Hindawi
International Journal of Aerospace Engineering
Volume 2024, Article ID 2122895, 14 pages
https://doi.org/10.1155/2024/2122895

Research Article

Online Energy-Aware Scheduling for Deadline-Constrained Applications in Distributed Heterogeneous Systems

Yifan Liu[1], **Chengelie Du**[2], **Jinchao Chen**[2] and **Xiaoyan Du**[2]

[1]*School of Software, Northwestern Polytechnical University, Xi'an 710072, China*
[2]*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China*

Correspondence should be addressed to Jinchao Chen; cjc@nwpu.edu.cn

Received 18 July 2023; Revised 12 March 2024; Accepted 26 March 2024; Published 2 May 2024

Academic Editor: Shunan Wu

In the current computing environment, the significance of distributed heterogeneous systems has gained prominence. The research on scheduling problems in distributed systems that consider energy consumption has garnered substantial attention due to its potential to enhance system stability, achieve energy savings, and contribute to environmental preservation. However, efficient scheduling in such systems necessitates not only the consideration of energy consumption but also the ability to adapt to the dynamic nature of the system. To tackle these challenges, we propose an online energy-aware scheduling algorithm for deadline-constrained applications in distributed heterogeneous systems, leveraging dynamic voltage and frequency scaling (DVFS) techniques. First, the algorithm models the continuously arriving applications and heterogeneous processors and proposes a novel task-sorting method to prioritize tasks, ensuring that more applications are completed within their respective deadlines. Second, the algorithm controls the selection range of processors based on the task's subdeadline and assigns the task to the processor with the minimum energy consumption. Through experiments conducted with randomly generated applications, our approach consistently exhibits superior performance when compared to similar scheduling algorithms.

Keywords: deadline constrained; dynamic systems; energy consumption; online scheduling

1. Introduction

Distributed heterogeneous systems stand out for their network of myriad computing nodes, each furnished with a unique combination of computing resources across various types and performance tiers. In the current computing landscape, the significance of such systems has gained remarkable prominence [1, 2]. These systems offer a multitude of benefits, including high-performance, efficient, and highly reliable computing capabilities, making them suitable for a wide range of application scenarios. These domains range from scientific computation, big data analytics, and artificial intelligence to cloud services, the Internet of Things, and beyond. By harnessing the diverse array of computing resources available within distributed heterogeneous systems, it becomes feasible to adeptly meet diverse application requirements [3]. Furthermore, leveraging these resources allows for enhanced system performance, improved energy utilization efficiency, and the facilitation of innovation and development across diverse fields [4]. In the orchestration of these resources, scheduling algorithms assume a pivotal role, ensuring their efficient allocation and utilization.

The issue of energy consumption poses a significant challenge in the context of distributed heterogeneous systems. Due to the varying energy consumption characteristics of different types of computing nodes within the system, it becomes imperative to devise effective approaches for managing and optimizing energy consumption. Addressing the energy consumption problem entails the development of resource allocation and task scheduling strategies that aim to minimize overall energy consumption while meeting the performance requirements of tasks. Through the utilization of energy-aware scheduling algorithms in tandem with dynamic voltage and frequency scaling (DVFS) techniques,

distributed heterogeneous systems can effectively manage and control energy consumption [5]. This approach allows the system to enhance energy efficiency and sustainability by dynamically adjusting the voltage and frequency levels of computing nodes in response to workload demands.

As is widely recognized, the energy-conscious task scheduling problem is classified as an NP-hard problem [6]. Many existing algorithms primarily address static or offline scenarios, wherein all tasks are predetermined and the environment is assumed to be stable [7–10]. Despite the wealth of insightful research on static scheduling algorithms, their adaptability falls short in the face of the dynamic landscape of distributed heterogeneous systems, which are characterized by fluctuating performance levels and node availability. Consequently, the focus on online scheduling becomes critical. Through online scheduling, dynamic decisions are made concerning task distribution and scheduling by taking into account the system's current state as well as the real-time influx of tasks, necessitating persistent surveillance of the system and resource conditions in either real time or near real time. By doing so, the system can adapt to changing conditions, allocate tasks efficiently, and optimize energy consumption in real time, thereby enhancing overall system performance and resource utilization.

Therefore, when scheduling applications in distributed heterogeneous systems, it is necessary to consider both energy consumption and online scheduling issues simultaneously. But most studies are static [11, 12] or do not consider energy consumption [13, 14]. To tackle these challenges, we propose a dynamic scheduling algorithm for randomly arrived applications, namely, online minimum energy consumption with deadline-constrained scheduling (OMECDS) algorithm. The algorithm makes informed decisions about application allocation, resource selection, and frequency scaling, which can dynamically adapt to the system's changing conditions and optimize resource usage based on workload characteristics, energy profiles of processors, and timing constraints. The contributions are as follows:

1. By leveraging models of complex applications and heterogeneous processors, we propose a formulation for the energy-aware application scheduling problem in heterogeneous distributed systems. This formulation revolves around a constrained optimization approach, aiming to minimize the energy consumption of applications while adhering to strict deadline constraints.

