The Journal of Supercomputing (2021) 77:11643–11681
https://doi.org/10.1007/s11227-021-03764-x

Reliability-aware task scheduling for energy efficiency on heterogeneous multiprocessor systems

Zexi Deng[1] · Dunqian Cao[2] · Hong Shen[1] · Zihan Yan[1] · Huimin Huang[1]

Accepted: 19 March 2021 / Published online: 29 March 2021

Abstract

Recent studies mainly focus on high performance or low power consumption for task scheduling on heterogeneous multiprocessor systems (HMSs). Dynamic voltage and frequency scaling (DVFS) is an important energy reduction technique, which adjusts the voltage and frequency of the processor while the task is executing. However, some studies have shown that reducing the voltage of processor increases the transient failure rate, which reduces system reliability. In this paper, we aim at addressing the scheduling problem of optimizing energy under makespan and reliability constraints on HMSs with DVFS. We first propose an improved whale optimization algorithm (WOA) deploying opposition-based learning and individual selection strategy, which can balance the exploration and exploitation ability. To maintain population diversity, we then apply a constrained rank-based method which retains some infeasible individuals in the population. Finally, we reschedule the Critical Path Nodes (CPNs) to further improve the performance of improved WOA. The main difference between our work and most previous works is that we study a new scheduling problem, and utilize an improved WOA algorithm integrating with rescheduling CPNs and a constrained rank-based method. Extensive experiments are conducted to evaluate our proposed algorithm, and the evaluation results show that our proposed algorithm is compelling in comparison with the state-of-the-art algorithms.

Keywords Task scheduling · Whale optimization algorithm · Opposition-based learning · Heterogeneous multiprocessor systems

✉ Hong Shen
shenh3@mail.sysu.edu.cn

[1] School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

[2] School of Science, Guangxi University for Nationalities, Nanning 530006, Guangxi, China

1 Introduction

Heterogeneous multiprocessor systems (HMSs), which are usually composed of heterogeneous computing and communication resources, are primary and cost-effective infrastructures used for commercial and expensive scientific applications. HMSs, e.g., the Sunway TaihuLight system [1], use various processors with different processing capabilities [2]. Energy consumption of HMSs leads to serious economic, ecological and technical problems. The Central Processing Unit (CPU), power supply, memory and fan constitute the main power consumption of high performance computing nodes, thus reducing the energy consumed by the CPU can effectively save energy. In this paper, we focus on the processor power consumption.

Due to the significance of energy conservation, various techniques have been investigated and developed on this issue, including dynamic voltage and frequency scaling (DVFS), resource hibernation, and memory optimization [3]. Among these techniques, DVFS has been considered to be a very promising technique because of its capability for energy saving [4, 5], which changes the voltage and frequency ratio and maintains performance to save energy [6]. When the processor's voltage/frequency is reduced to save power, transient faults in the system increase [7, 8]. Reliability is a critical issue for high-performance systems. Due to the affection of system performance, chip temperature and cosmic rays, transient faults occur unpredictably when executing tasks [9]. System failures caused by transient failures occur frequently in practice, and thus the system failure probability cannot be ignored in high-performance systems [9]. Low frequency leads to low reliability and low energy consumption, while high frequency leads to high reliability and high energy consumption [8].

Chen et al. [5] and Deng et al. [12] considered the problem of minimizing energy consumption under time constraints. Their works have received extensive attention from researchers; however, they did not consider reliability. Zhang et al. [13] considered maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. Zhang et al. [14] considered a bi-objective scheduling problem of the energy consumption and reliability in HMSs [5, 12]. However, they used energy consumption models which were different from those proposed in [5, 12]. Inspired by previous works [5, 12–14], we propose a new model, the goal of which is to minimize energy under makespan and reliability constraints while still meet the precedence constraints of tasks on HMSs with DVFS. The energy consumption component used in our model is the same as those in [5, 12], but different from those in [13, 14]. To the best of our knowledge, this is the first attempt that this problem has been raised. How to implement an efficient scheduling scheme to reduce the system power consumption in HMSs has become a hot topic in today's research [2, 12]. In this work, we solely concern with the static scheduling issues.

Generally, task scheduling on HMSs is an NP-hard problem [15–20], and the number of choices for voltage adjustment increases exponentially on such systems [5, 12]. There is a conflict between energy and reliability goals [8], which makes it more difficult to solve energy optimization than not considering reliability. Task scheduling algorithms can be classified into: heuristics and random search ones.

Heuristic scheduling methods are based on greedy optimal selection of some heuristic strategies [5, 18, 20–24]. In complex scheduling situation, if only the basic heuristic scheduling method is used, its performance is often not very good [5, 12, 25]. A heuristic strategy, named priority of critical path nodes modification, was used to reschedule the critical path nodes to reduce the makespan [26]; however, it only considered reducing completion time without considering energy consumption and adjusting voltage scales, which may not be able to minimize the system energy consumption to the greatest extent by adjusting voltage scales, and thus this strategy cannot be used directly to solve our problem.

Many meta-heuristic algorithms have been adopted to task scheduling problems [5, 12, 14, 27–31]. Chen et al. proposed a Quantum-inspired Hyper-heuristics Algorithm (QHA) [5] to solve the problem of power and performance tradeoff optimization; however, it has high space complexity, and its schedule may not satisfy the priority constraint relationship. In our other work, we proposed a cuckoo search algorithm deploying Gaussian random walk and Adaptive discovery probability (GACS) to address the performance-constrained energy optimization, and then used a cost-to-time ratio strategy to improve the performance of GACS [12]; however, the cost-to-time ratio strategy did not consider reliability, which may result in the fact that the generated solutions violate reliability constraints, and thus it cannot be used to solve the problem raised in this paper. In our another work, we proposed two heuristic algorithms to address problem of task scheduling and data allocation [32]; however, reference [32] focused on data-dependent scheduling, which is different from data-independent scheduling discussed in this paper. Besides, these algorithms did not take reliability into account.

Although some scheduling algorithms have been proposed to solve constrained optimization problems, they still suffer from a number of limitations: (1) Some methods use heuristic algorithms based on greedy strategies to solve power and performance tradeoff optimizations, but these heuristic methods are not aimed at multi-constraint optimization problem [8, 13, 33, 34]; (2) some existing methods use meta-heuristics to solve power and performance tradeoff optimization; however, these scheduling algorithms do not focus on multi-constrained optimization problems [5, 12, 14], or some meta-heuristics algorithms have high time costs [5, 14]; (3) some existing algorithms do not work well for task scheduling problem when solving large-scale task graphs [5, 12]. Thus, it is a challenging and interesting work to design a simple and effective scheduling algorithm with good scalability for multi-constraint optimization.

The whale optimization algorithm (WOA) is a new intelligent optimization algorithm proposed by Mirjalili and Lewis in 2016 [35]. WOA simulates the hunting behavior of humpback whales in nature, and optimizes the complex optimization problems through the process of encircling prey, bubble-net attacking and search for prey, which has the characteristics of simple principle, few parameter settings and strong optimization performance [35, 36]. WOA has been proven to be significantly superior to particle swarm optimization (PSO) algorithm and gravitational search algorithm (GSA) in terms of solution accuracy and convergence speed performance [35], and achieves excellent results when solving large-scale complex optimization problems [36]. It has been successfully applied to task scheduling and other fields

[36, 37]. Abdel-Basset et al. in [38] proposed an improved whale algorithm (IWA) to solve the task scheduling in multiprocessor systems, and the results showed the superiority of the improved WOA; however, they did not consider multi-constrained optimization problems. Therefore, we try to apply the WOA algorithm to solve task scheduling problems in this paper. However, its convergence accuracy and its ability to jump out of local optimum still need to be improved. Thus, we first propose an effective whale optimization algorithm based on opposite learning and individual selection strategy (OIWOA) which can balance the ability of global exploration and local exploitation and then use it to search task graphs.

Our approach is inspired by the fact that each scheduling methods has its pros and cons, and sometime they may be complementary to each other [12, 25]. Since HMSs also consume energy in idle time, if we can redistribute Critical Path Nodes (CPNs) to reduce the tasks completion time without increasing energy consumption, and adjust its frequency to maintain high system reliability, then we can optimize energy consumption under makespan and reliability constraints. Thus, we propose an OIWOA algorithm improved by a heuristic principle (OIWOAH). Specifically, we first propose the OIWOA algorithm to evolve the task-to-processor mapping solutions and then use a downward-ranking heuristic to determine the task priority assignment in each processor. After performing the OIWOA algorithm, we attempt to redistribute CPNs to the processor where its parent node which is also a critical path node is located, and adjust the processor's frequency state to reduce system completion time and power consumption simultaneously.

The main difference between our work and most previous works is that we study a new scheduling problem of minimizing energy under makespan and reliability constraints while still meeting the precedence constraints of tasks on HMSs with DVFS, and utilize an improved WOA algorithm combining with rescheduling CPNs and a constrained rank-based method. We address the multi-constrained scheduling problem effectively by using the OIWOAH algorithm. In the experiment, our algorithm and the compared algorithms adopt a consistent strategy: we first perform the random heuristic search strategy within the number of fitness evaluations and then use the heuristic strategy which reschedule the critical path nodes only one time. The main contributions of this paper are summarized as below.

- We study a new scheduling problem of minimizing energy under makespan and reliability constraints while still meeting the precedence constraints of tasks on HMSs with DVFS. As far as we know, this is the first attempt that this problem has been raised.
- We propose an improved WOA algorithm deploying opposite learning and individual selection strategy, which can effectively balance global exploration and local exploitation in task graph search.
- To maintain population diversity, we apply a constrained rank-based method which retains some infeasible individuals in the population.
- We reschedule CPNs to reduce makespan without increasing energy consumption and adjust its frequency to maintain high system reliability, which can further improve the performance of OIWOA.

- We conduct extensive experiments results, which reveal that our OIWOAH algorithm outperforms the state-of-the-art algorithms.

In the next section, we present some existing related researches. In Sect. 3, we present the model of HMSs. In Sect. 4, we detail our OIWOAH algorithm. In Sect. 5, we evaluate our OIWOAH algorithm with some algorithms. Finally, we conclude this paper and discuss future work.

2 Related work

The energy-aware task scheduling has gradually become one of the most significant problems in high-performance computing [4, 7, 8, 11, 13, 38–42]. Tang et al. [15] investigated the energy-aware scheduling problem of deadline-constrained workflow applications on HMSs. Taheri et al. [2] proposed a two-phase genetic algorithm hybridized with a spectral partitioning (G-SP) to minimize energy consumption. Goubaa et al. in [43] investigated scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems. Many studies have been investigated energy-aware task scheduling on processors with DVFS, which usually consider power and performance tradeoff optimization [44–52]. For the continuous DVFS situation, there were many studies considered reducing energy consumption [53, 54]. For the discrete DVFS situation, the authors in [4, 5, 10, 23, 33, 38, 55, 56] investigated the scheduling problems. Lee and Zomaya in [4] proposed two classical algorithms named Energy-Conscious Scheduling (ECS) and ECS+idle. Abdel-Basset et al. in [38] proposed an energy-aware whale optimization algorithm for real-time task scheduling in multiprocessor systems to optimize the energy consumption and makespan. Hu et al. [23] proposed a reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints. Quan et al. [18] investigated a problem of optimizing the makespan for energy consumption constrained parallel applications on heterogeneous computing systems. Salami et al. [42] investigated both the fairness problem and energy efficiency in heterogeneous multi-core processors. However, these algorithms assume that all nodes do not fail during execution and do not accurately reflect real parallel computing systems.

Transient faults inevitably occur in practice, and thus reliability is an important factor to consider for designing high-performance system [57]. Reliability-aware task scheduling algorithms have received widespread attention [9, 58–63]. However, these works mainly focus on optimizing makespan or reliability, and they do not consider energy consumption.

Recently, some researches consider both energy consumption and reliability for task scheduling with DVFS. According to different algorithm strategies, the proposed algorithms are either heuristic or meta-heuristic. Heuristic algorithms usually schedule tasks based on task priorities and greedy mapping strategies [8, 13, 34, 64–68]. Kumar et al. [68] adopted an active replication-based framework to obtain reliability-aware energy optimized scheduling of non-preemptive periodic real-time tasks on HMSs. Huang et al. [69] investigated dynamic scheduling of tasks modeled

by directed acyclic graphs; however, they did not consider time constraint. Hassan et al. [70] proposed a smart energy and reliability-aware scheduling algorithm for workflow execution in DVFS-enabled cloud environment. Abdi et al. [71] proposed a quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. However, these heuristic algorithms cannot produce consistent results for a wide range of problems, and they usually produce unacceptable solutions in complex scheduling situations. Meta-heuristic algorithms usually use a group of randomly generated populations to find solutions in the search space [7, 14]. Huang et al. [7] investigated the scheduling of real-time application on HMSs integrated with PMM; however, their method cannot achieve the given reliability target. Zhang et al. [14] proposed a bi-objective genetic algorithm (BOGA) to optimize energy and reliability for scheduling problem. However, the BOGA algorithm assumes that there is no memory contention when accessing tasks on the same core, which is not realistic, and its computation cost is high since it needs to sort the non-dominated solution.

