Complex & Intelligent Systems (2022) 8:4475–4482
https://doi.org/10.1007/s40747-021-00479-7

ORIGINAL ARTICLE

EHEFT-R: multi-objective task scheduling scheme in cloud computing

Honglin Zhang[1] · **Yaohua Wu**[1] · **Zaixing Sun**[2]

Received: 2 April 2021 / Accepted: 19 July 2021 / Published online: 31 July 2021

Abstract
In cloud computing, task scheduling and resource allocation are the two core issues of the IaaS layer. Efficient task scheduling algorithm can improve the matching efficiency between tasks and resources. In this paper, an enhanced heterogeneous earliest finish time based on rule (EHEFT-R) task scheduling algorithm is proposed to optimize task execution efficiency, quality of service (QoS) and energy consumption. In EHEFT-R, ordering rules based on priority constraints are used to optimize the quality of the initial solution, and the enhanced heterogeneous earliest finish time (HEFT) algorithm is used to ensure the global performance of the solution space. Simulation experiments verify the effectiveness and superiority of EHEFT-R.

Keywords Cloud computing · Task scheduling · HEFT · QoS · Energy consumption

Introduction

As a computing paradigm with centralized processing, cloud computing provides end users with on-demand services, thereby enabling various terminal devices with limited capabilities to load more complex applications. As of 2020, the scale of smart terminal devices has reached 50 billion units, the global total number is more than 40ZB, and more than half of these data need to be analyzed, processed and stored in the cloud [1].

Resource allocation and task scheduling optimization is one of the important research problems of cloud computing systems, and its solutions are related to the effectiveness of resource use and user service experience [2]. In view of the heterogeneity of cloud computing resources, the geographical dispersion of processors, and the optimization requirements of power consumption, new challenges are formed for resource allocation and task scheduling optimization.

✉ Honglin Zhang
201920522@mail.sdu.edu.cn

Yaohua Wu
MIKE.WU@263.NET

Zaixing Sun
Szx_1010@stu.hit.edu.cn

[1] Faculty of Control Science and Engineering, Shandong University, Jinan, China

[2] School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China

In reference [3], task scheduling problems in cloud computing scenarios are studied and more than 40 optimization indicators are summarized. These indicators can be divided into three categories: performance indicators, energy consumption indicators, and expenditure indicators. In terms of performance indicators, Literature [4] considers the number of gateways in the cloud computing architecture and the occupancy rate of buffers, and defines multiple time delay calculation equations. Reference [5] considers the subtask deadlines in the workflow, divides tasks into hard deadlines and soft deadlines, and considers their legitimacy and delay time. Literature [6] integrates response time, network congestion and service coverage as the user's QoS evaluation indicators, and designs a cloud-fog system to reduce time delay, improve end-user coverage, and guide the next step of resource allocation. In terms of energy consumption, Document [7] minimizes the total energy consumption of cloud computing equipment under the constraints of application deadlines. In terms of cost, literature [8] comprehensively considers constraints, such as performance and virtual machine capacity, and takes the sum of virtual machine configuration costs and communication costs as the optimization goal to obtain the optimal user base station selection, virtual machine matching and other solutions. In addition, some other relevant terms of scheduling problems and algorithms are as follows. In [13], the IPSO algorithm is proposed to improve the efficiency of resource scheduling while facing a large amount of tasks. In [14], a discrete imperialist competitive algorithm (DICA) was proposed

to minimize the makespan and energy consumption of the resource-constrained hybrid flowshop problem (RCHFS). In [15], efficient exact algorithms are devised for the multitasking scheduling problems with batch distribution and due date assignment (DDA). Reference [16] provides a comprehensive literature review of production scheduling for intelligent manufacturing systems with the energy-related constraints and objectives, energy efficiency-related publications are classified and analyzed according to five criteria.