2. We propose an OMECDS algorithm which efficiently allocates suitable processors for stochastic incoming applications with lower time complexity. Moreover, we propose a novel approach to prioritize tasks from different applications to enhance the success rate of application execution. The proposed algorithm integrates a processor selection mechanism that considers the subdeadline defined for each task, with the primary objective of minimizing energy consumption while adhering to the imposed deadline constraints.

3. To evaluate the performance of our proposed approach, we conducted experiments using simulations with randomly generated applications. The results obtained indicate that OMECDS outperforms other existing approaches.

The remainder of this article is organized as follows. In Section 2, we provide an overview of the related work on task scheduling algorithms. Section 3 introduces the application model, energy model, and heterogeneous processor model and provides a detailed description of the problem. In Section 4, we present the algorithm proposed in this study. Section 5 presents the experimental results and provides an in-depth analysis. Lastly, in Section 6, we discuss the conclusion and further work of our research.

2. Related Work

Considerable attention has been devoted to investigating scheduling algorithms that incorporate energy consumption considerations in distributed heterogeneous systems. This research has broadly categorized existing energy-conscious scheduling strategies into two main groups: those prioritizing completion times and those driven by multiple objectives, according to their scheduling aims.

Completion time–oriented methods prioritize minimizing the overall completion time of applications in a distributed heterogeneous system, while also taking energy consumption into account. By adjusting the voltages and frequencies of system processors through techniques like DVFS, these methods are aimed at optimizing the performance-energy trade-off. Xiao et al. [15] introduced an algorithm called MSLECC, which addresses the challenge of minimizing the dispatch length of parallel applications while ensuring limited energy consumption in heterogeneous distributed systems. MSLECC incorporates an energy consumption constraint to ensure that the energy consumed during the scheduling process remains within predefined limits. By considering the energy efficiency trade-offs associated with DVFS, the algorithm strikes a balance between achieving shorter schedule lengths and maintaining energy consumption at an acceptable level. Chen et al. [11] introduced an enhanced approach to address the challenges of the MSLECC algorithm. They proposed two key strategies: an efficient task prioritization strategy and a weight-based energy distribution strategy. Building upon the genetic algorithm framework, Liu et al. [16] proposed an energy-conscious dependence task scheduling algorithm with a multiobjective fitness function to balance the makespan and energy consumption. These aforementioned studies are geared toward enhancing scheduling efficiency in distributed heterogeneous systems by transforming the multifaceted challenge of minimizing completion time and energy consumption into a singular scheduling objective. This conversion allows for the application of various optimization techniques and algorithms to find an efficient solution.

Multiple objective-oriented scheduling algorithms are designed to address scheduling problems that involve multiple conflicting objectives [17, 18]. In scheduling scenarios, multiple objectives must be simultaneously addressed, including minimizing energy consumption, lowering costs,

maximizing resource utilization, and meeting deadline constraints. The multiobjective optimization scheduling algorithm endeavors to identify an optimal set of solutions, specifically the nondominated solution set, often referred to as the Pareto optimal solution set. Zhang et al. [19] focus on the combinatorial optimization problem of biobjective optimization, aiming to achieve a high level of system reliability while minimizing energy consumption in the context of parallel tasks. Li et al. [20] introduced a hybrid energy-aware multiobjective optimization algorithm that encompasses eight distinct types of neighborhood structures. In order to balance global and local search abilities, a hybrid deep exploitation and deep exploration method is employed. Khiat, Haddadi, and Bahnes [21] employed an innovative amalgamation of hybrid genetic, max–min, min–min, and random algorithms to orchestrate a harmonious equilibrium between energy consumption and overall response time. Such an equilibrium was accomplished by the judicious modulation of processor voltages and frequencies, thereby facilitating the attainment of an optimal computational solution. Rehman et al. [22] proposed an algorithm called MOGA, which was employed to address the conflicting interests of cloud stakeholders in optimization problems. This approach is aimed at minimizing the makespan while adhering to budget and deadline constraints, while also offering an energy-efficient solution through the utilization of DVFS. Additionally, a gap search algorithm was proposed in this study to optimize the resource utilization of the cloud's available resources.

While the papers mentioned earlier primarily focused on static scheduling methods with offline task allocation processes, there is ongoing research in the field of online scheduling algorithms that can effectively handle the arrival of random tasks [23]. Zhou et al. [24] presented a multiworkflow scheduling algorithm designed to dynamically schedule concurrent workflows while adhering to user-defined deadline and budget constraints. In their approach, they use the concept of rank_u from HEFT [25] as a task priority metric. Additionally, they introduced a bifactor selection method for resources, aiming to strike an optimal balance between the budget and deadline constraints. Arabnejad, Bubendorfer, and Ng [26] involve the utilization of a central queue to manage the workflow that has arrived. In this process, the subdeadline of each task is calculated, and the algorithm employs the early deadline first strategy to prioritize and select tasks for scheduling. This strategy is aimed at improving the success number of the workflow by ensuring that tasks with earlier deadlines are scheduled first.