Deng et al. [72] proposed an improved co-evolution multi-population ant colony optimization (ICMPACO) algorithm which uses the positive feedback mechanism and multi-population strategy; however, its search speed is slow. Song et al. [73] proposed a Harris Hawks optimization based on Gaussian mutation and cuckoo search strategy (GCHHO); however, it was not designed for scheduling problems, and its performance in solving task scheduling problems still needs to be improved. Some algorithms have been proposed previously to solve power and performance tradeoff optimization; however, these algorithms do not work well for large-scale problems [5, 12, 72], or have high time costs [5], or some tasks in the solution set do not meet the priority constraints [5]. Our proposed algorithm differs from the above algorithms in that we utilize an adaptive WOA-based algorithm combining with rescheduling CPNs to optimize the energy consumption under makespan and reliability constraints.

3 The models

In this section, we discuss the mathematical models of HMSs with DVFS.

We assume that the multiprocessor model studied in this paper has the following characteristics [12, 25]: (1) non-preemptive; (2) heterogeneous; (3) communications between processors are performed with different start time and bandwidth; (4) task duplication is forbidden; (5) each processor has an independent I/O unit that can perform communication and computation simultaneously [12, 25].

3.1 System model

The heterogeneity model of a computing system can be classified as processor-based heterogeneity model (PHM) and task-based heterogeneity model (THM) [74, 75]. In a PHM model, a processor executes the tasks at the same speed, no matter what type of tasks [74, 75] is. In a THM model, the execution speed of a processor executing a

task depends on how well the heterogeneous processor architecture matches the task features and requirements [74, 75]. In this paper, we assume a PHM model. Main notations used in this work are summarized in Table 1.

The system consists of a set of heterogeneous processors $P = \{P_1, P_2, \ldots, P_M\}$ that are fully interconnected with a high-speed network [5, 12, 74], where M represents the number of heterogeneous processors. Each processor is DVFS-enabled. Let $h(k)$ be the types number of the voltage supply levels for the k-th processor P_k [5, 12], and V_{kr} be the voltage corresponding to the r-th voltage supply level of P_k [5, 12]. Then, $V_k = (V_{k1}, \ldots, V_{kh(k)})$ is the voltage supply vector of P_k. When processor P_k is idle, the voltage supplied (i.e., $V_{kh(k)}$) is minimal [5, 12]. Specially, we denote f_{kr} as the frequency corresponding to the r-th voltage supply level of processor P_k,

Table 1 Some important notations

Notation	Description
N	Number of tasks
M	Number of processors
V_{kr}	The rth voltage supply level of processor P_k
P_{kr}	The processor P_k given supply voltage V_{kr}
W_{ikr}	The computation time of task T_i on P_k with V_{kr}
$C_w(T_i, T_j)$	The communication amount between T_i and T_j
$\mathrm{EFT}(T_i)$	The earliest finish time of T_i
$\mathrm{EST}(T_i)$	The earliest start time of T_i
$\mathrm{SL}(G)$	The schedule length of a task graph G
$E_{\mathrm{tot}}(G)$	The total energy consumption of G
FP	The failure probability of system
$Rn(T_i)$	The downward rank of T_i
Np	The population size
OIWOA	The WOA algorithm based on opposite learning and individual Selection strategy
OIWOAH	The OIWOA algorithm improved by rescheduling CPNs
QHA [5]	The quantum-inspired hyper-heuristics algorithm
QHAH	The QHA algorithm improved by rescheduling CPNs
ICMPACO [72]	The improved co-evolution multi-population ant colony Optimization algorithm
ICMPACOH	The ICMPACO algorithm improved by rescheduling CPNs
GACS [12]	The cuckoo search algorithm deploying Gaussian random walk and adaptive discovery probability
GACSH	The GACS algorithm improved by rescheduling CPNs
IWA [38]	The improved whale algorithm
IWAH	The IWA algorithm improved by rescheduling CPNs
GCHHO [73]	Harris Hawks optimization based on Gaussian mutation and cuckoo search strategy
GCHHOH	The GCHHO algorithm improved by rescheduling CPNs

and P_{kr} as the processor P_k given supply voltage V_{kr}. For example, Table 2 shows a modified voltage and frequency pairs, where the non-idle voltage index I_2 denotes that a processor with its voltage is 1.4 and its frequency is 0.8. For a system with four different types of processors as shown in Table 2, we can obtain that $V_{12} = 1.4$, $f_{23} = 0.8$, $h(2) = 7$ and $V_2 = (V_{21}, V_{22}, \ldots, V_{27}) = (1.5, 1.4, 1.3, 1.2, 1.1, 1.0, 0.9)$, and we can use P_{32} to represent the processor P_3 given supply voltage $V_{32} = 0.85$. When P_2 is idle, the voltage supplied $V_{27} = 0.9$ is the minimal voltage among the voltages of processor P_2.

3.2 Application model

In this work, a task graph $G = (T, E)$ can be represented by a Directed Acyclic Graph (DAG) consisted of a set of N tasks $T = \{T_1, \ldots, T_N\}$ [26], where E is composed of the edges which represent task precedence constrains [13]. The vertices of G represent tasks [26]. We assume that each task is executed sequentially without preemption in the same processor [25]. For a pair of dependent tasks T_i and T_j, if the execution of T_j depends on the output from the execution of T_i, then T_i is the predecessor of T_j, and T_j is the successor of T_i [76]. There is an entry task and an exit task in a DAG. The vertex weight of T_i is denoted as $W_v(T_i)$, which represents the computation amount of T_i [12]. The edge weight of T_i and T_j is denoted as $C_w(T_i, T_j)$, which represents the communication amount between task T_i and T_j [12, 76]. Our application model is similar to those in reference [26, 74]. Inspired by reference [26], we design an example of DAG, which is shown in Fig. 1. Inspired by references [26, 77, 78], we set the corresponding task weight values as shown in Fig. 1. Figure 1 shows a DAG with eleven tasks, where the weight 4 of T_8 represents the computation amount denoted by $W_v(T_8) = 4$, and the weight 10 of edge between T_1 and T_8 represents the communication amount denoted by $C_w(T_1, T_8) = 10$.

3.3 Schedule length model

The earliest finish time of task T_i on processor P_k given supply voltage V_{kr} is defined as [5, 12]

$$\mathrm{EFT}(T_i) = \mathrm{EST}(T_i) + W_{ikr}, \tag{1}$$

where $\mathrm{EFT}(T_i)$ represents the earliest finish time of T_i, $\mathrm{EST}(T_i)$ represents the earliest start time of T_i, and W_{ikr} represents the computation time of T_i on P_k given supply voltage V_{kr}. Then, W_{ikr} can be defined as [5, 12]

$$W_{ikr} = \frac{W_v(T_i)}{f_{kr}}. \tag{2}$$

The communication between processors only relies on message-passing [7]. If two tasks are assigned to the same processor, then the communication time is equal to zero since intraprocessor communication can be ignored [7]. Let $B(P_k, P_l)$ be data transfer rates between P_k and P_l, and $C_s(P_k)$ be the communication startup cost of P_k

Table 2 A modified voltage and frequency pairs of Ref. [4, 5, 12]

Level	Pair1			Pair2			Pair3			Pair4		
	Index	Voltage	Frequency	Index	Voltage	Frequency	Index	Voltage	Frequency	Index	Voltage	Frequency
0	I_1	1.75	1.0	I_4	1.5	1.0	I_{10}	2.2	1.0	I_{14}	1.5	1.0
1	I_2	1.4	0.8	I_5	1.4	0.9	I_{11}	1.9	0.85	I_{15}	1.2	0.8
2	I_3	1.2	0.6	I_6	1.3	0.8	I_{12}	1.6	0.65		0.9	0.5
3		0.9	0.4	I_7	1.2	0.7	I_{13}	1.3	0.5			
4				I_8	1.1	0.6		1.0	0.35			
5				I_9	1.0	0.5						
6					0.9	0.4						

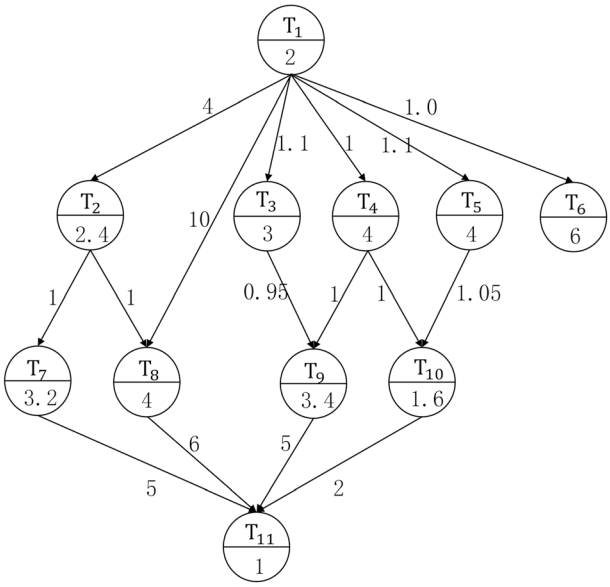

Fig. 1 A DAG containing 11 tasks

[74]. The communication time $C(T_i, T_j)$, which represents the time spent in transferring data from T_i (assigned to P_k) to T_j (assigned to P_l), can be expressed as [5, 12]

$$C(T_i, T_j) = \begin{cases} 0, & \text{if } \ k = l, \\ C_s(P_k) + \frac{C_w(T_i, T_j)}{B(P_k, P_l)}, & \text{if } \ k \neq l. \end{cases} \tag{3}$$

It can be seen from Eq. (3) that the communication time is taken into account if T_i and T_j are assigned to different processors, and it is considered to be zero if T_i and T_j are assigned on the same processor [12, 26].

The earliest start time of task T_i on processor P_k given supply voltage V_{kr} is defined as [4, 5, 12]

$$\mathrm{EST}(T_i) = \begin{cases} 0, & \text{if } \ T_i = T_{\text{entry}} \\ \max\{\mathrm{avl}(P_k), \max\limits_{T_j \in pr(T_i)} \{\mathrm{EFT}(T_j) \\ +C(T_i, T_j)\}\}, & \text{otherwise} \end{cases} \tag{4}$$

where $\mathrm{avl}(P_k)$ represents the earliest available time when P_k is ready for task schedule, $pr(T_i)$ is the immediate predecessor task set of task T_i, and T_{entry} represents the entry task [5, 12]. It can be seen from Eq. (4) that T_j is a predecessor of T_i [5, 12].

The schedule length (i.e., makespan) of G is denoted as $\mathrm{SL}(G)$, which is given by [5, 12]

$$\mathrm{SL}(G) = \max_{T_i \in T} \mathrm{EFT}(T_i). \tag{5}$$

3.4 Energy consumption model

Our energy consumption model is similar to the classic power model proposed in [4, 5, 12, 33]. The power consumption is dominated by dynamic power dissipation ϕ, which is defined as [4, 5, 12]

$$\phi = CV^2 f, \tag{6}$$

where C is the effective switched capacitance, V is the supply voltage, and f is the frequency [14]. Since $f \propto V$, so $\phi = \lambda V^3$, where λ represents a parameter that differs from one machine to another [5, 12]. For the energy consumption model, we follow the setting in Ref. [5, 12] and let the processor P_k work with the same fixed parameter λ_k. The energy consumption of each processor is the sum of its dynamic energy consumption and idle energy consumption.

Let $E_{\mathrm{dyn}}(ikr)$ be the dynamic energy consumption of T_i on P_k given supply voltage V_{kr}. Then, $E_{\mathrm{dyn}}(ikr)$ can be calculated as [4, 5]

$$E_{\mathrm{dyn}}(ikr) = \lambda_k V_{kr}^3 W_{ikr}, \tag{7}$$

Then, the dynamic energy consumption of all tasks executed can be expressed as [5, 12]

$$E_{\mathrm{dyn}} = \sum_{k=1}^{M} \sum_{T_i \in U_k} \lambda_k V_{kr}^3 W_{ikr}, \tag{8}$$

where U_k is the task set on P_k.