It can be seen from the above review that although scholars have optimized and solved different indicators of cloud computing task scheduling, there is still a lack of comprehensive optimization scheduling algorithms. Besides, the performance of classic HEFT algorithm is not satisfactory. In this paper, an initial scheduling method based on priority rule is proposed to improve the performance of HEFT. And a novel EHEFT-R task scheduling algorithm is proposed based on this method. The contributions of this paper are as follows:

1. An initial ranking algorithm based on priority rules is designed;
2. An enhanced HEFT algorithm is designed to realize the synchronization of task sorting and processor allocation;
3. The multi-objective optimization of makespan, energy consumption and QoS.

The remainder of the paper is as follows: the second part describes the problem and its models. The third part describes the research methodology. The fourth part is experiments and data analysis. Finally, the fifth part presents the conclusion.

Problem description and models

System model

Task scheduling and resource allocation are two sequential processes of cloud computing system [9]. The essence of task scheduling is to sort different task streams from different users. Resource allocation is to allocate ordered tasks to corresponding computing resources, namely virtual machines. Task scheduling is a sorting problem, and resource allocation is an assignment problem. Task scheduling and resource allocation are combined optimization problems, which are NP-hard.

The diagram of the task scheduling model is shown in Fig. 1. In the task scheduling model, users send one or more groups of computing task requests with priority constraints to the cloud server. After receiving the calculation request, the data center issues scheduling instructions to the task scheduler. After optimizing the scheduling algorithm

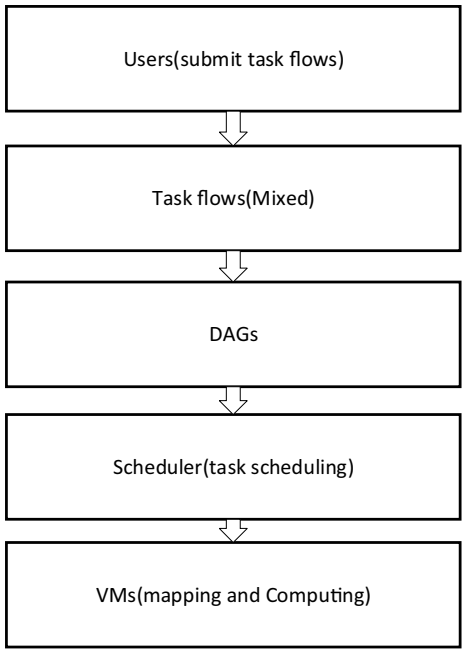

Fig. 1 System model of task scheduling

and considering the priority constraints between tasks, the scheduler sorts the tasks and returns the sorting results to the data center. Subsequently, the data center assigns the sorted tasks to cloud computing nodes and subordinate VMs.

DAG priority constraint

The task flow submitted by the user contains many subtasks. For the task flow of the same user, these tasks are often sequence-related or have strong priority constraints. Taking Fig. 2 as an example, suppose that tasks $T1$–$T10$ are subtasks in a set of task flows of a user. In Fig. 2, $T2$ can only be executed after $T1$ has completed the calculation. For the task flows of different users, as shown in Fig. 1, the task flows of different users will be sorted after being mixed. Suppose that Fig. 2 is a mixed task flow, and $T1$–$T10$ are subtasks in the task flow from 10 different users. The subtask T2 from user 2 needs to be executed after the calculation of the subtask T1 of user 1 is completed.

According to the above explanation of priority constraints and the morphological characteristics of Fig. 2, the task flow with priority constraints can be represented by DAG [9]. DAG can be defined as:

$$G = (T, E, C, W) \tag{1}$$

According to [9], parameters and variables are defined as follows: T represents the set of all tasks, where $T = \{T_i | i = 1, 2, ..., n\}$, in which T_i represents a task in DAG, n is the number of tasks. E is the set of edges between tasks,