In the domain of task scheduling for distributed heterogeneous systems, it is crucial to take into account both energy consumption and online scheduling issues. By employing appropriate online energy consumption-aware scheduling algorithms, it enhances the overall system performance and improves energy utilization efficiency.

3. Model

In this section, we establish the heterogeneous processor model, application model, and energy model utilized in our study.

3.1. Heterogeneous Processor Model. The distributed heterogeneous system we considered consists of m diverse processors, denoted as $P = \{p_k | k \in [1, 2, \cdots, m]\}$, interconnected by a high-speed network. Each individual processor, $p_k \in P$, is equipped with DVFS capabilities, allowing it to switch between different frequencies, denoted as f_k. The available frequency range for processor p_k spans from the minimum value, $f_{k_{\min}}$, to the maximum value, $f_{k_{\max}}$. It is important to note that the execution time of a task decreases as the processor's frequency increases, while the energy consumption increases. We disregard the energy impact resulting from processor frequency-state transitions. In our assumption, when a task is assigned to a processor, it is executed immediately. However, it is important to note that a processor can only execute one task at a time. This restriction ensures that the processor's resources are effectively utilized and that tasks are processed sequentially to avoid any conflicts or resource contention [27, 28].

3.2. Application Model. A parallel application, composed of several interlinked tasks, can be depicted as a directed acyclic graph (DAG). In this representation, each node corresponds to an individual task, and each edge delineates the data dependencies among tasks. The application is submitted in a stochastic manner, but once submitted, all relevant information about the application is known. We denote the set of dynamic applications as $\eta = \{G_1, G_2, \cdots, G_w\}$. A specific DAG within this set, denoted as G_s, is modeled as $G_s = \{a^s, d^s, V^s, E^s\}$. In this model, a^s represents the arrival time of G_s, d^s represents the deadline of G_s, V^s is the set of nodes, where each node $v_i^s \in V^s$ represents a task, and E^s is the set of edges, where each directed edge $e_{i,j}^s \in E^s$ indicates the data dependency between tasks v_i^s and v_j^s. The communication time between tasks v_i^s and v_j^s is denoted as $\text{dt}_{i,j}^s$. If v_i^s and v_j^s are located on the same processor, we assume the communication time $\text{dt}_{i,j}^s = 0$. A task can only be executed when all the data from its predecessors have been transferred. A task without any immediate predecessors is referred to as an entry task, symbolized as v_{entry}^s, while a task with no successors is referred to as an exit task, symbolized as v_{exit}^s. Figure 1 shows an example of a DAG-based parallel application that describes the data dependencies of eight tasks.

Due to the heterogeneity of processors, the execution time of a task can vary across different processors. The average execution time of task v_i^s across all processors with the maximum frequency is defined as follows:

$$\overline{\text{et}(v_i^s)} = \sum_{p_k \in P} \frac{\text{et}(v_i^s)_k}{m} \tag{1}$$

where $\text{et}(v_i^s)_k$ is the execution time of task v_i^s on processor p_k with the maximum frequency. The performance of a processor is influenced by its operating frequency, which in turn can lead to varying execution times for the same task. The actual execution time of task v_i^s when allocated

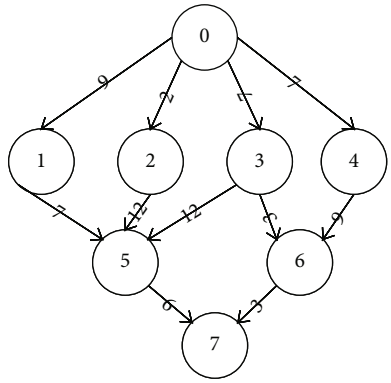

Figure 1: An example of a DAG-based parallel application with eight tasks.

on processor p_k with frequency $f_{k,h}$ is defined as follows:

$$\mathrm{et}(v_i^s)_{k,h} = \frac{f_{k,\max}}{f_{k,h}} \times \mathrm{et}(v_i^s)_k \tag{2}$$

where $f_{k,h}$ is the actual frequency of p_k and $f_{k,\max}$ is the maximum frequency of p_k.

The completion of all tasks in the DAG G_s signifies the completion of G_s as a whole.

Therefore, we define makespan^s as the time it takes for G_s to be fully completed. The calculation of makespan^s can be determined by

$$\mathrm{makespan}^s = \max_{v_i^s \in V^s} \{\mathrm{aft}(v_i^s)\} - a^s \tag{3}$$

where $\mathrm{aft}(v_i^s)$ is the actual finish time of v_i^s.

