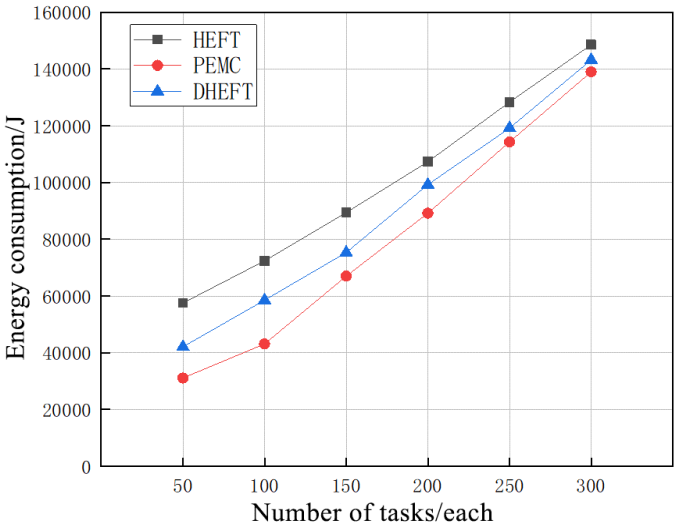

Figure 6: The energy consumption on Sipht workflow.

7. CONCLUSIONS

Considering the energy consumption in scheduling, this paper designs an energy-saving cloud workflow scheduling algorithm called the PMEC, which schedules the workflow by setting the priority of each task and selecting the virtual machine with relatively low energy consumption for task execution. Compared with the existing list-based scheduling algorithms, the PMEC

performs excellently in energy consumption. However, the runtime of our algorithm is much longer than that of the existing algorithms. This paper only considers the successful scheduling and a single workflow. The future research will investigate the scheduling of multiple workflows and scheduling failures.

ACKNOWLEDGEMENT

This work was supported by the Scientific and Technological Research Program of Chongqing Municipal Education Commission (KJ1602901; KJQN201903108), Chongqing College of Electronic Engineering Scientific Research Project (XJZK201809) and Xinxiang Medical University Education and Teaching Reform Research Project (2017-XYJG-41).

REFERENCES

[1] Luo, Z. Y.; Zhu, Z. H.; You, B.; Liu, J. H. (2018). Virtual workflow constrained time-accuracy optimization algorithm scheduling by iterative reduction, *Journal of Electronics and Information Technology*, Vol. 40, No. 8, 2013-2019, doi:10.11999/JEIT171038

[2] Hsieh, F.-S. (2017). A hybrid and scalable multi-agent approach for patient scheduling based on Petri net models, *Applied Intelligence*, Vol. 47, No. 4, 1068-1086, doi:10.1007/s10489-017-0935-y

[3] Jiang, H.; Liu, C. Y. (2019). Scheduling optimization of cloud resource supply chain through multi-objective particle swarm optimization, *International Journal of Simulation Modelling*, Vol. 18, No. 1, 163-174, doi:10.2507/IJSIMM18(1)CO3

[4] Luo, Z.-Y.; Wang, P.; You, B.; Zhu, S.-X. (2016). Serial reduction optimization research of complex product workflow's accuracy under the time constraint, *Advances in Mechanical Engineering*, Vol. 8, No. 10, 9 pages, doi:10.1177/1687814016672119

[5] Venuthurumilli, P.; Mandapati, S. (2019). An energy and deadline aware scheduling using greedy algorithm for cloud computing, *Ingénierie des Systèmes d'Information*, Vol. 24, No. 6, 583-590, doi:10.18280/isi.240604

[6] Golneshini, F. P.; Fazlollahtabar, H. (2019). Meta-heuristic algorithms for a clustering-based fuzzy bi-criteria hybrid flow shop scheduling problem, *Soft Computing*, Vol. 23, No. 22, 12103-12122, doi:10.1007/s00500-019-03767-0

[7] Liu, L.; Dessouky, M. (2017). A decomposition based hybrid heuristic algorithm for the joint passenger and freight train scheduling problem, *Computers & Operations Research*, Vol. 87, 165-182, doi:10.1016/j.cor.2017.06.009

[8] Borisovsky, P.; Eremeev, A.; Kallrath, J. (2020). Multi-product continuous plant scheduling: combination of decomposition, genetic algorithm, and constructive heuristic, *International Journal of Production Research*, Vol. 58, No. 9, 2677-2695, doi:10.1080/00207543.2019.1630764