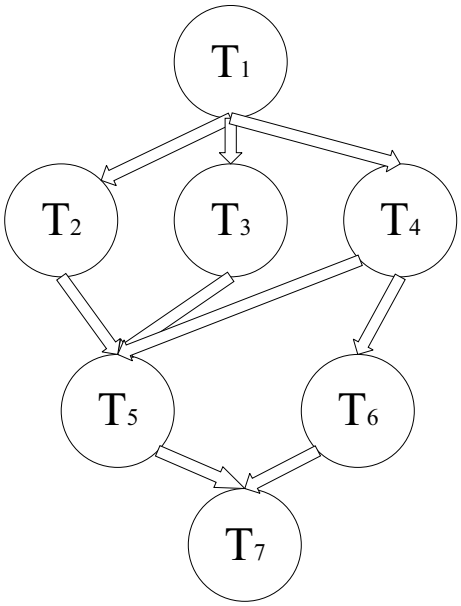

Fig. 2 DAG of task flow

where $E = \{E_{ij} = (T_i, T_j)|T_i, T_j \in T, ij\}$, in which E_{ij} is the edge between T_i and. C is the set of communication costs between tasks with connections, where $C = \{C_{ij}|ij\}$, in which C_{ij} represents the communication cost between T_i and T_j. W is the set of weights of tasks, where $W = \{W_i|i = 1, 2, ..., n\}$, it represents the computation costs of the tasks; for example, W_i is the weight of task T_i, which represents the computation cost of T_i.

Mathematical models

In this section, mathematical models are designed to evaluate makespan, QoS and total energy consumption of the studied cloud system.

First, Makespan. In production scheduling, makespan represents the completion time of the last workpiece, that is, the maximum completion time. Similarly, in cloud computing task scheduling, makespan represents the time when the last subtask is completed, that is, the maximum execution time. Mathematically, makespan can be calculated by the equation represented by (2):

$$C_{\max} = \max(\sum_{i=1}^{n} T_{i,1}F_{i,1} \cdots \sum_{i=1}^{n} T_{i,k}F_{i,k}) \tag{2}$$

$$\text{where} \quad F_{i,k} = \begin{cases} 1, & if \ T_i \to VM_k \\ 0, & else \end{cases} \tag{3}$$

where $T_i \to VM_k$ means that task T_i is allocated to VM k, $T_{i,k}$ is the execution time of task T_i on VM k.

Second, the total energy consumption. Generally speaking, the energy consumption of cloud computing mainly includes CPU, memory, storage and network transmission. These devices or processes that generate energy consumption can be divided into two categories, dynamic energy consumption and static energy consumption. Dynamic energy consumption is the main reason for the huge energy consumption of cloud computing centers. At the same time, since the static energy consumption is approximately linear and the dynamic energy consumption is random, the optimization of dynamic energy consumption is more challenging. We study dynamic energy consumption. According to the energy consumption calculation method proposed in reference [10], the execution energy consumption is shown in (4) and (5):

$$E_{i,k} = P_k E_{T_i^j} \tag{4}$$

$$E_{total}^{busy} = \sum_{i=1}^{n}\sum_{k=1}^{m} x_{ij}P_k E_{T_i^j} \tag{5}$$

where $E_{T_i^j}$ denotes the execution time of task T_i onVM k, P_k is the power of VM k, and $E_{i,k}$ denotes energy consumption consumed by task T_i executing on VM k.

Finally, the QoS. QoS is an important indicator for evaluating users' satisfaction with cloud computing services. In this paper, what affects user QoS is service efficiency, namely makespan. A QoS evaluation formula based on makespan is designed, as shown in (6).

$$QoS = \frac{\min \sum_{i=1}^{n}\sum_{j=1}^{m} P_{i,j}^{EFT}}{C\max} \tag{6}$$

where $P_{i,j}^{EFT}$ denotes the earliest completion time of task i on the fastest VM j. Since total $P_{i,j}^{EFT}$ is fixed, the less makespan, the better QoS.

EHEFT-R

HEFT algorithm is a classic and efficient static task scheduling algorithm. For task scheduling with DAG constraints, HEFT algorithm can effectively reduce makespan. The design idea of HEFT algorithm is to realize scheduling through two stages of task scheduling and virtual machine selection according to the order correlation of task scheduling and resource allocation.