3.3. Energy Model. For the processor's power model, we used the references in the existing articles [12, 29, 30]. The power consumption manifests in two forms: dynamic and static power consumption. The power consumption $\mathrm{Power}(p_k)$ of processor p_k with frequency $f_{k,h}$ can be computed via

$$\mathrm{Power}(p_k) = P_k^s + h\left(P_k^{\mathrm{ind}} + P_k^d\right) = P_k^s + h\left(P_k^{\mathrm{ind}} + C_k^{\mathrm{ef}} f_{k,h}^{m_k}\right) \tag{4}$$

where P_k^s represents the static/leakage power consumed when the processor is turned on, P_k^{ind} denotes the frequency-independent dynamic power, C_k^{ef} represents the effective switching capacitance, m_k is the dynamic power exponent, and h is a Boolean coefficient indicating whether the processor is active or inactive. If a task is being executed on this processor, $h = 1$; otherwise, $h = 0$. It is assumed in this paper that the static power, which can only be eliminated by turning off the processor, is ignored. Additionally, it is assumed that the processor can only change its frequency when no task is being executed on it.

While DVFS can indeed contribute to curbing energy consumption, it is imperative to acknowledge that operating at diminished frequencies may inevitably lead to prolonged completion times for applications. Consequently, considering system-level power alone, lower frequencies may not always be the optimal choice for achieving energy savings. Thus, there exists a minimum energy-efficient frequency for processor p_k, denoted as $f_{k,ee}$ [31–33], which is determined by

$$f_{k,ee} = \sqrt[mk]{\frac{P_k^{\mathrm{ind}}}{(m_k - 1)C_k^{\mathrm{ef}}}}. \tag{5}$$

Assuming that the processor's frequency can be adjusted continuously within the range of $f_{k,\min}$ and $f_{k,\max}$, it is crucial for energy efficiency to limit the actual frequency f within the range of $f_{k,\mathrm{low}}$ and $f_{k,\max}$ (i.e., $f \in [f_{k,\mathrm{low}}, f_{k,\max}]$). $f_{k,\mathrm{low}}$ is determined as the maximum value between $f_{k,\min}$ and $f_{k,ee}$, which is calculated by

$$f_{k,\mathrm{low}} = \max\left\{f_{k,ee}, f_{k,\min}\right\}. \tag{6}$$

The energy consumption $E(v_i^s, p_k, f_{k,h})$ of task v_i^s on the processor p_k with frequency $f_{k,h}$ is calculated as

$$E\left(v_i^s, p_k, f_{k,h}\right) = \left(P_k^{\mathrm{ind}} + C_k^{\mathrm{ef}} f_{k,h}^{m_k}\right) \times \frac{f_{k,\max}}{f_{k,h}} \times \mathrm{et}(v_i^s)_k. \tag{7}$$

Therefore, the total energy consumption of application G_s can be obtained by

$$E_{\mathrm{total}}(G_s) = \sum_{i=1}^{n^s} E(v_i^s). \tag{8}$$

The used notations in this paper are presented in Table 1.

3.4. Problem Description. The primary aim of this study is to tackle the issue of allocating available processors to all tasks within incoming applications, considering the appropriate frequency settings for each task. The objective is to minimize the energy consumption of the application while simultaneously guaranteeing that the execution time of the application remains within the specified deadline constraints. The algorithm tries to find an optimal allocation strategy that achieves a balance between minimizing energy usage and meeting the application's deadline requirements. Hence, the optimization objective is defined as follows:

$$\begin{aligned} &\min \sum_{G_s \in \eta} E_{\mathrm{total}}(G_s) \\ &\text{s.t.} \\ &(1)\ \mathrm{aft}\left(v_i^s\right) + \mathrm{dt}_{i,j}^s \le \max_{v_j^s \in \mathrm{succ}\left(v_i^s\right)} \left\{\mathrm{ast}\left(v_j^s\right)\right\} \\ &(2)\ \sum_{k=1}^{m} x_{i,k}^s \le 1, \forall i \in \{1, 2, \cdots, n^s\}, \forall s \in \{1, 2, \cdots, w\} \\ &(3)\ \mathrm{makespan}^s \le d^s, \forall s \in \{1, 2, \cdots, w\}. \end{aligned} \tag{9}$$

Table 1: The used notations in this paper.

Notation	Implication
p_k	The k-th processor in the system
m	The number of processors
G_s	The s-th DAG in the system
a^s	The arrival time of DAG G_s
d^s	The deadline of G_s
V^s	The set of tasks in G_s
E^s	The set of edges between all tasks in G_s
v_i^s	The i-th task in G_s
n^s	The number of tasks in G_s
$\text{pred}(v_i^s)$	The set of predecessors of task v_i^s
$\text{succ}(v_i^s)$	The set of successors of task v_i^s
$\text{dt}_{i,j}^s$	The data transfer time between v_i^s and v_j^s
$\overline{\text{et}}(v_i^s)$	The average execution time of v_i^s
$\text{et}(v_i^s)_{k,h}$	The actual execution time of task v_i^s on processor p_k with frequency $f_{k,h}$
$\text{est}(v_i^s)$	The earliest start time of v_i^s
$\text{eft}(v_i^s)$	The earliest finish time of v_i^s
$\text{avail}(p_k)$	The earliest time that the processor p_k can execute a task
makespan_s	Schedule completion time of G_s
$E(v_i^s, p_k, f_{k,h})$	The energy consumption of task v_i^s on the processor p_k with frequency $f_{k,h}$