Let $E_{\mathrm{id}}(k)$ be the idle energy consumption of processor P_k , and $T(G)$ be the schedule length. Then, $E_{\mathrm{id}}(k)$ can be expressed as [5]

$$E_{\mathrm{id}}(k) = \left(\mathrm{SL}(G) - \sum_{T_i \in U_k} W_{ikr} \right) \lambda_k V_{kh(k)}^3, \tag{9}$$

where $V_{kh(k)}$ is the lowest supply voltage on P_k [5]. Thus, the total idle power consumption of all processors is expressed as [4, 5, 12]

$$E_{\mathrm{id}} = \sum_{k=1}^{M} \left(\left(\mathrm{SL}(G) - \sum_{T_i \in U_k} W_{ikr} \right) \lambda_k V_{kh(k)}^3 \right). \tag{10}$$

Finally, let E_{tot} be the total energy consumption of system. Then, it can be calculated as [5, 12]

$$E_{\mathrm{tot}} = E_{\mathrm{dyn}} + E_{\mathrm{id}}. \tag{11}$$

3.5 Reliability model

Transient faults are short-term faults that disappear without damaging device, and are caused by electromagnetic interference or cosmic ray radiations [7]. It usually occurs more frequently than permanent faults [13, 14]. In this work, we focus on transient faults. Task's reliability is the probability of its successfully execution in the absence of transient faults [7, 13, 14]. Zhu et al. in [79] proposed two different models for the fault rate: the linear model for frequency scaling (with fix voltage) and exponential model for voltage scaling. Traditionally, it has been recognized as an exponential relationship between the transient fault rate and the circuit's critical cost [80]. We use the exponential model proposed in [79] for our problem in this work.

Let ζ_k be the failure rate of the P_k with the frequency f_{kr}. Then, ζ_k can be expressed as

$$\zeta_k = \zeta_0 10^{\frac{d(f_{k,\max}-f_{kr})}{f_{k,\max}-f_{k,\min}}}, \tag{12}$$

where ζ_0 represents the average failure rate of the maximum frequency [13, 14]. d is a constant which represents the sensitivity of failure rates on frequency and voltage scaling [13, 14].

Let $R(T_i, P_k, f_{kr})$ be the reliability of the task T_i executed on P_k with the frequency f_{kr}. Then, $R(T_i, P_k, f_{kr})$ can be expressed as [8, 9, 13, 14, 81]

$$R(T_i, P_k, f_{kr}) = e^{-\zeta_k \cdot W_{ikr}}. \tag{13}$$

From Eqs. (12) and (13), it can be seen that system reliability is a monotonically increasing function of frequency on the same processor [13].

Let R_s be the system reliability, which represents the probability of successfully performing all tasks [13, 14]. Then, R_s can be calculated as follows [13, 14]

$$R_s = \prod_{i=1}^{N} R(T_i, P_k, f_{kr}). \tag{14}$$

3.6 Problem model

Let FP be the failure probability of system. Then, FP is equal to $1 - R_s$, i.e., $FP = 1 - R_s$.

Our scheduling problem of performance and reliability constrained energy optimization can be defined as:

$$\text{Minimize: } E_{\text{tot}}, \tag{15}$$

subject to:

$$\begin{aligned} \text{EFT}(T_i) &\leq \text{EST}(T_j), T_i \in pr(T_j), \\ \text{SL}(G) &\leq S_c, \\ \text{FP} &\leq \text{FP}_c, \end{aligned} \tag{16}$$

where S_c represents the time constraint, and FP_c represents the constraint of failure probability.

4 Algorithm design

In this section, we detail our proposed OIWOAH algorithm.

4.1 OIWOAH-based task scheduling algorithm

In this work, to effectively address the multi-constraint optimization problem for task scheduling on HMSs, we propose an effective whale optimization algorithm, which is built based on opposite learning and individual selection strategy and combines with a heuristic principle. The main idea of our OIWOAH algorithm is that we use an improved WOA algorithm to search for solution of task-to-resource mapping, determine the task order assignment on each processor by using a downward-ranking heuristic, and then reschedule CPNs to improve the performance of WOA. The OIWOAH-based algorithm can solve a wide range of DAG applications and search globally in the solution set without violating the precedence constraint of task. The outline of OIWOAH-based task scheduling algorithm is depicted in Algorithm 1.

Algorithm 1 OIWOAH scheduling algorithm

Require: ~~Parameters for OIWOAH and scheduling~~ The population size Np, maximum number of fitness evaluations FE_{max}, parameter of spiral shape γ, a DAG $G = (T, E)$ and a set of DVFS-enabled processors.
Ensure: A task schedule.
$g = 0, FE = 0$;
Initial the whales population X^0;
while the termination condition is not meet **do**
  $g = g + 1$;
  $X^g = X^{g-1}$;
  Generate the opposite population OX^g of X^g;
  $X^g = Selectbest(X^g, OX^g)$(Algorithm 2);
  $FE = FE + Np$;
  Update η by Eq.(29), and then update A, B, ζ and r_3;
  if $r_3 < 0.5$ **then**
    if $|A| < 1$ **then**
      Update the current position by Eq.(17);
    else
      Select a random x^g_{rand};
      Update the search agent by Eq.(25);
    end if
  else
    Update the search agent by Eq.(22);
  end if
  Check if any search agent has exceeded the search space and modified it;
  Sort individuals and ~~calculate $\frac{rk(i)}{Np}$~~ calculate the i-th individual's rank $rk(i)$ (i.e., the ranking of the i-th individual from the best to the worst in X^g);
  $FE = FE + Np$;
  for $i = 1 : Np$ **do**
    if $rand \leq \frac{rk(i)}{Np}$ **then**
      Generate a new mutant by Eq. (30);
      if $x_i^{'(g+1)}$ is better than $x_i^{(g+1)}$ **then**
        Replace $x_i^{'(g+1)}$ with $x_i^{(g+1)}$ and update X^g;
        $FE = FE + 1$;
      end if
    end if
  end for
  Update the best solution x_{best};
end while
Call Algorithm 3 to improve the best solution x_{best};
Return a near-optimal solution.

Algorithm 2 $Selectbest(X, OX)$

Calculate the fitness and constraint violation of population X;
Calculate the fitness and constraint violation of population OX;
for $i = 1$ to Np **do**
  if OX_i is better than X_i **then**
    $X_i = OX_i$;
  end if
end for

Specifically, we use the OIWOA algorithm to evolve the task-processor mapping solution and determine the priority queue in each processor by utilizing a widely used downward heuristic shown in Eq. (32). Before each evaluation of the fitness value and the constraint violation degree, we need to calculate the task priority by using Eq. (32) according to the mapping of tasks to processors.

We first initial the whales population X^0(line 2), use opposition-based learning to generate opposite population OX^0 of X^0(line 6), and then call Algorithm 2 to select best solution of X^0, OX^0 (line 7). In line 9, we update η by Eq. (29). Then, in line 10–20, we perform the WOA operator. Thereafter, we use individual selection strategy to select individuals (line 21–31). In line 25, we generate a new mutant by Eq. (30). If $x_i^{'(g+1)}$ is better than $x_i^{(g+1)}$, then it is retained; otherwise, it is discarded (line 26–29). The loop ends until the termination condition is met, and the best solution x_{best} of OIWOA is obtained. Finally, we reschedule the critical path nodes to improve the performance of OIWOA (line 34).

4.2 Whale optimization algorithm

In this section, we introduce the basic whale optimization algorithm. WOA algorithm is a meta-heuristic optimization that simulates the hunting behavior of humpback whales [35, 36, 38]. WOA algorithm has simple structure and few parameters, and it abstracts three whale behaviors: encircling prey, spiral bubble-net feeding maneuver and search for prey [35, 36, 38].

(1) Encircling prey. The WOA algorithm assumes that the current optimal solution is the position of the prey or the position closest to the prey, and the other whales will approach the optimal position. The formula for updating the position is defined as [35, 36, 38]

$$x^{g+1} = x_{\text{best}}^{g} - A \cdot D, \tag{17}$$

$$D = |B \cdot x_{\text{best}}^{g} - x^{g}|, \tag{18}$$

where $g + 1$ represents the current number of iterations, x_{best} represents the best individual, $A \cdot |B \cdot x_{\text{best}}^{g} - x^{g}|$ represents the surrounding step, and A, B are defined as

$$A = 2\eta \cdot r_1 - \eta, \tag{19}$$

$$B = 2 \cdot r_2, \tag{20}$$

where r_1, r_2 are two random number of interval [0, 1], and η is called the convergence factor, which is calculated as

$$\eta = 2 - 2g/G, \tag{21}$$

where G represents the maximum generations.

(2) Bubble-net attacking method. The WOA algorithm designed two different methods to describe the bubble net predation behavior of whales, namely the contraction surrounding mechanism and the spiral renewal location [35, 36, 38].

1. Shrinking encircling mechanism. The shrinking encircling mechanism is implemented by equations Eqs. (18) and (20). Then, the position update of the enclosing process is defined by Eq. (17).

2. Spiral updating position: the distance between the individual whale and the current optimal location is calculated, and the whale moves closer to the current best individual in a spiral movement. Its mathematical model can be expressed as [35, 36, 38]

$$x^{g+1} = x^{g}_{\text{best}} + D' e^{\gamma\zeta} \cos(2\pi\zeta), \tag{22}$$

$$D' = |x^{g}_{\text{best}} - x^{g}|, \tag{23}$$

where γ is a logarithmic spiral shape constant; and ζ is a random number of interval $[-1, 1]$ [35, 36].

The whale selects the contraction enveloping mechanism and the spiral update position to update the whale position with a 50% probability, which is defined as [35, 36, 38]

$$x^{g+1} = \begin{cases} x^{g}_{\text{best}} - A \cdot D, & r_3 < 0.5 \\ x^{g}_{\text{best}} + D' e^{\gamma\zeta} \cos(2\pi\zeta), & r_3 \geq 0.5 \end{cases} \tag{24}$$

where r_3 is a random number of interval [0,1].

3. Search for prey. When $A > 1$, whales randomly search for food. At this time, the whales are randomly searched according to each other's position, and the mathematical model can be expressed as [35, 36, 38]

$$x^{g+1} = x^{g}_{\text{rand}} - A \cdot |B \cdot x^{g}_{\text{rand}} - x^{g}|, \tag{25}$$

where x^{g}_{rand} is a randomly selected whale individual.

4.3 Whale optimization algorithm based on opposite learning and individual selection strategy (OIWOA)

To improve the global exploration and local exploitation abilities of basic WOA algorithm, we present an improved whale optimization algorithm based on opposite learning and individual selection strategy in this section.

(1) Opposition-based learning.

In the late iteration of basic WOA algorithm, the population keeps moving closer to a certain optimal individual area and thus loses the opportunity to explore other locations in the space. The loss of population diversity makes the basic WOA algorithm easy to fall into a local optimum. The quality of the population affects the accuracy and convergence speed of the algorithm [36, 73]. The diversity of the population can greatly improve the search performance of the WOA algorithm. Opposition-based learning is an intelligent technology which considers the candidate solutions and their corresponding opposite solutions simultaneously in the optimization process [83]. Some literatures [83, 84] show that the opposition-based learning strategy can expand the search range of the population and increase the diversity of algorithm, and selecting the better solutions from the current population and opposite population can improve the search efficiency of the algorithms. Thus, in this work,

we use the opposition-based learning strategy to enrich the diversity of the search space of WOA, and accelerate the convergence speed of WOA. The description of selecting the better solutions from the current population X^g and opposite population OX^g is depicted in Algorithm 2.

If z_i is a real number in the interval of $[a_i, b_i]$, then the reverse point oz_i of z_i is defined as follows:

$$oz_i = a_i + b_i - z_i. \tag{26}$$

Seif et al. [85] proposed a new opposition-based algorithm, in which a new reverse solution oz_i is given as follows:

$$oz_i = \begin{cases} \text{rand}(a_i, z_i), & z_i \geq M_i \\ \text{rand}(z_i, b_i), & z_i < M_i \end{cases} \tag{27}$$

where $M_i = (a_i + b_i)/2$.

In this paper, we use Eq. (27) proposed by Seif et al. to define reverse solution. The reverse solution $x_{i,j} \in [a_j, b_j]$ is in a search space with dynamic boundaries. A dynamic boundary can make the reverse solution in a shrinking search space. The dynamic boundary is expressed as

$$a_j = \min(x_{i,j}), \quad b_j = \max(x_{i,j}). \tag{28}$$

(2) Adaptive convergence factor.

In the basic WOA algorithm, the convergence factor η decreases linearly with the number of iterations. As the number of iterations increases, the individual quickly tends to reach the vicinity of the optimal solution. The change of linear decreasing strategy is too single, which results in its limited ability to adjust and adapt to the nonlinear and complex optimization process, and makes the basic WOA algorithm easy to fall into local extremes. Since the iterative evolution of the algorithm in the task graph search process using the WOA algorithm is complex and non-linearly changing, the linear decreasing strategy of η cannot make the algorithm well match the scheduling search process. If η decreases slowly after the algorithm has reached the vicinity of optimal solution, it will help the algorithm to search for the optimal solution. Thus, we propose a new nonlinear adaptive method to adjust the convergence factor η. We use a larger η in the early iteration to facilitate the global exploration ability and increase the diversity of the population, and make η decrease slowly from large to small to coordinate the balance between global search and local search, and finally utilize a smaller η in the later iteration to improve the local exploitation ability of the algorithm and accelerate the convergence speed. The new convergence factor η can be defined as follows:

$$\eta = 2\left(\frac{\text{FE}}{\text{FE}_{\max}}\right)^2 - \frac{4\text{FE}}{\text{FE}_{\max}} + 2, \tag{29}$$

where FE represents the current number of fitness evaluations, and $\text{FE}_{\max}$ represents the maximum number of fitness evaluations.