Task sequencing phase

Although DAG has strong constraints on the priority of tasks, there may be multiple tasks under the same priority, so the priority of these tasks at the same level needs to be determined before the tasks are sorted.

To determine the priority of tasks at the same level, first calculate the rank value of task *i* by (7). Tasks with a higher rank value get higher priority and will be prioritized to the virtual machine.

$$rank_{T_i} = \overline{\omega_i} + \max(\overline{c_{ij}} + rank_{T_j}) \tag{7}$$

$$\overline{\omega_i} = \frac{1}{m}\sum_{k=1}^{m}\omega_{i,k} \tag{8}$$

$$c_{ij} = L_k + \frac{d_{ij}}{B_{kl}} \tag{9}$$

$$\overline{c_{ij}} = \overline{L} + \frac{d_{ij}}{\overline{B}} \tag{10}$$

where $\overline{\omega_i}$ is the average computing cost of task T_i, $\overline{c_{ij}}$ is the average communication cost of T_i and T_j, $rank_{T_j}$ is rank value of task T_j.

VM allocation phase

After the tasks are sorted according to DAG priority and rank priority, the second stage is to arrange the tasks on the virtual machine. The principle of virtual machine selection is the earliest completion time rule, that is, the task TI is scheduled to the virtual machine that completes the computing task earliest.

By priority sorting and virtual machine selection, the two stages of HEFT are completed, and the final task scheduling plan is obtained after the last task is completed. The steps of the HEFT algorithm are summarized as follows:

Step 1. Set communication costs of tasks; set communication costs of edges.

Step 2. Compute rank values of all tasks in reverse order, from the exit task to the entry task.

Step 3. Sort the ranked tasks into list L_u, with non-increasing order of values.

Step 4. Compute EFTs. Take the first task T_1^u from L_u, for each VM P_m, compute $EFT(T_1^u, P_m)$, allocate T_1^u to P_m^* that minimizes EFT of T_1^u.

Step 5. Repeat the computing of EFTs for all tasks, until the last task T_{last}^u is assigned.

Step 6. Obtain $C_{\max}$.

EHEFT-R

Although HEFT is a classic algorithm that can minimize makespan and is widely used in cloud computing task scheduling, it has fatal flaws. First, HEFT concentrates only on sorts of makespan, which is a single-objective optimization algorithm, and it is difficult to achieve feasible results when facing multiple objectives. Second, it simply divides task scheduling into two stages: task sequencing and virtual machine allocation, ignoring the coupling characteristics of these two stages, which is also a common problem in two-stage optimization. According to the two shortcomings of HEFT, a rule-based enhanced heft algorithm (EHEFT-R) for multi-objective task scheduling is proposed.

In this paper, three indicators of completion time, energy consumption and QoS are considered at the same time, and HEFT obviously cannot achieve multi-objective optimization. Therefore, HEFT is improved to have multi-objective optimization scheduling capabilities. According to [9], the rank value is composed of calculation cost and communication cost, both of which only consider the factors of makespan index. Consider incorporating energy consumption and QoS into the calculation of rank, as shown in (11):

$$rank_{T_i} = \alpha\overline{\omega_i} + \beta\max(\overline{c_{ij}} + rank_{T_j}) + \gamma p_k e_i \tag{11}$$

where α,β,γ are weight of makespan, energy and QoS.

In addition, as much as possible, the algorithm should not violate the characteristics of the problem itself. Although task sequencing and processor selection are order-related, they are still a coupled process rather than two independent processes. Unlike the classic HEFT algorithm, this paper treats task sequencing and virtual machine allocation as parallel processes. After one or more tasks determine their execution order, they are immediately arranged on the virtual machine corresponding to the earliest completion time. After all tasks are sorted, the virtual machine selection process is also approximately completed synchronously, and a task sequence π is obtained. Subsequently, since task sequencing and virtual machine selection at the same time may cause the task sequence to be sub-optimal, π is used as the initial task sequence for task-virtual machine remapping. Remapping performs virtual machine selection based on rules.