[9] Abazari, F.; Analoui, M.; Takabi, H.; Fu, S. (2019). MOWS: multi-objective workflow scheduling in cloud computing based on heuristic algorithm, *Simulation Modelling Practice and Theory*, Vol. 93, 119-132, doi:10.1016/j.simpat.2018.10.004

[10] Hosseinioun, P.; Kheirabadi, M.; Tabbakh, S. R. K.; Ghaemi, R. (2020). A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm, *Journal of Parallel and Distributed Computing*, Vol. 143, 88-96, doi:10.1016/j.jpdc.2020.04.008

[11] Alzidani, M.; Dao, T.-M. (2019). Optimization of the cellular manufacturing scheduling using the RC-filter and EGD hybrid meta-heuristics approach, *International Journal on Interactive Design and Manufacturing*, Vol. 13, No. 4, 1549-1556, doi:10.1007/s12008-019-00562-x

[12] Singh, V.; Gupta, I.; Jana, P. K. (2018). A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources, *Future Generation Computer Systems*, Vol. 79, Part 1, 95-110, doi:10.1016/j.future.2017.09.054

[13] Entezari-Maleki, R.; Trivedi, K. S.; Sousa, L.; Movaghar, A. (2018). Performability-based workflow scheduling in grids, *The Computer Journal*, Vol. 61, No. 10, 1479-1495, doi:10.1093/comjnl/bxx125

[14] Ambursa, F. U.; Latip, R.; Abdullah, A.; Subramaniam, S. (2017). A particle swarm optimization and min-max-based workflow scheduling algorithm with QoS satisfaction for service-oriented grids, *The Journal of Supercomputing*, Vol. 73, No. 5, 2018-2051, doi:10.1007/s11227-016-1901-x

[15] Chirkin, A. M.; Belloum, A. S. Z.; Kovalchuk, S. V.; Makkes, M. X.; Melnik, M. A.; Visheratin, A. A.; Nasonov, D. A. (2017). Execution time estimation for workflow scheduling, *Future Generation Computer Systems*, Vol. 75, 376-387, doi:10.1016/j.future.2017.01.011

[16] Sathish, K.; Reddy, A. R. (2017). Workflow scheduling in grid computing environment using a hybrid GAACO approach, *Journal of The Institution of Engineers (India): Series B*, Vol. 98, 121-128, doi:10.1007/s40031-016-0230-z

[17] Kintsakis, A. M.; Psomopoulos, F. E.; Mitkas, P. A. (2019). Reinforcement learning based scheduling in a workflow management system, *Engineering Applications of Artificial Intelligence*, Vol. 81, 94-106, doi:10.1016/j.engappai.2019.02.013

[18] Shirvani, M. H. (2020). A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems, *Engineering Applications of Artificial Intelligence*, Vol. 90, Paper 103501, 20 pages, doi:10.1016/j.engappai.2020.103501

[19] Dubey, K.; Shams, M. Y.; Sharma, S. C.; Alarifi, A.; Amoon, M.; Nasr, A. A. (2019). A management system for servicing multi-organizations on community cloud model in secure cloud environment, *IEEE Access*, Vol. 7, 159535-159546, doi:10.1109/ACCESS.2019.2950110

[20] Rezaeian, A.; Naghibzadeh, M.; Epema, D. H. J. (2019). Fair multiple-workflow scheduling with different quality-of-service goals, *The Journal of Supercomputing*, Vol. 75, No. 2, 746-769, doi:10.1007/s11227-018-2604-2

[21] Deldari, A.; Naghibzadeh, M.; Abrishami, S. (2017). CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud, *The Journal of Supercomputing*, Vol. 73, No. 2, 756-781, doi:10.1007/s11227-016-1789-5

[22] Stavrinides, G. L.; Karatza, H. D. (2019). An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations, *Future Generation Computer Systems*, Vol. 96, 216-226, doi:10.1016/j.future.2019.02.019

[23] Barika, M.; Garg, S.; Ranjan, R. (2020). Cost effective stream workflow scheduling to handle application structural changes, *Future Generation Computer Systems*, Vol. 112, 348-361, doi:10.1016/j.future.2020.05.036

[24] Stavrinides, G. L.; Duro, F. R.; Karatza, H. D.; Blas, J. G.; Carretero, J. (2017). Different aspects of workflow scheduling in large-scale distributed systems, *Simulation Modelling Practice and Theory*, Vol. 70, 120-134, doi:10.1016/j.simpat.2016.10.009

[25] Arabnejad, H.; Barbosa, J. G. (2017). Multi-QoS constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems, *Future Generation Computer Systems*, Vol. 68, 211-221, doi:10.1016/j.future.2016.10.003