Constraint (1) in Equation (9) states that a task cannot be executed until all of its predecessors are completed, and all data transmissions between them have been finished. The variable $\text{ast}(v_j^s)$ represents the actual start time of task v_j^s. The $\text{aft}(v_i^s)$ is the actual finish time of task v_i^s. The aft (v_i^s) is determined by adding the actual execution of v_i^s to the actual start time $\text{aft}(v_i^s)$. In Constraint (2), the binary variable $x_{i,k}^s$ indicates whether task v_i^s is assigned to processor p_k. This assignment is defined as follows:

$$x_{i,k}^s = \begin{cases} 1, & \text{if } v_i^s \text{ scheduled on } I_k \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Constraint (2) ensures that each task can only be scheduled once, preventing duplication or multiple assignments. Constraint (3) states that the actual completion time of the application must satisfy its deadline requirement.

4. Proposed OMECDS Algorithm

In this section, we introduce the OMECDS algorithm, tailored for managing the scheduling of multiple applications with diverse arrival times within a heterogeneous distributed system. The main objective of the algorithm is to minimize the overall energy consumption of the system while efficiently scheduling as many applications as possible. This is accomplished through meticulous optimization of resource allocation and task scheduling, all the while adhering to the specified deadline constraints of the applications. The algorithm is divided into two main phases: the task selection phase and the processor allocation phase.

During the task selection phase, we introduce a novel approach to calculate the priority and subdeadline of each ready task derived from the incoming applications. The computation of priority and subdeadline takes into account an array of factors including task dependencies, arrival time, overarching scheduling objectives, and other pertinent considerations.

In the processor allocation phase, we map each eligible task to an available processor, taking into account the task's subdeadline. The goal is to assign the task to a processor and select an appropriate frequency setting that ensures the subdeadline is not violated while minimizing energy consumption. This phase involves making intelligent decisions to optimize the allocation of processors and frequencies, considering the dynamic characteristics of the tasks and the available resources. The objective is to achieve efficient task execution while conserving energy resources.

4.1. Task Selection Phase. In contrast to static scheduling, online scheduling operates under the condition that the topology and data information of the application are unknown until the application is submitted. This necessitates that the algorithm must be capable of handling incoming applications, parsing their task information, and making decisions promptly. The algorithm must dynamically adapt to the evolving task requirements and efficiently make real-time scheduling decisions. It should be able to process incoming applications on the fly, adjust the task allocation, and optimize the resource utilization based on the available information. This ability to handle unknown and evolving application characteristics is essential for effective online scheduling. As the DAGs arrive, they are stored in the DAG_{Pool}, and their parameters are parsed during the task selection phase. In this phase, the algorithm is aimed at identifying the ready tasks, determining their priority, and allocating subdeadline for scheduling.

When a DAG is added to the DAG_{Pool}, the earliest start time and the last finish time of its tasks can be calculated based on the provided feature parameters. The earliest start time represents the earliest possible time for a task to be executed on a processor, which means that all tasks are finished at a minimum time. We denote the earliest start time of task v_i^s as $\text{est}(v_i^s)$, which is defined as follows:

$$\text{est}(v_i^s) = \begin{cases} a^s, & \text{if } v_i^s = v_{\text{entry}}^s \\ \max\limits_{v_j^s \in \text{pred}(v_i^s)} \left\{ \text{est}\left(v_j^s\right) + \text{dt}_{i,j}^s \right\}, & \text{otherwise} \end{cases} \tag{11}$$

where the $\text{pred}(v_i^s)$ is the set of the immediate predecessors of task v_i^s. The entry task's earliest start time is its

Require: The arrival DAG $G_s = \{a^s, d^s, V^s, E^s\}$.
Calculate $est(v_i^s)$, $eft(v_i^s)$ and $lft(v_i^s)$ for each tasks in V^s;
Add all entry tasks into RTL;
Add other tasks in V^s into $TaskPool$;
Calculate the $urgency(v_i^s)$ of all tasks in RTL; and sort them in non-descending order;
Call ScheduleReadyTask(RTL;);

Algorithm 1: The preprocessing for an arrival DAG.

application's arrival time. Accordingly, the earliest finish time of v_i^s is defined as

$$\mathrm{eft}(v_i^s) = \mathrm{est}(v_i^s) + \overline{\mathrm{et}(v_i^s)}. \tag{12}$$

The latest finish time of a task represents the maximum allowable time for its completion, ensuring that the task does not exceed the specified deadline constraint. We define $lft(v_i^s)$ as latest finish time of v_i^s, which can be calculated by

$$\mathrm{lft}(v_i^s) = \begin{cases} d^s, & \text{if } v_i^s = v_{\mathrm{exit}}^s \\ \min\limits_{v_p^s \in \mathrm{succ}(v_i^s)} \left\{ \mathrm{lft}\left(v_p^s\right) - \overline{\mathrm{et}\left(v_p^s\right)} - \mathrm{dt}_{i,p}^s \right\}, & \text{otherwise} \end{cases} \tag{13}$$

where $\mathrm{succ}(v_i^s)$ is the set of immediate successors of v_i^s.