π may not be the optimal ranking because although the tasks are sorted according to the rank value from high to low, there will be no conflicts between tasks at different priority levels, and partial optimization can be guaranteed. But at the same priority level, similar to task sequencing, the rank value does not completely guarantee that tasks at this level are allocated to the optimal virtual machine. To solve the virtual machine selection priority of tasks of the same

level, the remapping rules are formulated as follows: In π, start from the task with the highest rank value and check in descending order of rank. If two adjacent tasks are not in the same layer, the virtual machine assignments of the two tasks will not be changed; if two adjacent tasks are in the same layer and their earliest completion times fall on different virtual machines, the virtual machine allocation will not be changed; if two adjacent tasks are in the same layer and their earliest completion time falls on the same virtual machine, compare the EFT/rank of the two tasks, the one with the larger value will be assigned to this virtual machine. This rule is called a hybrid SPT rule based on EFT and rank (SPT-EFT-R).SPT-EFT-R. The flow of SPT-EFT-R is shown in Fig. 3.

Experiments

In this part, a set of small-scale experiments are designed to verify the effectiveness of EHEFT-R. Then, five sets of benchmark experiments were used to test the performance of EHEFT-R. According to [17], sensitivity analysis should be performed while there are parameters influencing the iterations and evolutions. In this paper, however, the proposed EHEFT-R is designed based on exact rules, and no iteration or evolution is concluded. Therefore, sensitivity analysis is not performed in this subsection.

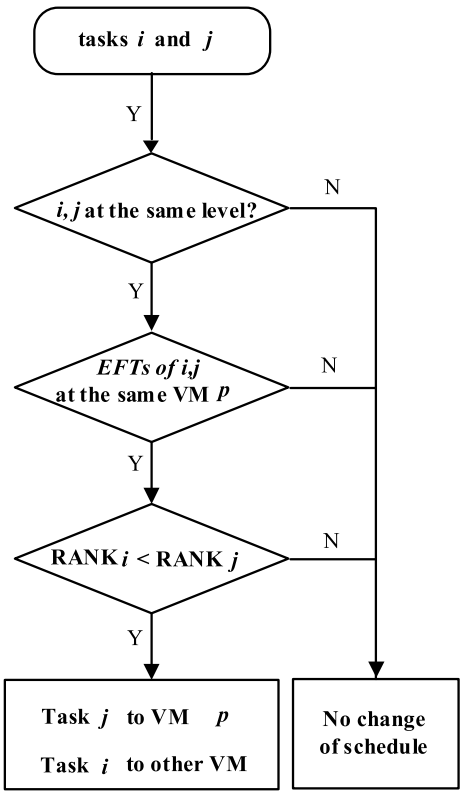

Fig. 3 SPT-EFT-R

Small-scale experiment

We modified the illustrative example of the literature (Zhaotong) for testing. Considering that the energy consumption and QoS are not calculated in the experiment in [9], the energy consumption information table has been added, as shown in Table 1.

It can be seen from Table 2 that in the experiment of [9], EHEFT-R has the best three indicators of makespan, energy consumption and QoS. Therefore, the EHEFT-R proposed in this paper is effective.

Large-scale benchmark

In this subpart, the benchmark datasets provided by [11] are used to compare EHEFT-R, algorithms proposed in reference [12], and HEFT. According to [12], the problem cases are classified into 12 categories in the benchmark, and 100 different cases are included in each category. Each problem case in dataset records the makespan and energy consumption for the 512_16 data, i.e. 512 tasks assigned to 16 VMs. Any VM in the dataset is consistent, inconsistent, or semi-consistent in terms of consistency configurations. Task heterogeneity and VM heterogeneity could be high or low for different configurations of the VM. In this way, 12 problem

Table 1 Unit energy consumption

T_i	E_1	E_2	E_3
T_1	132	98	102
T_2	165	291	201
T_3	146	35	192
T_4	210	180	161
T_5	391	340	290
T_6	190	160	140
T_7	98	81	151
T_8	41	20	69
T_9	193	152	201