(3) Individual selection strategy. In order to prevent the algorithm from falling into the local optimum, inspired by [12], this paper uses an individual selection strategy to select individuals to improve the diversity of the population, which can improve the exploitation ability of individuals. First, the individuals are sorted by constraint optimization technology. Let $rk(i)$ be the rank in which the individual i is sorted from the best to the worst in the population, and $x_i^{'(g+1)}$ be a new mutant. Then, $x_i^{'(g+1)}$ can be defined as follows:

$$x_i^{'(g+1)} = \begin{cases} (1-\chi)x_i^{(g+1)} + \chi N(x_{\text{best}}^{(g+1)}, \xi), & r_4 \le \dfrac{rk(i)}{Np} \\ x_i^{(g+1)}, & \text{otherwise} \end{cases} \tag{30}$$

where $\xi = |(x_{\text{best}}^{(g+1)} - x_j^{(g+1)})|$, r_4 represents a random number of interval [0, 1], and χ is defined as: $\chi = 1 - (\frac{g}{g+1})^{10}$. If $x_i^{'(g+1)}$ is better than $x_i^{(g+1)}$, then it is retained; otherwise, it is discarded. In the early stage of evolution, it can ensure that the obtained mutant individual $x^{'(g+1)}$ is relatively different from the previous individual position, which is conducive to the global search of the population; and in the later stage of evolution, the search scale is small, and the Gauss distribution is used to make the population search around the optimal individual, which is helpful to improve the precision of the solution.

4.4 Coding scheme

The encoding mechanism of our algorithm consists of two sections: task-to-processor mapping and task order scheduling. We use the OIWOAH algorithm to evolve the mapping solutions. After the task-to-processor mapping is determined, the processor to each task is assigned and its voltage state is determined. Then, a downward ranking heuristic is used to calculate the task priorities in each processor, which guarantees the feasibility of each evolved solution.

In this section, we present the whale representation for task-to-resource mapping, and then show the policy of task priority calculation.

4.4.1 Whale representation

In this work, we orderly encode the non-idle voltage supply levels of all processors one by one [12], and associate each non-idle voltage supply level with its corresponding unique index. Let N_{nv} be the total number of non-idle voltage (NIV) states for a HMS. Then, $I = (I_1, \ldots, I_{N_{\text{nv}}})$ is the index vector for the HMS. As shown in Table 2, the total number of NIV states is 15, and the corresponding NIV index I_2 denotes that a processor with its voltage is 1.4 and its frequency is 0.8. The encoding of individual is chosen randomly from 1 to N_{nv}. Since the index of NIV state to which the task is mapped is a real number (see Fig. 2), inspired by [12, 86, 87], we convert it into an integer by using the round function.

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
0.6	1.3	3.8	9.4	5.2	13.7	1.5	9.6	10.9	14.3	0.3

Fig. 2 Whale's position

The dimension of the population individual is equal to the number of tasks, and each component of the individual corresponds to the NIV state to which the task is assigned, denoted by I_1 to $I_{N_{\text{nv}}}$. In the solution space, task T_i is assigned to NIV index I_k, where k can be calculated as

$$k = \text{round}\left(N_{\text{nv}} \cdot \frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}} + 0.5\right), \tag{31}$$

where $round(x)$ is a rounding function for x, and $x_{i,\min}$, $x_{i,\max}$ represents the minimum and maximum value of x_i. An example of whale's position is shown in Fig. 2, where $x_{i,\min}$ and $x_{i,max}$ are set as 0 and 15, respectively. The task-to-resource mapping corresponding to Fig. 2 is shown in Fig. 3. For example, task T_5 is assigned to a processor of Pair2 with the NIV index of I_6, i.e., task T_5 is assigned to the processor with its voltage equal to 1.3 and its frequency equals to 0.8.

4.4.2 Task priority calculation

After determining the task-to-resource mapping scheme, the formation of the final schedule still needs to determine the sequence of the tasks. Then, we can calculate the makespan, reliability, and energy consumption of each individual, and then evaluate the fitness of the individual. In this work, we use a widely used downward ranking heuristic to calculate task priority. Suppose that T_i is executed on P_k with V_{kr}, and $Rn(T_i)$ is the downward rank of T_i. Then, $Rn(T_i)$ can be defined as [88]:

$$Rn(T_i) = \max_{T_j \in pr(T_i)} (W_{jkr} + C(T_i, T_j) + Rn(T_j)), \tag{32}$$

where $pr(T_i)$ represents the immediate predecessors set of task T_i. Similar to literatures [77, 78], we assume a system with four processors with the following parameters: $C_s(P_1) = 0.001, C_s(P_2) = 0.0011, C_s(P_3) = 0.0012, C_s(P_4) = 0.0011, B(P_1, P_2) = 1.1, B(P_1, P_3) = 1.0, B(P_1, P_4) = 1.0, B(P_2, P_3) = 0.95, B(P_2, P_4) = 1.05, B(P_3, P_4) = 1.0, \lambda_1 = 12.1, \lambda_2 = 12.3, \lambda_3 = 13.9, \lambda_4 = 13.7, S_c = 18$ and $\text{FP}_c = 0.00000040$. Then, we can calculate the tasks priority of the task-to-processor mapping via Eq. (32). An example of the tasks priority of Fig. 3 from high to low is

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
I1	I2	I4	I10	I6	I14	I2	I10	I11	I15	I1

Fig. 3 Task-to-processor mapping

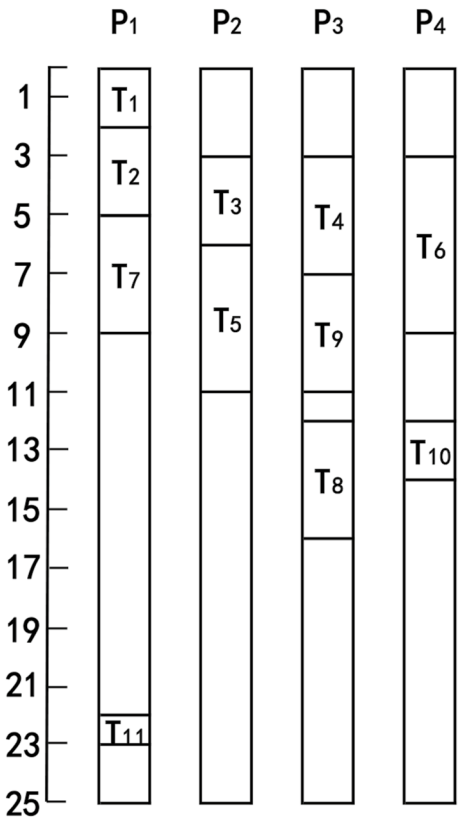

Fig. 4 A sample schedule of Fig. 3 with SL(G) = 23.0022, E_{tot} = 3.12877869E + 3, R_s = 0.99999981

{ T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_9, T_{10}, T_8, T_{11}}. Thereafter, we obtain the final schedule. Figure 4 shows a sample schedule of Fig. 3.

4.5 Constraint handling technique

Generally, a constrained optimization problem can be described as follows:

$$\begin{aligned} &\min y = f(X) \\ &s.t. \begin{cases} g_i(X) \le 0, & i = 1, \ldots, p \\ h_j(X) = 0, & j = p+1, \ldots, q \end{cases} \end{aligned} \tag{33}$$

where $X \in \Omega \subseteq S$ is a decision vector, Ω is feasible area, S is a search space, $f(X)$ is an objective function, $g_i(X) \le 0$ is the ith inequality constraint, and $h_j(X) = 0$ is the jth equality constraint.

The equality constraint in Eq. (33) can be transformed into inequality treatment, i.e., $|h_j(X)| - \epsilon \le 0, j = p+1, \ldots, q$, where ϵ represents the tolerance value of equality constraint, and generally ϵ is a small positive number. Therefore, the j-th

constraint violation degree of the individual $X_i(i = 1, \ldots, Np)$ in the population can be expressed as

$$G_j(X_i) = \begin{cases} \max\{g_j(X_i), 0\}, & i = 1, \ldots, p \\ \max\{|h_j(X_i)| - \epsilon, 0\}, & j = p+1, \ldots, q \end{cases} \tag{34}$$

where Np represents the population size.

Let $G_{j,\max}$ be the maximum value that the individuals of the population violate the j-th constraint. Then, $G_{j,\max}$ can be expressed as

$$G_{j,\max} = \max_{i=1,\ldots,Np}(G_j(X_i)), j = 1, \ldots, q. \tag{35}$$

Let $\psi(X_i)$ be the constraint violation degree of individual X_i. Then, $\psi(X_i)$ can be expressed as

$$\psi(X_i) = \frac{1}{q}\sum_{j=1}^{q}\frac{G_j(X_i)}{G_{j,\max}}, \ i = 1, \ldots, Np. \tag{36}$$

The authors in [89] proposed a constrained rank-based method, which sorts and assigns rank to each individual, i.e., the lower the rank, the better the quality of the solution [89, 90]. The idea of this constrained rank-based method is: a relative large constraint violation degree is tolerated to prevent the individual from converging to the local optimal value at the beginning of the iterations, and a small constraint violation degree is tolerated to promote the individual to converge to the global optimal value in the late evolutionary algorithm [89, 90].

Let $\varepsilon(k)$ be the level control function. Then, it can be expressed as [89, 90]

$$\varepsilon(k) = \begin{cases} \varepsilon(0)\left(1 - \frac{1}{K_c}\right)^{\rho}, & 0 < k < K_c \\ 0, & k \geq K_c \end{cases} \tag{37}$$

where $\varepsilon(0) = \psi(x_\beta)$, and x_β represents the top β-th individual and $\beta = 0.2Np$; K_c represents a generations control parameter, and ρ is a constraint control constant. With the increase in the number of iterations, the constraint level becomes smaller and smaller until it is 0, which makes the constraint violation degree larger in the early iteration and smaller in the later iteration.

Let $f_1(f_2)$ be the objective function of $X_1(X_2)$, $\psi_1(\psi_2)$ be the constraint violation degree of $X_1(X_2)$, and $\varepsilon \geq 0$. Then the comparison of ε-level between (f_1, ψ_1) and (f_2, ψ_2) can be defined as [89]:

$$(f_1, \psi_1) < (f_2, \psi_2) \Leftrightarrow \begin{cases} f_1 < f_2, & \psi_1, \psi_2 \leq \varepsilon \\ \psi_1 < \psi_2, & \text{otherwise} \end{cases} \tag{38}$$

$$(f_1, \psi_1) \leq (f_2, \psi_2) \Leftrightarrow \begin{cases} f_1 \leq f_2, & \psi_1, \psi_2 \leq \varepsilon \\ \psi_1 \leq \psi_2, & \text{otherwise} \end{cases} \tag{39}$$

4.6 Heuristic modification strategy

In [26], the authors proposed a heuristic principle to reschedule the critical path nodes to reduce the makespan; however, it only considered reducing completion time without considering energy consumption and adjusting voltage scales, which may not be able to minimize the system energy consumption to the greatest extent by adjusting voltage scales, and thus this strategy cannot be used directly in our problems [26]. Since the idle energy consumption can be calculated by Eq. (9), each processor consumes energy in idle time. Because the communication time of two nodes in the same processor is zero, we assign the critical path node to the processor where its parent node is located and adjust its frequency to reduce system completion time, maintain high system reliability and reduce energy consumption. Examples of schedules are given in Fig. 5. In these examples, we assume the system with four processors with the same parameters as in Sect. 4.4. The idle time of each processor in Fig. 5 is shorter than in Fig. 4. Since $S_c = 18$ and $\mathrm{FP}_c = 0.00000040$, we can see from Fig.5a, b that these two schedules both satisfy the makespan and reliability constraints. Thus, the schedule shown in Fig. 5b is better than that in Fig. 5a. In this work, we reschedule the critical path nodes to improve the performance of OIWOA.