To accommodate the system's dynamics and adhere to task priority constraints, the OMECDS employs a strategy of exclusively scheduling ready tasks during each iteration. This approach prevents the scenario where the initial applications occupy a significant portion of system resources, potentially impeding the completion of subsequent applications. By prioritizing the scheduling of ready tasks, the algorithm ensures that tasks from different applications have a fair opportunity to be executed and progress toward completion according to their priorities. This approach promotes efficient utilization of system resources and facilitates the timely execution of all arriving applications.

Ready task means a task has no predecessor or its all immediate predecessors have been completed. When new DAGs arrive, all the entry tasks within the DAG are considered ready tasks and are added to the ready task list (RTL). Additionally, when the last unfinished predecessor of a task is completed, that task becomes a ready task and is also added to the RTL. As a result, the RTL consists of all tasks that are currently ready for execution. All tasks within the RTL can be executed in parallel, as they have met their dependencies and are independent of each other in terms of execution order.

Sorting the ready tasks is a crucial aspect of this work. Unlike previous studies [15, 34], which focused on a single DAG and utilized the rank_u as the tasks' priority, this work deals with tasks from multiple DAGs that have distinct deadlines. To maximize the number of completed DAGs within their respective deadlines, we introduce the concept of $\mathrm{urgency}(v_i^s)$ as the priority of tasks, which can be calculated by

$$\mathrm{urgency}(v_i^s) = \frac{\mathrm{et}(v_i^s)}{\mathrm{lft}(v_i^s) - \mathrm{est}(v_i^s)}. \tag{14}$$

The closer the task's $\mathrm{est}(v_i^s)$ is to the $\mathrm{lft}(v_i^s)$, the more priority it has. We rank urgency of all ready tasks. This prioritization allows for the scheduling algorithm to focus on tasks that are time-critical to ensure the successful completion of the corresponding DAGs within their deadlines. When comparing priorities between two tasks v_i^s and v_j^s, if both urgency of v_i^s and v_j^s are positive or negative, the task exhibiting greater urgency is deemed to have higher priority. Conversely, if the urgency value is one positive and the other negative, the task with negative urgency is ascribed higher priority.

The process of handling newly arrived applications in OMECDS is outlined in Algorithm 1. The algorithm utilizes the RTL to manage tasks that are ready for execution. Upon the arrival of new DAGs, the algorithm calculates the earliest start time $\mathrm{est}(v_i^s)$, earliest finish time $\mathrm{eft}(v_i^s)$, and latest finish time $\mathrm{lft}(v_i^s)$ for all tasks in the DAG (Line 1). This information is crucial for scheduling and processor allocation. For the new DAGs, only entry tasks are considered ready tasks. The algorithm adds these entry tasks to the RTL (Line 2). Additionally, all unselected tasks are added to the TaskPool (Line 3). The TaskPool serves as a repository for tasks that are not yet ready for execution. The algorithm sorts the ready tasks in the RTL based on their $\mathrm{urgency}(v_i^s)$ values (Line 4). This prioritization ensures that tasks with higher urgency, often associated with closer deadlines or critical dependencies, are scheduled first. Finally, ready tasks in the RTL are allocated to available processor using the function ScheduleReadtTask(RTL) (Line 5). This step involves assigning suitable processors and frequencies to the tasks, while considering their timing constraints and energy efficiency objectives.

In a DAG with n tasks, the maximum number of edges e is given by $((n-1)n)/2$. In Algorithm 1, Line 1 calculates $\mathrm{est}(v_i^s)$, $\mathrm{eft}(v_i^s)$, and $\mathrm{lft}(v_i^s)$ for each task in order to determine their scheduling parameters. This calculation requires a time complexity of $O(n^2)$. In Line 4, the length of the RTL queue is determined by the maximum indegree of a task in G_s, denoted as n_{in}. Sorting tasks in the RTL queue takes a time complexity of $O(n_{\mathrm{in}} \times \log(n_{\mathrm{in}}))$. The calculation of urgency in the algorithm also takes a time complexity of $O(n_{\mathrm{in}})$. Therefore, the overall time complexity of task selection for an arrival DAG G_s in the algorithm is $O(n^2 + n_{\mathrm{in}} \times$

Require: A completed task v_i^s
For each $v_j^s \in succ(v_i^s)$ **do**
  If all of pred (v_j^s) have been completed **then**
    Calculate $urgency(v_j^s)$;
    Add task v_j^s into RTL and remove v_j^s from $TaskPool$;
  Endif
Endfor
Sort all tasks in RTL by $urgency$ in a non-descending order;
Call ScheduleReadyTask(RTL);

Algorithm 2: The processing for the finish of a task.