Table 2 Small-scale experiment

	Makespan	Energy consumption	QoS
EHEFT-R	**88**	**3487**	**1.7159**
QL-HEFT	90	3563	1.6778
HEFT-D	97	4016	1.5567
HEFT-U	99	4095	1.5253
CPOP	116	4817	1.3017

Table 3 Large-scale benchmark

VMs	Makespan (min)			Energy consumption (kWh)		
	HEFT	NSGA-II	EHEFT-R	HEFT	NSGA-II	EHEFT-R
c-hihi	181	134	**106**	23,150	18,924	**16,926**
c-lohi	174	128	**105**	22,847	18,084	**16,915**
c-hilo	169	130	**106**	22,796	18,241	**16,926**
c-lolo	182	134	**101**	23,215	18,924	**16,251**
i-hihi	98	67	**52**	13,084	8173	**7048**
i-lohi	94	58	**54**	12,597	7763	**7256**
i-hilo	81	67	**48**	10,194	8173	**6752**
i-lolo	82	59	**50**	10,236	7794	**6982**
s-hihi	207	175	**84**	6452	3470	**1695**
s-hilo	93	58	**53**	4207	3760	**2870**
s-lohi	81	63	**39**	5926	5076	**3008**
s-lolo	81	38	**29**	5156	4356	**3983**