Let $ft(T_k, P_{jr})$ be the finish time of T_k on P_{jr}. Then, Data Arrival Time (DAT) of T_i at P_j can be expressed as [26]:

$$\mathrm{DAT} = \max\{ft(T_k, P_{jr}) + C(T_i, T_k)\}; k = 1, \ldots, pn \tag{40}$$

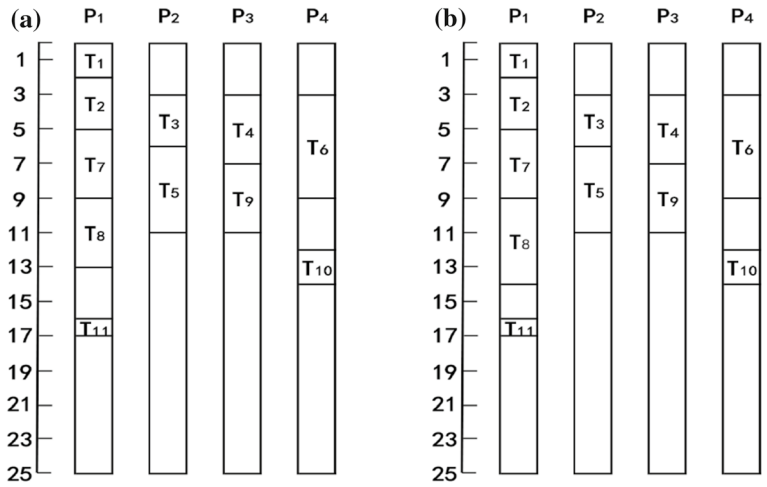

Fig. 5 Two example schedules after applying the reschedule of the CPNs on schedule of Fig. 4. **a** An improved schedule where T_8 is assigned to P_{11} with $\mathrm{SL}(G) = 17.0022$, $E_{\mathrm{tot}} = 2.54609425\mathrm{E}+3$, $R_s = 0.99999981$. **b** An improved schedule where T_8 is assigned to P_{12} with $\mathrm{SL}(G) = 17.0022$, $E_{\mathrm{tot}} = 2.45271250\mathrm{E}+3$, $R_s = 0.99999976$

where pn is the number of T_i's parents [26]. We call the parent task that maximizes Eq. (41) as the favorite predecessors of T_i [26], and denote it as fap(T_i, P_j).

A critical path is the path with the maximum execution time among all paths, and a node in critical path is called as critical path node [26]. First, we find a critical path of the graph. Let PT = fap(T_i, P_j), i.e., PT is scheduled to P_j. If the system reliability satisfies the reliability constraint, we reduce the energy consumption by moving critical path nodes to the processor where its parent node is located to reduce system completion time, maintain high system reliability and reduce energy consumption. If some frequencies in the frequency set FS make $Ic(T_i, P_{jr'}) \leq 0$ hold, then we assign T_i to P_j with the largest frequency in FS, where $Ic(T_i, P_{jr})$ represents the increased power consumption when T_i is scheduled from the currently assigned processor to a new processor P_{jr} [91]. If the system reliability is greater than the given reliability constraint, we should improve the system reliability. Since adopting the maximum frequency can significantly improve the system reliability and shorten the system completion time, we assign T_i to P_j with the maximum frequency $f_{j,max}$. The description of the modification is depicted in Algorithm 3.

Algorithm 3 Heuristic modification Routine

Require: A task schedule.
Ensure: A near-optimal result.
Find the critical path and generate a list of critical path node;
while the list of critical path node is not empty **do**
Delete T_i from the list;
Let $PT = fap(T_i, P_j)$ (i.e., PT is scheduled to P_j);
if $FP \leq FP_c$ **then**
if there exist some frequencies in FS such that $Ic(T_i, P_{jr'}) \leq 0$ hold **then**
Assign T_i to P_j with the highest frequency in FS;
end if
else
Assign T_i to P_j with the maximum frequency $f_{j,max}$;
end if
end while
return a near-optimal result.

4.7 Time and space complexities analysis

In the OIWOAH algorithm, the time complexity of evaluating the fitness function is $O(e * N_{\text{vsl}})$, where e represents the number of edges, and N_{vsl} represents the amount of the non-idle voltage supply levels. The time complexity of the heuristic modification principles is $O(N^2 + 2 * N + e)$, where N represents the number of tasks. Thus, the time complexity of the OIWOAH algorithm can be calculated as

$$\begin{aligned} O(\text{ OIWOAH}) &= O((2N + e * N_{\text{vsl}} + log(Np)) * Np) \\ &= O(e * N_{\text{vsl}} * Np), \end{aligned} \tag{41}$$

where Np represents the population size.

In the OIWOAH, we need an array of size N to store each individual. Thus, the space complexity of OIWOAH is $O(N * Np)$.

Table 3 lists the time and space complexities of six algorithms for task scheduling problem, where M represents the number of processors. Since $e < N^2$, the time complexity of the QHAH, GACSH, IWAH, GCHHOH and OIWOAH algorithms is lower than that of ICMPACOH algorithm. Since $M < N_{\text{vsl}}$, the space complexity of the QHAH algorithm is lower than that of OIWOAH algorithm, and larger than those of GACSH, IWAH, GCHHOH and OIWOAH algorithms.

5 Simulation and results

5.1 Experiment setup

The simulation are performed on the same personal computer with an Intel Core i7-3770 3.40 GHz CPU with a 12.0 GB RAM. We use Python 2.7 to evaluate the performance of algorithms.

Since gem5 has been widely used in computer architecture simulations and in the evaluation of new architectures for high performance computing systems [92], our experiments are performed on the gem5 simulator. We evaluate the accuracy of gem5 by comparing its results to those of real hardware, find the sources of errors in gem5 and discuss how to deal with those errors, and then validate the gem5 [92]. A set of fully interconnected heterogeneous processors which are DVFS-enabled will be simulated. Communications between processors are performed with different start time and bandwidth, and each processor has an independent I/O unit that can perform communication and computation simultaneously [12, 25]. Meanwhile, it is assumed that each task is executed sequentially without preemption in the same processor [12, 25]. In the experiment, we consider four different types of processors, and the number of different types of processors will be randomly generated uniformly [5, 12]. The parameter λ_k of processor P_k is generated randomly from a uniform distribution in the range [12,14]. The occurrence of transient fault follows a Poisson distribution, and the equation is defined in Eq. (12) [13], where $d = 3$ and $\zeta_0 = 10^{-9}$.

In this paper, we apply two sets of graphs to compare the performance of algorithms. They are real-world problems (Modified Molecular Dynamics Code (MMDC) and Fast Fourier Transformation (FFT) [4, 5, 12]), and randomly generated task graphs [12]. The parameters of randomly graph generator are set the same

Table 3 The time and space complexities of six algorithms for task scheduling problem

Algorithm	Time complexity	Space complexity
QHAH	$O(e * N_{\text{vsl}} * Np)$	$O(M * N * Np)$
ICMPACOH	$O(N^2 * N_{\text{vsl}} * Np)$	$O(N_{\text{vsl}} * N * Np)$
GACSH	$O(e * N_{\text{vsl}} * Np)$	$O(N * Np)$
IWAH	$O(e * N_{\text{vsl}} * Np)$	$O(N * Np)$
GCHHOH	$O(e * N_{\text{vsl}} * Np)$	$O(N * Np)$
OIWOAH	$O(e * N_{\text{vsl}} * Np)$	$O(N * Np)$

as [5], and the parameters of random graph generator are listed in Table 4. The graph height of a randomly generated graph is randomly generated from a uniform distribution with its mean value equal to $\frac{\sqrt{N}}{\alpha}$ [5, 12], where N represents the tasks number which is given in advance, and α represents a parallelization parameter [5, 12]. The average computation amount of each task T_i [5, 12], i.e., $\overline{w_v}$, is set randomly from a uniform distribution with a range of $[0, 2 \times \overline{w_g}]$ [5, 12], where $\overline{w_g}$ represents the average computation amount of the given graph [5, 12]. $W_v(T_i)$ is denoted as the computation amount of each task T_i on each processor P_k [25], and $W_v(T_i)$ is set randomly from the following range: $\overline{w_v} \times (1 - \frac{\delta}{2}) \leq W_v(T_i) \leq \overline{w_v} \times (1 + \frac{\delta}{2})$, where δ represents the range percentage of computation amount on processors [5, 12]. Let W_{ikr} be the processing time of task T_i on P_k given supply voltage V_{kr}. Then, W_{ikr} is calculated as $\frac{W_v(T_i)}{f_{kr}}$. The communication cost among tasks is set randomly from a uniform distribution with range $[0, 2 \times CCR \times \overline{w_g}]$ [5, 12], where CCR represents the ratio of communication to computation [5, 12].

As can be seen from the above, the larger α indicates that the parallelism of the DAG is better [5]. And the larger the δ, the greater the heterogeneity of the system [5]. A small CCR indicates that the DAG is a computationally intensive task graph, and a large CCR indicates that the DAG is a communication-intensive task graph [5].

QHA, ICMPACO [72], GACS [12], IWA [38] are four published algorithms to address the scheduling problem, and GCHHO [73] is a recently proposed meta-heuristic algorithm. They are suitable to be considered as baselines for comparisons in the experiments. To ensure fairness, we first use the same heuristic principle mentioned in Algorithm 3 to improve the performance of QHA, ICMPACO [72], GACS [12], IWA [38] and GCHHO [73], and then denote them as QHAH, ICMPACOH, GACSH, IWAH and GCHHOH, respectively. In Table 1, we can see some algorithm notations used in this paper. In our experiment, we compare our OIWOAH algorithm with the state-of-the-art algorithms, i.e., QHAH, ICMPACOH, GACSH, IWAH and GCHHOH. The constraint violation degree of individual in the compared algorithms is calculated as in Eq. (36). The QHAH and GACSH use the same constraint handling technique as their corresponding literatures, and the ICMPACOH, IWAH and GCHHOH algorithms use the same constraint handling technique as this paper. In the experiment, we first perform the random heuristic search strategy within the number of fitness evaluations and then use the heuristic strategy which reschedules the critical path nodes only one time.

Table 4 Parameters and their values of random graph generator

Parameter	Value	Description
N	{32, 64, 128, 256, 512}	The number of tasks
M	{4, 8, 16}	The number of processors
α	{0.5, 1, 2}	The shape parameter of the graph
δ	{0.1, 0.5, 0.75, 1}	The computation capacity heterogeneity factor
CCR	{0.1, 0.5, 1, 2, 10}	The communication to computation ratio

5.2 Comparison metrics

The energy consumption is a main performance measure of a scheduling algorithm on a DAG. Energy consumption ratio (ECR) is used as the primary comparison metric and defined as the normalized energy of the graphs to a lower bound. The ECR of an algorithm can be expressed as [5, 12]

$$\mathrm{ECR} = \frac{E}{\sum_{i=1}^{n} \min_{P_k \in P}(\lambda_k \times W_{ikh(k)} \times V_{kh(k)}^3)}, \tag{42}$$

where E represents the energy consumption of an algorithm with DVFS, and $V_{kh(k)}$ represents the minimal voltage supply of P_k. The denominator of Eq. (42) is the sum of the minimum power consumption for each task [5, 12].

Energy saving ratio (ESR) is another important performance metric for our comparison [12]. The parameter ESR is defined as [12, 33]

$$\mathrm{ESR} = \frac{E_{\mathrm{ECS}} - E}{E_{\mathrm{ECS}}}, \tag{43}$$

where E_{ECS} represents the energy consumption of all tasks in the ECS algorithm [12]. The makespan extension can be defined by: $\mathrm{SL}(G) \leq (1+\beta) * \mathrm{SL}_{\mathrm{best}}$, where $\mathrm{SL}_{\mathrm{best}}$ represents the best schedule length of the ECS algorithm, and β represents the makespan extension rates [12]. Failure probability extension can be defined by: $\mathrm{FP} \leq (1+\beta) * \mathrm{FP}_{\mathrm{best}}$, where $\mathrm{FP}_{\mathrm{best}}$ represents the best failure probability of the ECS algorithm. In this paper, we set β equal to 0, 0.1, 0.2, 0.3, 0.4, respectively [12].

5.3 Parameter setting

The optimization effect of algorithm is affected to some extent by the parameters. Our algorithm uses uniform parameter settings, and the parameters are repeatedly verified by a large number of experiments. To reduce the error caused by accidental errors, all experiment results are the statistic result of 50 independent running. We use the control variable method to obtain the parameters of the OIWOAH algorithm, i.e., we first fix other parameter values and then discuss the influence of a particular parameter on the algorithm. To ensure fairness, the population size and the maximum number of fitness evaluations of all algorithms are set to 30 and 30,000, respectively.