$\log(n_{\text{in}})$). Considering that n_{in} is typically small compared to n, we can simplify the time complexity to $O(n^2)$.

The process of the algorithm that handles the completed task is detailed by Algorithm 2. When a task v_i^s is completed, the algorithm identifies the immediate successor tasks that become ready as a result. These ready tasks are added to the RTL to indicate that they are now eligible for execution (Lines 1–5). The urgency(v_j^s) values of selected tasks are calculated (Line 3). The selected tasks are then removed from the TaskPool to indicate that they have been scheduled for execution (Line 4). Tasks in the RTL are sorted based on their urgency values (Line 7). This ensures that the tasks with higher urgency, which reflects their priority, are scheduled first. Finally, the function ScheduleReadyTask(RTL) is called to allocate the resources and execute the ready tasks (Line 8).

The completion of a task indicates that its immediate successor tasks are likely to be ready for execution. By promptly identifying and scheduling ready tasks, the algorithm can potentially complete more applications within their respective deadline constraints. This dynamic handling of ready tasks enhances the overall efficiency and effectiveness of the scheduling algorithm. When a task v_i^s is completed, Algorithm 2 needs to traverse each task in succ(v_i^s), which needs time $O(n_{\text{out}})$. $O(n_{\text{out}})$ is the maximum outdegree of a task among G_s. The time complexity in Line 2 is $O(n_{\text{in}})$. The length of RTL saving the ready task in succ(v_i^s) is at most equal to n_{out}. Sorting tasks in RTL requires time $O(n_{\text{out}} \log(n_{\text{out}}))$. Hence, the time complexity of the processing for the finish of a task is $O(n_{\text{in}} \times n_{\text{out}} + n_{\text{out}} \times \log(n_{\text{out}}))$.

4.2. Processor Allocation Phase. In the processor allocation phase, the algorithm assigns appropriate processors and frequencies to the tasks in the RTL based on their priority order. The objectives are to ensure the timely completion of tasks, thereby averting any severe consequences, and to reduce the overall energy consumption of the system.

To meet the deadline constraints, we segment the overarching deadline of the application into subdeadlines, each meticulously assigned to individual tasks. The problem of matching the overall deadline of an application is transformed into the problem of matching the subdeadline of each individual task within the application. This transformation allows for a more fine-grained and task-centric approach to deadline management. The subdeadline of v_i^s can be calculated as

$$\text{SD}(v_i^s) = \text{eft}(v_i^s) + (\text{lft}(v_{\text{n}^s}^s) - \text{eft}(v_{n^s}^s)) \frac{\text{est}(v_i^s) - \text{lft}(v_i^s)}{\text{lft}(v_{n^s}^s) - \text{est}(v_1^s)} \quad (15)$$

where $\text{lft}(v_{n^s}^s)$ is the last finish time of last task in G_s and $\text{est}(v_1^s)$ is the earliest time of first task. It should be emphasized that the slack time $\text{SD}(v_i^s)$, associated with task v_i^s, must not surpass its latest finish time $\text{lft}(v_i^s)$. Should the calculation of $\text{SD}(v_i^s)$ based on Equation (15) result in a value greater than $\text{lft}(v_i^s)$, the $\text{SD}(v_i^s)$ is accordingly adjusted to equal $\text{lft}(v_i^s)$. Transforming the problem in this way enables the algorithm to schedule tasks based on their individual subdeadlines. It provides a more detailed understanding of the temporal requirements of each task and facilitates more accurate processor allocation and scheduling decisions.

The algorithm iterates through all available processors and frequencies to find the best combination for each ready task v_i^s. It considers the energy consumption $E(v_i^s, p_k, f_{k,h})$ for each processor p_k and frequency $f_{k,h}$ and checks if the estimated finish time $\text{eft}(v_i^s, p_k, f_{k,h})$ on that combination is less than the task's subdeadline $\text{SD}(v_i^s)$. $\text{eft}(v_i^s, p_k, f_{k,h})$ can be calculated by

$$\text{est}(v_i^s, p_k, f_{k,h}) = \begin{cases} \max\{\text{avail}(p_k), a^s\}, & \text{if } v_i^s = v_{\text{entry}}^s \\ \max\left\{\text{avail}(p_k), \max_{v_j^s \in \text{pred}(v_i^s)}\left\{\text{aft}\left(v_j^s\right) + \text{dt}_{i,j}^s\right\}\right\}, & \text{otherwise} \end{cases} \quad (16)$$

$$\text{eft}\left(v_i^s, p_k, f_{k,h}\right) = \text{est}\left(v_i^s, p_k, f_{k,h}\right) + \frac{\text{et}(v_i^s)_k * f_{k,\max}}{f_{k,h}} \quad (17)$$

where $\text{avail}(p_k)$ is the earliest time that p_k is idle.