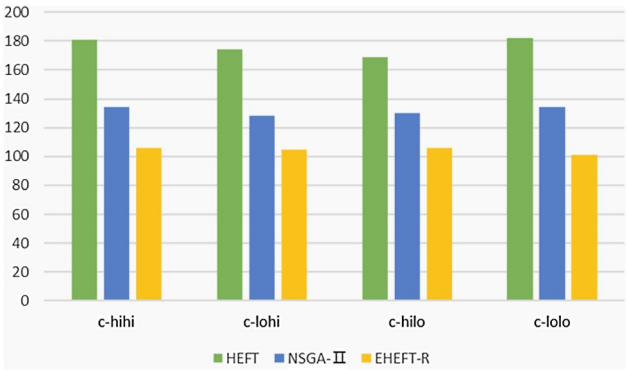

Fig. 4 *u-c* experiments on makespan

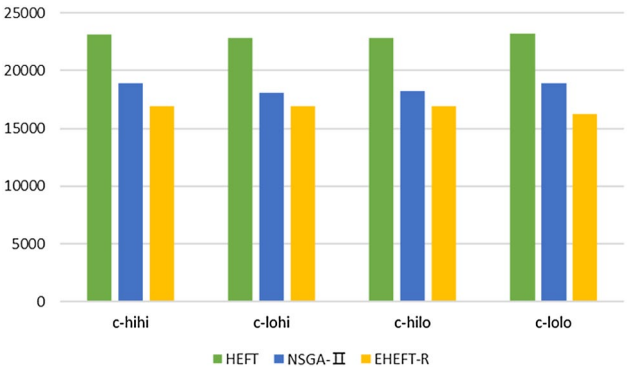

Fig. 5 *u-c* experiments on energy consumption

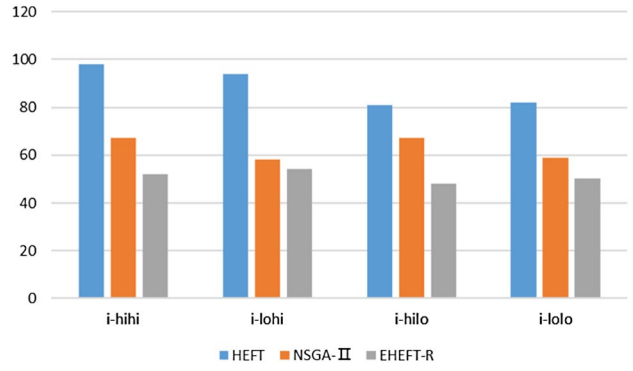

Fig. 6 *u-i* experiments on makespan

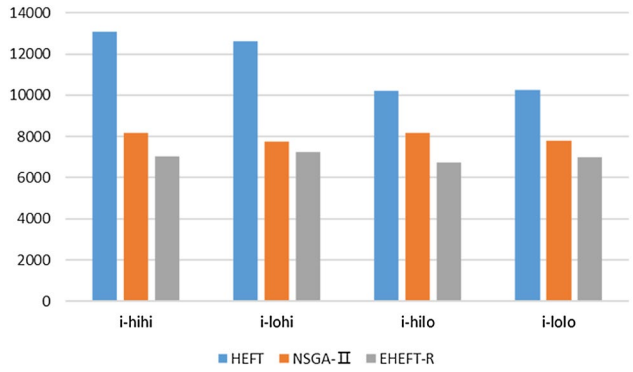

Fig. 7 *u-i* experiments on energy consumption

cases arise out of the 12 combinations of consistency, task heterogeneity and machine heterogeneity.

Table 3 shows the comparative results for the 12 VMs. From the results, it is evident that EHEFT-R is better than NSGA-II and HEFT for both makespan and energy consumption. For detailed comparison of the algorithms, the indicator histograms of u-c, u-i and u-s are drawn, as shown in Figs. 4, 5, 6, 7, 8 and 9. Figures 4 and 5 list the comparison of the results in the four cases of c-hihi, c-lohi, c-hilo and c-lolo. Figures 6 and 7 list the comparison of the results in the four cases of i-hihi, i-hilo, i-lohi and i-lolo. Figures 8 and 9 list the comparison of the results in the four cases of

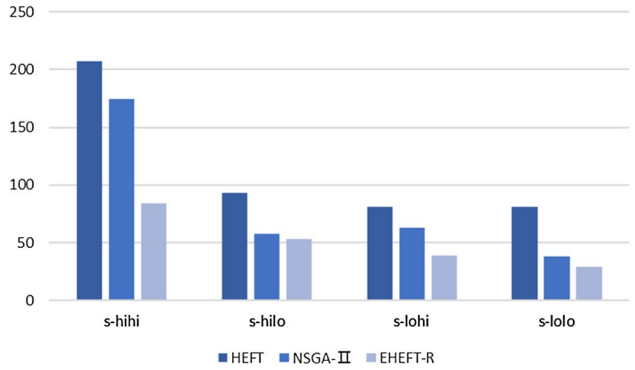

Fig. 8 *u-s* experiments on makespan

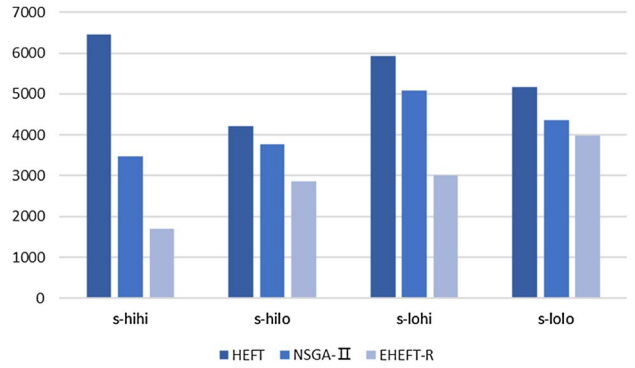

Fig. 9 *u-s* experiments on energy consumption

s-hihi, s-hilo, s-lohi and s-lolo. From three sets of detailed comparison histograms, EHEFT-R has achieved better results than NSGA-II and HEFT in 12 cases, which verifies the effectiveness and superiority of the proposed algorithm.

Conclusion

In this paper, we proposed an novel EHEFT-R task scheduling algorithm to solve the static task scheduling problem in the cloud computing environment. The design of EHEFT-R considers two key points. One is to solve the segmentation optimization defect of HEFT through the synchronization of task sequencing and resource allocation, and the other is to optimize the local search of the sequencing that may be caused by remapping resource allocation.