In some references [35, 36, 38], the logarithmic spiral shape's parameter γ is usually set as 1.0. The logarithmic spiral shape's parameter γ can greatly affects the performance of OIWOA algorithm. Next, we analyze the influence of the logarithmic spiral shape's parameter on the performance of the OIWOA algorithm. A set of candidate values $\{0.8, 0.9, 1.0, 1.1, 1.2\}$ are given for γ. We randomly select seven test instances, i.e., B1, B3, B6, B8, B9, B11 and B14, from 18 randomly generated test instances as shown in Table 5. Table 6 shows the statistical results of 50 independent

Table 5 Test instance

Instance	N	M	α	CCR	δ	S_c	FP_c
B1	32	8	1	1	0.5	160	2.0×10^{-4}
B2	64	8	1	1	0.5	240	4.0×10^{-4}
B3	128	8	1	1	0.5	300	9.6×10^{-4}
B4	128	8	1	1	1	300	9.6×10^{-4}
B5	256	8	1	1	0.5	700	2.4×10^{-3}
B6	256	8	0.5	1	0.5	1200	2.4×10^{-3}
B7	512	8	0.5	1	0.5	2200	6.0×10^{-3}
B8	512	8	1	1	0.5	1200	6.0×10^{-3}
B9	512	8	2	1	0.5	1100	6.0×10^{-3}
B10	512	8	1	1	0.1	1200	6.0×10^{-3}
B11	512	8	1	1	0.75	1200	6.0×10^{-3}
B12	512	8	1	1	1.0	1200	6.0×10^{-3}
B13	512	4	1	1	0.5	2200	6.0×10^{-3}
B14	512	16	1	1	0.5	900	6.0×10^{-3}
B15	512	8	1	0.1	0.5	1200	6.0×10^{-3}
B16	512	8	1	0.5	0.5	1200	6.0×10^{-3}
B17	512	8	1	2	0.5	1400	6.0×10^{-3}
B18	512	8	1	10	0.5	2600	6.0×10^{-3}

Table 6 Results of OIWOA algorithm with different γ values.

	Instance	B1	B3	B6	B8	B9	B11	B14
$\gamma = 0.8$	Best	2.3641	2.5403	3.2618	2.8813	3.2549	3.5731	4.0920
	Mean	2.6015	2.8795	3.5159	3.1393	3.5812	3.8290	4.3642
	Std	3.28E–2	4.08E–2	3.47E–2	3.79E–2	4.16E–2	4.76E–2	5.04E–2
$\gamma = 0.9$	Best	2.3268	2.4549	3.2052	2.8902	3.1343	3.5829	3.9283
	Mean	2.5605	2.8281	3.4550	3.0927	3.5055	3.8365	4.1962
	Std	2.79E–2	2.61E–2	4.03E–2	2.65E–2	2.54E–2	2.18E–2	3.84E–2
$\gamma = 1.0$	Best	**2.2749**	2.4658	**3.1696**	**2.8208**	3.1556	**3.4702**	**3.8402**
	Mean	**2.5033**	**2.7724**	**3.3842**	**3.0242**	**3.4703**	**3.7186**	**4.1020**
	Std	**2.09E–2**	2.87E–2	**1.94E–2**	**2.37E–2**	2.63E–2	2.66E–2	**3.15E–2**
$\gamma = 1.1$	Best	2.6736	2.8176	3.7394	3.3378	3.7214	4.1038	4.4708
	Mean	2.9421	3.2979	4.0307	3.6392	4.0744	4.3960	4.7756
	Std	4.22E–2	3.97E–2	3.01E–2	4.89E–2	5.61E–2	5.11E–2	6.37E–2
$\gamma = 1.2$	Best	2.7870	2.8363	3.8379	3.2937	3.6835	4.1216	4.6137
	Mean	3.0669	3.3230	4.1369	3.6171	4.0947	4.4173	5.0351
	Std	5.84E–2	5.06E–2	7.48E–2	5.17E–2	1.89E–1	6.03E–2	7.58E–2

The evaluation scores of our algorithm per test instance are marked in boldface if they achieve the best performance

runs of the OIWOA algorithm under different γ values. It can be seen from Table 6 that the performance of OIWOA with gamma of 1.0 is better than that with other γ values in many test instances, such as B1, B6, B8, B14. When γ is set as 1.0, the OIWOA achieves the best mean in seven test instances. Thus, in the OIWOA $\gamma = 1.0$ is more appropriate.

Other parameters of QHAH, ICMPACOH, GACSH, IWAH and GCHHOH are set the same with their corresponding references.

5.4 Computation time of the algorithms

In this part, we compare the running time of OIWOA with five state-of-the-art algorithms: QHA, ICMPACO, GACS, IWA and GCHHO. We tested four algorithms on a random generated graph of different tasks number. Figure 6 shows the average computation time of 50 runs under 2000 function evaluations. We can see from Fig. 6 that the average computation time of OIWOA algorithm is faster than those of QHA, ICMPACO, GACS, IWA and GCHHO by 15.35%, 12.90%, 6.67%, 2.19% and 4.26%, respectively. Particularly when solving large-scale problems, our algorithm takes less time than other algorithms. The reason lies in that compared to the QHA, ICMPACO, GACS, IWA and GCHHO algorithms, our OIWOA evolution strategy is relatively simpler and more efficient, has fewer adjustment parameters, and uses the opposition-based learning strategy which can accelerate the convergence speed.

5.5 Real-world application graphs

In this section, we evaluate the performance of OIWOAH by considering modified molecular dynamic code (MMDC) [5] and fast Fourier transformation (FFT) [4].

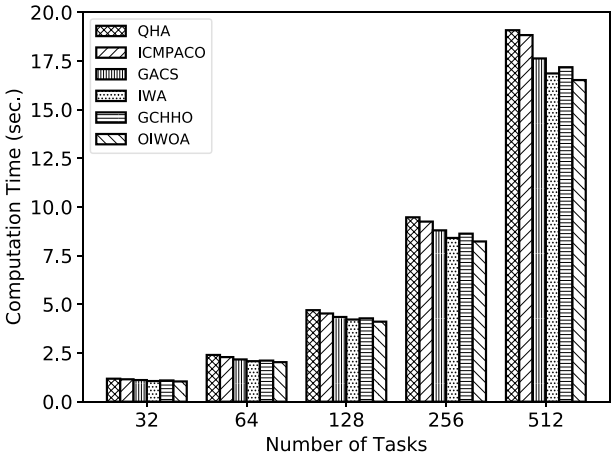

Fig. 6 Average computation time versus different tasks number

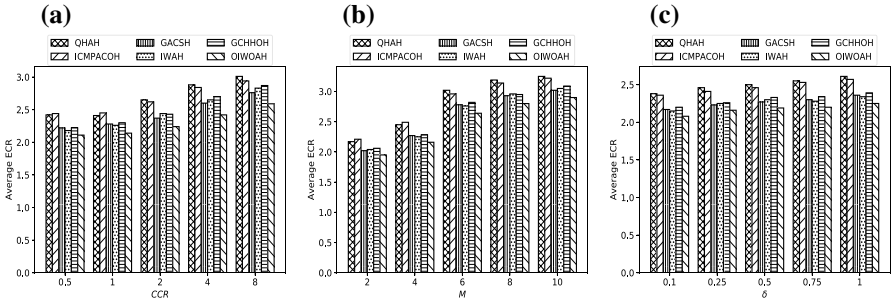

Fig. 7 Average ECRs of the algorithms for MMDC. **a** Average ECRs versus the values of CCR ($M = 4$, $\delta = 0.5$). **b** Average ECRs versus the values of M (CCR = 1.0, $\delta = 0.5$). **c** Average ECRs versus the values of δ (CCR = 1.0, $M = 4$)

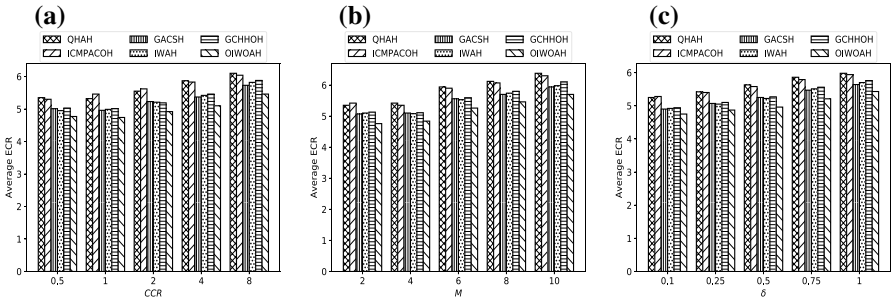

Fig. 8 Average ECRs of the algorithms for FFT. **a** Average ECRs versus the values of CCR ($M = 4$, $\delta = 0.5$). **b** Average ECRs versus the values of M (CCR = 1.0, $\delta = 0.5$). **c** Average ECRs versus the values of δ (CCR = 1.0, $M = 4$)

We apply ECS to the problem of MMDC or FFT, and obtain the results of makespan and reliability, which are taken as the constraint of QHAH, ICMPACOH, GACSH, IWAH, GCHHOH and OIWOAH, respectively.

We first use MMDC to test the search effectiveness of OIWOAH. The number of the nodes and the shape of the MMDC are both determined. Figure 7 depicts the average ECRs of the algorithms for MMDC with 95% confidence intervals. The results show that our WOAH outperforms QHAH, ICMPACOH, GACSH, IWAH and GCHHOH algorithms. The average ECRs of OIWOAH on graphs of modified molecular dynamic code are better than QHAH, ICMPACOH, GACSH, IWAH and GCHHOH algorithms by 12.82%, 12.13%, 4.78%, 5.20% and 6.47%, respectively. The reason behind this is that, compared with other algorithms, our OIWOAH algorithm uses an opposition-based learning strategy which can improve population diversity and global search capabilities.

We then use FFT to test the performance of OIWOAH. Our experiment randomly generates task graphs of FFT with 16, 32, 64, 128, and 256 nodes number, respectively. The algorithms will run 50 times independently, and then the average ECR

values are obtained. Figure 8 presents the average ECRs of the algorithms for FFT with 95% confidence intervals. We can see from Fig. 8a–c that the average ECR of OIWOAH algorithm outperforms QHAH and ICMPACOH. Compared with QHAH, ICMPACOH, GACSH, IWAH and GCHHOH, our proposed OIWOAH can on average reduce ECR by 10.88%, 10.61%, 4.71%, 4.97% and 5.81%, respectively. The reason lies in that OIWOAH algorithm maintains population diversity by using a new constraint handling technique which encourages the population to maintain diversity in the early stages of iteration, and to ensure that the algorithm gradually converges in the later stages of iteration.

5.6 Random generated application graphs

In Table 5, 18 randomly generated test instances are utilized to verify the performance of OIWOAH, where S_c represents the time constraint.

We first compare OIWOAH with QHAH, ICMPACOH, GACSH, IWAH and GCHHOH. The experimental results of randomly generated application graphs are shown in Table 7. The results of Table 7 show that OIWOAH can achieve better performance than QHAH, ICMPACOH, GACSH, IWAH and GCHHOH in many of the test instances, such as B5, B7, B8, B10, B11, B13 and B16. In most cases, the best and mean values of OIWOAH are better than other five algorithms.

To determine the performance of OIWOAH algorithm, we take B5 and B8 as representative to study their evolution curves. Figure 9a, b plots the convergence of energy consumption for processing the B5 and B8 test instances. Each data point in the graph represents the best result after every 500 fitness evaluations intervals, which is a statistical value of 50 independent runs. Figure 9 shows that the algorithms convergence speeds are rather different, i.e., the OIWOAH algorithm converges faster than QHAH, ICMPACOH, GACSH, IWAH and GCHHOH algorithm. As can be seen from Fig. 9, after applying the heuristic strategy to improve the QHAH, ICMPACOH, GACSH, IWAH, GCHHOH and OIWOAH algorithms, the performances of these algorithms are improved, and the final energy consumption achieved by OIWOAH algorithm is better than that of the other five algorithms. The main reason comes from the fact that OIWOAH algorithm uses an adaptive convergence factor which improve the population diversity.

The energy saving results vs. different tasks number and makespan extension rates are depicted in Figure 10a–c, where β represents the makespan extension rates. Figure 10 shows the energy saving results of algorithms with 95% confidence intervals. The tasks number is 400, 700 and 1000, respectively, and the energy saving results of QHAH, ICMPACOH, GACSH, IWAH, GCHHOH and OIWOAH algorithms tend to increase as β increases. We can see from Fig. 10 that the OIWOAH algorithms can save more energy than the QHAH, ICMPACOH, GACSH, IWAH and GCHHOH algorithms under different tasks number. For example, the OIWOAH's average ESR is better than QHAH, ICMPACOH, GACSH, IWAH and GCHHOH by: (11.76%, 10.24%, 4.53%, 3.53%, 5.21%), (10.10%, 8.68%, 4.97%, 4.00%, 6.62%), (11.82%, 10.41%, 5.74%, 6.71%, 8.02%), for tasks number of 400, 700 and

Table 7 Result of test instance

	Instance	B1	B2	B3	B4	B5	B6	B7	B8	B9
QHAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	2.7293	2.4096	2.9812	3.3659	2.9010	3.4936	3.7447	3.2268	3.5448
	Worst	3.1464	2.8617	3.4743	3.9704	3.3022	3.9482	4.2959	3.6752	4.4310
	Mean	2.9543	2.6524	3.2095	3.6803	3.1197	3.7456	4.0478	3.4731	4.0038
	Std	5.13E–2	3.87E–2	4.37E–2	4.14E–2	3.39E–2	4.76E–2	5.29E–2	4.57E–2	5.22E–2
ICMPACOH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	2.7820	2.4481	2.8947	3.3937	2.8891	3.6364	3.6348	3.2013	3.4946
	Worst	3.2141	2.8817	3.4220	3.9842	3.3080	4.0227	4.2587	3.7057	4.3129
	Mean	2.9864	2.6857	3.1718	3.7084	3.0801	3.8453	3.9297	3.4394	3.9193
	Std	4.72E–2	5.03E–2	3.76E–2	4.60E–2	3.67E–2	5.56E–2	3.83E–2	5.54E–2	4.13E–2
GACSH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	2.5070	2.2427	2.5847	3.1220	2.6927	3.4815	3.5465	3.0810	3.4013
	Worst	3.0035	2.6818	3.3243	3.7277	3.1427	3.8604	4.2462	3.5229	4.1719
	Mean	2.7748	2.4801	2.9728	3.4475	2.9018	3.6939	3.9157	3.32400	3.7547
	Std	3.77E–2	3.69E–2	3.54E–2	3.82E–2	3.26E–2	4.33E–2	3.79E–2	4.86E–2	3.89E–2
IWAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	2.5407	2.2353	2.6005	3.0843	2.7122	3.4739	3.5177	3.0680	3.4134
	Worst	3.0261	2.6429	3.3482	3.6949	3.2028	3.9280	4.3114	3.5846	4.1567
	Mean	2.7834	2.4391	2.9743	3.3908	2.9681	3.7009	3.9145	3.3288	3.7801
	Std	3.59E–2	3.28E–2	2.78E–2	3.57E–2	2.96E–2	3.03E–2	3.40E–2	5.02E–2	4.37E–2
GCHHOH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	2.5697	2.2718	2.6223	3.1983	2.7394	3.4982	3.4913	3.1330	3.4199
	Worst	3.1401	2.8003	3.4471	3.7126	3.3469	4.0778	4.3714	3.6827	4.2206
	Mean	2.8376	2.5527	3.0170	3.4803	3.0296	3.8033	3.9602	3.4218	3.8410
	Std	3.46E–2	2.94E–2	2.85E–2	3.21E–2	3.49E–2	4.56E–2	4.27E–2	4.17E–2	3.64E–2

Table 7 (continued)

	Instance	B1	B2	B3	B4	B5	B6	B7	B8	B9
OIWOAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	**2.3672**	**2.1844**	**2.5530**	3.0936	**2.5964**	**3.2968**	**3.3163**	**2.9517**	**3.2595**
	Worst	**2.8318**	2.6496	**3.2767**	**3.6314**	**3.1678**	**3.7716**	**4.0983**	**3.3818**	**3.9403**
	Mean	**2.6090**	**2.4067**	**2.9003**	**3.3804**	**2.8995**	**3.5186**	**3.6915**	**3.1520**	**3.6220**
	Std	3.65E–2	**2.36E–2**	3.28E–2	3.46E–2	**2.07E–2**	3.24E–2	**2.47E–2**	**3.02E–2**	4.07E–2
	Instance	B10	B11	B12	B13	B14	B15	B16	B17	B18
QHAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	3.4340	3.8288	4.0929	2.9500	4.4088	3.3471	3.5764	3.7828	5.7173
	Worst	4.0612	4.3946	4.5917	3.3666	4.8755	3.9097	4.4815	4.5501	6.6970
	Mean	3.7703	4.1423	4.3694	3.1728	4.6705	3.6026	4.0522	4.1913	6.1771
	Std	5.75E–2	6.33E–2	3.02E–1	5.07E–2	5.37E–2	4.87E–2	5.69E–2	5.13E–2	4.28E–1
ICMPACOH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	3.5716	3.7071	3.9038	3.1039	4.3772	3.4204	3.5790	3.7130	5.6718
	Worst	4.1381	4.4206	4.5173	3.4412	4.9478	3.8647	4.3376	4.5116	6.4723
	Mean	3.8301	4.0856	4.2369	3.2528	4.6940	3.6687	3.9767	4.1023	6.0421
	Std	6.05E–2	4.46E–2	4.89E–2	4.63E–2	2.30E–1	5.34E–2	4.65E–2	4.30E–2	2.49E–1
GACSH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	3.4533	3.6341	3.9256	2.9079	4.1647	3.3039	3.4172	3.5848	5.3349
	Worst	4.0982	4.2965	4.4496	3.2903	4.6050	3.7630	4.0550	4.2485	6.2402
	Mean	3.7916	3.9812	4.1623	3.0791	4.3701	3.5194	3.7588	3.9310	5.8122
	Std	4.30E–2	3.86E–2	4.09E–2	4.11E–2	3.35E–2	3.76E–2	4.05E–2	5.63E–2	4.80E–2

Table 7 (continued)

	Instance	B10	B11	B12	B13	B14	B15	B16	B17	B18
IWAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	3.4725	3.6490	3.8785	2.9453	4.1573	3.2876	3.4589	3.5912	5.3566
	Worst	4.0834	4.3363	4.4540	3.2102	4.5263	3.7136	3.9399	4.2865	6.2547
	Mean	3.7730	3.9926	4.1675	3.0877	4.3418	3.5006	3.6994	3.9388	5.8057
	Std	4.58E–2	4.17E–2	3.63E–2	4.28E–2	2.87E–2	3.45E–2	3.84E–2	3.60E–2	3.90E–2
GCHHOH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	3.4826	3.6969	3.9301	2.9747	4.1630	3.3548	3.4454	3.5797	5.4029
	Worst	4.1455	4.3978	4.4453	3.1967	4.5439	3.8005	3.9790	4.4049	6.4296
	Mean	3.8250	4.0708	4.1635	3.0890	4.3739	3.5956	3.7358	4.0177	5.8907
	Std	3.49E–2	3.57E–2	4.17E–2	3.84E–2	3.16E–2	4.31E–2	4.28E–2	4.54E–2	4.93E–2
OIWOAH	Feasible rate	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Best	**3.3403**	**3.5961**	**3.7290**	**2.7562**	**3.9877**	**3.0683**	**3.2556**	**3.3644**	**5.0743**
	Worst	**3.9234**	**4.1485**	**4.2853**	**3.1637**	**4.5127**	**3.5093**	**3.7904**	**3.9738**	**6.0302**
	Mean	**3.6230**	**3.8820**	**4.0241**	**2.9410**	**4.2364**	**3.2309**	**3.5117**	**3.6458**	**5.5810**
	Std	**2.21E–2**	**3.27E–2**	3.67E–2	**3.54E–2**	3.43E–2	3.59E–2	**2.30E–2**	3.78E–2	4.09E–2

The evaluation scores of our algorithm per test instance are marked in boldface if they achieve the best performance

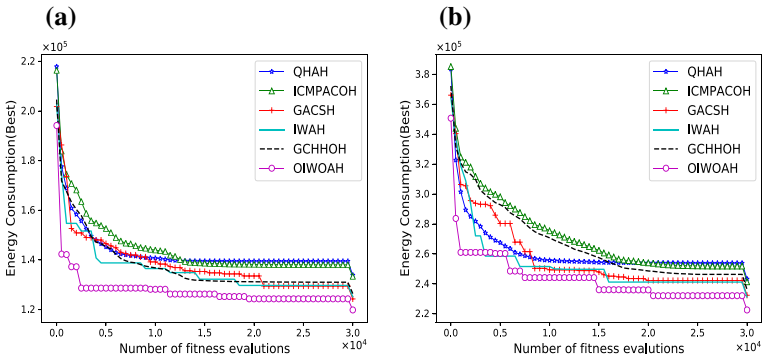

Fig. 9 **a** Best energy consumption simulation of B5; **b** Best energy consumption simulation of B8

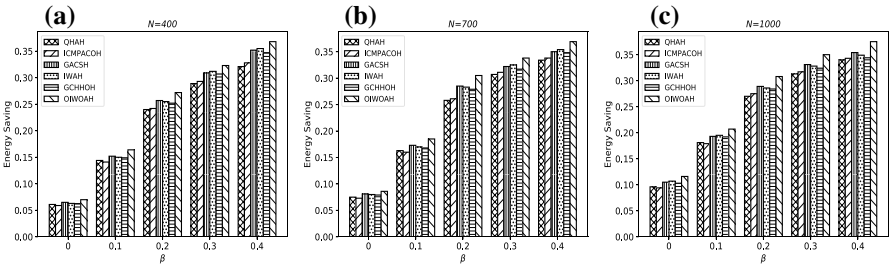

Fig. 10 Energy saving of the algorithms versus different tasks number and makespan extension rates ($M = 30, \text{CCR} = 0.5, \delta = 0.5$)

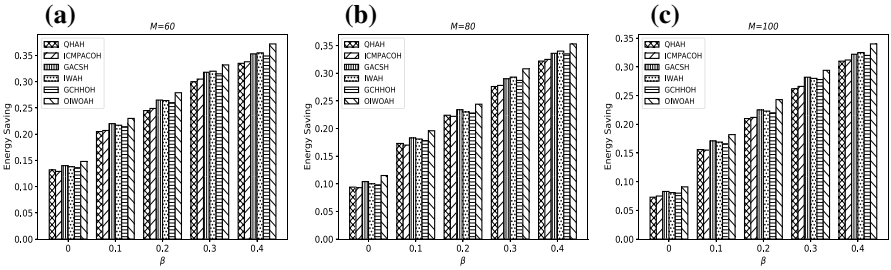

Fig. 11 Energy saving of the algorithms versus different processors number and makespan extension rates ($N = 600, \text{CCR} = 0.5, \delta = 0.5$)

1000, respectively, when $\beta = 0.3$. This shows that our OIWOAH algorithm has strong optimization ability and robustness when solving large-scale optimization problems.

Figure 11 depicts the results of the energy saving on different large-scale computing nodes with 95% confidence intervals. The ESR of OIWOAH algorithm is better than the QHAH, ICMPACOH, GACSH, IWAH and GCHHOH by: (11.04%,

10.06%, 5.38%, 4.79%, 6.59%), (9.63%, 8.62%, 5.06%, 3.82%, 5.37%) and (9.68%, 8.97%, 5.59%, 4.62%, 6.25%), for processors number of 60, 80 and 100, respectively, when $\beta = 0.4$. We can see from Fig. 11 that our OIWOAH algorithm outperforms the other five algorithms. This is because that the individual selection strategy adopted by our OIWOAH algorithm can maintain the diversity of the population and improve the exploitation ability of individuals, which can improve the search accuracy.

6 Conclusion

In this paper, we study the problem of power consumption and performance tradeoff optimization on HMSs with DVFS. Since the task scheduling issue is an NP-hard problem, we propose an improved whale optimization algorithm which combines a heuristic modification. We first propose an improved WOA algorithm deploying opposite learning and individual selection strategy, which increases the diversity of population and has stronger global search ability. Then we use the OIWOA algorithm to evolve the task-to-processor mapping solution, and apply a constrained rank-based method which retains some infeasible individuals in the population to maintain population diversity. Finally, we attempt to redistribute the critical path node to further improve the performance of OIWOA. A large number of real-world graphs and random experimental results are used to evaluate our OIWOAH algorithm. Our experiment results are statistical values of 50 independent runs, which show that our algorithm is not only efficient in searching, but also superior in performance to the state-of-art algorithms.

In the future, we intend to consider some effective meta-heuristic algorithms to solve problem of optimizing energy consumption for reliability-aware scheduling problem. Moreover, we plan to consider task scheduling problem incorporating with data allocation.

Acknowledgements Supported by Key-Area Research and Development Plan of Guangdong Province #2020B010164003 and National Key R&D Program of China Project #2017YFB0203201.