Finally, we designed two sets of experiments to compare EHEFT-R with other algorithms. The evaluation indicators are makespan, energy consumption and QoS. First, compare EHEFT-R, QL-HEFT, HEFT-D, HEFT-U and CPOP in small-scale experiments. Experimental results show that EHEFT is far superior to the other four algorithms. For large-scale problems, EHEFT-R, EGA and NSGAII are compared in the standard test set. The test results show that in a large-scale experimental environment, EHEFT-R obtains much better solutions than both two other algorithms.

Two key points make EHEFT-R performing better. First, reasonable sorting rules. Compared with HEFT, proposed EHEFT-R takes the coupling properties of task sequencing and virtual machine selection under consideration. Second, remapping and rescheduling. The rescheduling mechanism ensures the quality of the solution, avoids falling into local optimization, and enhances the ability of local optimization.

Although HEFT is not a well-designed algorithm, it still provides inspiration for our future research. Classic HEFT algorithm regards two coupled processes as two independent processes, which is wrong and inefficient, but we can get two relatively independent research objects through decoupling.

Acknowledgements Honglin Zhang would like to thank Prof. Yin Yunqiang for his kind advice on algorithm design. Prof. Yin's Optimization Theory course is the source of some of the ideas in this paper.

Declarations

Conflict of interest The corresponding author states that there is no conflict of interest.

References

1. Gupta R (2012) Above the clouds: a view of cloud computing. Eecs Dept Univ Calif Berkeley 53(4):50–58
2. Liu C, Li K, Li K et al (2017) A new service mechanism for profit optimizations of a cloud provider and its users. IEEE Trans Cloud Comput 2017:1–1
3. BellendorfJulian MZD (2020) Classification of optimization problems in fog computing. Fut Gen Comput Syst 107:158–176
4. Desikan K, Srinivasan M, Murthy C (2017) A novel distributed latency-aware data processing in fog computing-enabled IoT networks. ACM 1–6
5. Wu CG, Wang L (2019) A deadline-aware estimation of distribution algorithm for resource scheduling in fog computing systems. In: 2019 IEEE Congress on Evolutionary Computation (CEC). IEEE
6. Lin Y, Shen H (2015) IEEE 2015 44th International Conference on Parallel Processing (ICPP) - Beijing, China (2015.9.1–2015.9.4) 2015 44th International Conference on Parallel Processing - Cloud Fog: Towards High Quality of Experience in Cloud

Gaming[C]// International Conference on Parallel Processing. IEEE, 2015:500–509
7. Deng R, Lu R, Lai C et al (2017) Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J 3(6):1171–1181
8. Gu L, Zeng D, Guo S (2017) Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Trans Emerg Top Comput 5(1):108–119
9. Tong Z, Deng X, H Chen et al (2019) QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment. Neural Comput Appl
10. Wang X, Wang Y, Yue C (2016) An energy-aware bi-level optimization model for multi-job scheduling problems under cloud computing. Soft Comput 20(1)
11. Yan S, Lin F, Xu H (2018) Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. Wireless Pers Commun 102(1):1–17
12. Hussain M, Wei LF, Lakhan A et al (2021) Energy and performance-efficient task scheduling in heterogeneous virtualized cloud computing. Sustain Comput Inf Syst 30(3):100517
13. Yu H (2020) Evaluation of cloud computing resource scheduling based on improved optimization algorithm. Complex Intell Syst 2020:1–6
14. Tao XR, Li JQ, Huang TH et al (2020) Discrete imperialist competitive algorithm for the resource-constrained hybrid flowshop problem with energy consumption. Complex Intell Syst 7(1):311
15. Xu X, Yin G, Wang C (2020) Multitasking scheduling with batch distribution and due date assignment. Complex Intell Syst 7(1):191
16. Gao K, Huang Y, Sadollah A et al (2019) A review of energy-efficient scheduling in intelligent production systems. Complex Intell Syst 6(23):237–249
17. He L, Li W, Zhang Y et al (2019) A discrete multi-objective fireworks algorithm for flowshop scheduling with sequence-dependent setup times. Swarm Evolut Comput 51:100575