References

1. Liu X, Sun J, Zheng L, Wang S, Liu Y, Wei T (2021) Parallelization and optimization of NSGA-II on sunway taihulight system. IEEE Trans Parallel Distrib Syst 32(4):975–987
2. Taheri G, Khonsari A, Entezari-Maleki R, Sousa L (2020) A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems. Appl Soft Comput 1–14
3. Venkatachalam V, Franz M (2005) Power reduction techniques for microprocessor systems. ACM Comput Surv (CSUR) 37(3):195–237
4. Lee YC, Zomaya AY (2011) Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans Parallel Distrib Syst 22(8):1374–1381
5. Chen S, Li Z, Yang B, Rudolph G (2016) Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 27(6):1796–1810

6. Safari M, Khorsand R (2018) Pl-dvfs: combining power-aware list-based scheduling algorithm with dvfs technique for real-time tasks in cloud computing. J Supercomput 74(10):5578–5600
7. Huang K, Jiang X, Zhang X, Yan R, Wang K, Xiong D, Yan X (2018) Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems. IEEE Access 6:57614–57630
8. Xiao X, Xie G, Xu C, Fan C, Li R, Li K (2018) Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems. J Comput Sci 26:344–353
9. Zhou J, Cao K, Cong P, Wei T, Chen M, Zhang G, Yan J, Ma Y (2017) Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms. J Syst Softw 133:1–16
10. Xie G, Zeng G, Xiao X, Li R, Li K (2017) Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems. IEEE Trans Parallel Distrib Syst 28(12):3426–3442
11. Tang X, Liao X, Zheng J, Yang X (2018) Energy efficient job scheduling with workload prediction on cloud data center. Clust Comput 21(3):1581–1593
12. Deng Z, Yan Z, Huang H, Shen H (2020) Energy-aware task scheduling on heterogeneous computing systems with time constraint. IEEE Access 8:23936–23950
13. Zhang L, Li K, Xu Y, Mei J, Zhang F, Li K (2015) Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. Inf Sci 319:113–131
14. Zhang L, Li K, Li C, Li K (2017) Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf Sci 379:241–256
15. Tang X, Shi W, Wu F (2020) Interconnection network energy-aware workflow scheduling algorithm on heterogeneous systems. IEEE Trans Ind Inf 16(12):7637–7645
16. Tang X, Li X, Fu Z (2017) Budget-constraint stochastic task scheduling on heterogeneous cloud systems. Concurr Comput Pract Exp 29(19):e4210
17. Tang X, Li K, Liao G (2014) An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems. Clust Comput 17(4):1413–1425
18. Quan Z, Wang Z, Ye T, Guo S (2020) Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 31(5):1165–1182
19. Muhuri PK, Biswas SK (2020) Bayesian optimization algorithm for multi-objective scheduling of time and precedence constrained tasks in heterogeneous multiprocessor systems. Appl Soft Comput 1–27
20. Djigal H, Feng J, Lu J, Ge J (2021) IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 32(5):1057–1071
21. Chen J, Du C, Han P, Du X (2019) Work-in-progress: non-preemptive scheduling of periodic tasks with data dependency upon heterogeneous multiprocessor platforms. In: IEEE real-time systems symposium, RTSS 2019, Hong Kong, SAR, China, December 3–6, 2019. IEEE, pp 540–543
22. Aldegheri S, Bombieri N, Patel HD (2020) On the task mapping and scheduling for dag-based embedded vision applications on heterogeneous multi/many-core architectures. In: 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020, Grenoble, France, Mar 9–13, 2020. IEEE, pp 1003–1006
23. Hu Y, Li J, He L (2020) A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints. Neural Comput Appl 32(10):5681–5693
24. Zhang L, Zhou L, Salah A (2020) Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. Inf Sci 531:31–46
25. Wen Y, Xu H, Yang J (2011) A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. Inf Sci 181(3):567–581
26. Omara FA, Arafa MM (2010) Genetic algorithms for task scheduling problem. J Parallel Distrib Comput 70(1):13–22
27. Gu Q, Hao X (2018) Adaptive iterative learning control based on particle swarm optimization. J Supercomput 3615–3622
28. Kansal S, Kumar H, Kaushal S, Sangaiah AK (2020) Genetic algorithm-based cost minimization pricing model for on-demand iaas cloud service. J Supercomput 76(3):1–26
29. Alazzam H, Alhenawi E, Alsayyed RMH (2019) A hybrid job scheduling algorithm based on tabu and harmony search algorithms. J Supercomput 75(12):7994–8011

30. Asghari A, Sohrabi MK, Yaghmaee F (2021) Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm. J Supercomput 77(3):2800–2828
31. Alboaneen DA, Tianfield H, Zhang Y, Pranggono B (2021) A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. Future Gener Comput Syst 115:201–212
32. Deng Z, Shen H, Cao D, Yan Z, Huang H (2021) Task scheduling on heterogeneous multiprocessor systems through coherent data allocation. Concurr Comput Pract Exp 1–19
33. Hu Y, Liu C, Li K, Chen X, Li K (2017) Slack allocation algorithm for energy minimization in cluster systems. Future Gener Comput Syst 74:119–131
34. Zhao B, Aydin H, Zhu D (2010) On maximizing reliability of real-time embedded applications under hard energy constraint. IEEE Trans Ind Inf 6(3):316–328
35. Mirjalili SM, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
36. Gharehchopogh FS, Gholizadeh H (2019) A comprehensive survey: whale optimization algorithm and its applications. Swarm Evolut Comput 48:1–24
37. Chu D, Chen H, Wang X (2019) Whale optimization algorithm based on adaptive weight and simulated annealing. Acta Electr Sin 47(5):992–999
38. Abdel-Basset M, El-Shahat D, Deb K, Abouhawwash M (2020) Energy-aware whale optimization algorithm for real-time task scheduling in multiprocessor systems. Appl Soft Comput 93:106349
39. Li K, Tang X, Li K (2013) Energy-efficient stochastic task scheduling on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 25(11):2867–2876
40. Tang X, Fu Z (2020) CPU-GPU utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems. IEEE Access 8:58948–58958
41. Paul S, Chatterjee N, Ghosal P, Diguet J (2021) Adaptive task allocation and scheduling on noc-based multicore platforms with multitasking processors. ACM Trans Embed Comput Syst 20(1):4:1-4:26
42. Salami B, Noori H, Naghibzadeh M (2021) Fairness-aware energy efficient scheduling on heterogeneous multi-core processors. IEEE Trans Comput 70(1):72–82
43. Goubaa A, Khalgui M, Li Z, Frey G, Zhou M (2020) Scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems. Inf Sci 520:86–104
44. Ge R, Feng X, Cameron KW (2005) Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In: Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference. IEEE, pp 34–34
45. Li K (2008) Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed. IEEE Trans Parallel Distrib Syst 19(11):1484–1497
46. Li K (2016) Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels. J Parallel Distrib Comput 95:15–28
47. Nesmachnow S, Dorronsoro B, Pecero JE, Bouvry P (2013) Energy-aware scheduling on multicore heterogeneous grid computing systems. J Grid Comput 11(4):653–680
48. Li K, Tang X, Li K (2014) Energy-efficient stochastic task scheduling on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 25(11):2867–2876
49. Niu J, Liu C, Gao Y, Qiu M (2014) Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems. IEEE Trans Parallel Distrib Syst 25(8):2043–2052
50. Li D, Wu J (2015) Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. IEEE Trans Parallel Distrib Syst 26(3):810–823
51. Mezmaz M, Melab N, Kessaci Y, Lee YC, Talbi E-G, Zomaya AY, Tuyttens D (2011) A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J Parallel Distrib Comput 71(11):1497–1508
52. Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W (2015) Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Trans Parallel Distrib Syst 1:1
53. Zhang Y, Wang Y, Tang X, Yuan X, Xu Y (2018) Energy-efficient task scheduling on heterogeneous computing systems by linear programming. Concurr Comput Pract Exp 30(19):e4731
54. Thammawichai M, Kerrigan EC (2018) Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. Real Time Syst 54(1):132–165
55. Tang Z, Qi L, Cheng Z, Li K, Khan SU, Li K (2016) An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. J Grid Comput 14(1):55–74
56. Li K (2012) Energy efficient scheduling of parallel tasks on multiprocessor computers. J Supercomput 60(2):223–247

57. Xie G, Zeng G, Chen Y, Bai Y, Zhou Z, Li R, Li K (2020) Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. IEEE Trans Serv Comput 13(5):871–886
58. Girault A, Kalla H (2009) A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Trans Dependable Secure Comput 6(4):241–254
59. Dongarra JJ, Jeannot E, Saule E, Shi Z (2007) Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures. ACM, pp 280–288
60. Chen C-Y (2015) Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems. IEEE Trans Parallel Distrib Syst 27(2):521–532
61. Wang S, Li K, Mei J, Xiao G, Li K (2017) A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems. J Grid Comput 15(1):23–39
62. Jeannot E, Saule E, Trystram D (2012) Optimizing performance and reliability on heterogeneous parallel systems: approximation algorithms and heuristics. J Parallel Distribut Comput 72(2):268–280
63. Li R, Yu H, Jiang W, Ha Y (2020) Dvfs-based scrubbing scheduling for reliability maximization on parallel tasks in sram-based fpgas. In: 57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20–24, 2020. IEEE, pp 1–6
64. Zhang L, Li K, Li K, Xu Y (2016) Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems. Int J Electr Power Energy Syst 78:499–512
65. Zhu D, Aydin H (2009) Reliability-aware energy management for periodic real-time tasks. IEEE Trans Comput 58(10):1382–1397
66. Zhao B, Aydin H, Zhu D (2013) Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. ACM Trans Des Autom Electr Syst (TODAES) 18(2):23
67. Zhang L, Li K, Zheng W, Li K (2018) Contention-aware reliability efficient scheduling on heterogeneous computing systems. IEEE Trans Sustain Comput 3(3):182–194
68. Kumar N, Mayank J, Mondal A (2020) Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system. IEEE Trans Parallel Distrib Syst 31(4):871–885
69. Huang J, Li R, Jiao X, Jiang Y, Chang W (2020) Dynamic dag scheduling on multiprocessor systems: reliability, energy, and makespan. IEEE Trans Comput Aided Des Integr Circuits Syst 39(11):3336–3347
70. Hassan HA, Salem SA, Saad EM (2020) A smart energy and reliability aware scheduling algorithm for workflow execution in dvfs-enabled cloud environment. Future Gener Comput Syst 112:431–448
71. Abdi A, Girault A, Zarandi HR (2019) ERPOT: a quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. IEEE Trans Parallel Distrib Syst 30(10):2193–2210
72. Deng W, Xu J, Zhao H (2019) An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE Access 7:20281–20292
73. Song S, Wang P, Heidari AA, Wang M, Zhao X, Chen H, He W, Xu S (2021) Dimension decided harris hawks optimization with gaussian mutation: balance analysis and diversity patterns. Knowl Based Syst 215:106425
74. Xu Y, Li K, Hu J, Li K (2014) A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf Sci 270:255–287
75. Xu Y, Li K, He L, Zhang L, Li K (2015) A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 26(12):3208–3222
76. Manudhane KA, Wadhe A (2013) Comparative study of static task scheduling algorithms for heterogeneous systems. Int J Comput Sci Eng 5(3):166
77. Veeravalli B, Li X, Ko CC (2000) On the influence of start-up costs in scheduling divisible loads on bus networks. IEEE Trans Parallel Distrib Syst 11(12):1288–1305
78. Mingsheng S (2008) Optimal algorithm for scheduling large divisible workload on heterogeneous system. Appl Math Model 32(9):1682–1695
79. Zhu D, Melhem RG, Mossé D (2004) The effects of energy management on reliability in real-time embedded systems. In: 2004 International Conference on Computer-Aided Design, ICCAD 2004, San Jose, CA, USA, Nov 7–11, 2004. IEEE Computer Society/ACM, pp 35–40

80. Izosimov V, Pop P, Eles P, Peng Z (2005) Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In: 2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 Mar 05, Munich, Germany. IEEE Computer Society, pp 864–869
81. Wang S, Li K, Mei J, Xiao G, Li K (2017) A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems. Grid Comput 15(1):23–39
82. Tizhoosh HR (2005) Opposition-based learning: a new scheme for machine intelligence, vol 1. pp 695–701
83. Choi TJ, Togelius J, Cheong Y (2021) A fast and efficient stochastic opposition-based learning for differential evolution in numerical optimization. Swarm Evolut Comput 60:100768
84. Xu Y, Yang Z, Li X, Kang H, Yang X (2020) Dynamic opposite learning enhanced teaching-learning-based optimization. Knowl Based Syst 188:104966
85. Seif Z, Ahmadi MB (2015) An opposition-based algorithm for function optimization. Eng Appl Artif Intell 37:293–306
86. Kaur P, Mehta S (2017) Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm. J Parallel Distrib Comput 101:41–50
87. Qiu X, Hu Y, Li B (2016) Multiprocessor task scheduling based on improved differential evolution algorithm. Control Decis 31(2):217–224
88. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
89. Cheng L, Xing YJ, Ren MF, Xie G, Chen J (2018) Multipath estimation algorithm using ϵ constrained rank-based differential evolution. Acta Electr Sin 46(1):167–174
90. Takahama T, Sakai S (2012) Efficient constrained optimization by the ϵ constrained rank-based differential evolution. In: Proceedings of the IEEE congress on evolutionary computation, CEC 2012, Brisbane, Australia, June 10–15, 2012. IEEE, pp 1–8
91. Wang Y, Li K, Chen H, He L, Li K (2014) Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints. IEEE Trans Emerg Top Comput 2(2):134–148
92. Akram A, Sawalha L (2019) Validation of the gem5 simulator for x86 architectures. In: 2019 IEEE/ACM performance modeling, benchmarking and simulation of high performance computer systems, PMBS@SC 2019, Denver, CO, USA, Nov 18, 2019. IEEE, pp 53–58