By iterating through all possible combinations and selecting the one with the minimum energy consumption that satisfies the subdeadline constraint, the algorithm ensures that the task is allocated to the best available resources in terms of energy efficiency while meeting its timing requirements. If none of the processors can complete the task within its SD, then assign the task to the processor that completes it the fastest.

The pseudocode of the processor allocation phase is stated in Algorithm 3. The algorithm iterates through all

Require: each task v_i^s in RTL.
Ensure: The schedule scheme.
For each v_i^s in RTL **do**
  E$mineft, minE \longleftarrow MAXVALUE, bestPro, bestF \longleftarrow \varnothing$;
  For all processor p_k **do**
    For all frequency $f_{k,h}$ **do**
      Calculate the $est(v_i^s, p_k, f_{k,h})$, $eft(v_i^s, p_k, f_{k,h})$, and $E(v_i^s, p_k, f_{k,h})$;
      If $SD(v_i^s) > eft(v_i^s, p_k, f_{k,h})$ **then**
        If $minE > E(v_i^s, p_k, f_{k,h})$ **then**
          $mineft \longleftarrow eft(v_i^s, p_k, f_{k,h})$, $bestPro \longleftarrow p_k$, $bestF \longleftarrow f_{k,h}$, $minE \longleftarrow E(v_i^s, p_k, f_{k,h})$;
        Endif
      Endif
    Endfor
  Endfor
  Schedule v_i^s on $bestPro$ with $bestF$;
  Remove v_i^s from RTL;

Algorithm 3: Function ScheduleReadyTask(RTL).

available processors and frequencies to find the best combination for each ready task v_i^s (Lines 1–15). The algorithm initializes related variables for every task in RTL (Line 2). Then, traverse all processors and frequencies and calculate est, eft, and E (Lines 3–5). The algorithm checks if the earliest finish time ($\text{eft}(v_i^s, p_k, f_{k,h})$) of the task on the selected processor and frequency is less than the task's subdeadline ($\text{SD}(v_i^s)$) (Line 6). This ensures that the allocated processor can complete the task within the specified subdeadline. The algorithm selects the processor bestPro and frequency bestF combination that results in the minimum energy consumption for the task (Lines 7 and 8). Finally, assign task v_i^s to bestPro with bestF for execution and remove v_i^s from RTL (Lines 13–15).

In the processor allocation phase, we only select ready tasks for scheduling, which helps improve the system's responsiveness to dynamic events. By considering both energy consumption and subdeadline constraints, OMECDS chooses the best processor and frequency combination that minimizes energy consumption while considering the deadline constraint. The length of queue RTL is at most equal to n_{in}. For each task v_i^s in RTL, Algorithm 3 needs to traverse all processors. The number of computing processors is m, and the maximum number of frequencies is H. Lines 1–5 in Algorithm 3 are to calculate $\text{est}(v_i^s, p_k, f_{k,h})$, $\text{eft}(v_i^s, p_k, f_{k,h})$, and $E(v_i^s, p_k, f_{k,h})$ for each task and each processor, which need time $O(n_{\text{in}}^2 \times m + n_{\text{in}} \times m \times H)$. The time complexity of resource allocation is $O(n_{\text{in}} \times m \times (n_{\text{in}} + H))$.

5. Results

In this section, the performance of the proposed OMECDS algorithm is evaluated by comparing it with the LESA [11], MSLECC [15], and FCFS with rank_u algorithms. LESA addresses the challenge of energy-aware scheduling for dependent tasks on a heterogeneous multiprocessor system. It introduces a list-based algorithm to effectively determine start times and processor assignments. The approach optimizes task execution by balancing dependencies and energy constraints, strategically choosing suitable processors and speed settings for minimal completion time. MSLECC is a classic method of solving dependent task scheduling problems with the shortest schedule length while considering energy consumption constraints. It differs in task sequencing and energy allocation methods from LESA. FCFS with rank_u algorithm gives a rank for each DAG task according to the HEFT [25] and schedules DAGs according to the "first come first service." For the online scheduling problem considered in this article, the LESA and MSLECC algorithms are repeatedly executed at each new DAG arrival to handle the scheduling. All the algorithms used in the experiments are implemented in the Java programming language. The experiments were conducted on a computer with an Intel® Core™ i7-10700H CPU operating at 2.90 GHz, 16.00 GB RAM, and a 64-bit Windows 10 operating system.

5.1. Experimental Setup. The simulated heterogeneous distributed system consists of m processors with varying processing capabilities. The processor and application parameters used in the experiments are as follows: $10\,\text{ms} \le \text{et}(v_i^s, p_k) \le 100\,\text{ms}$, $10\,\text{ms} \le \text{dt}_{i,j}^s \le 100\,\text{ms}$, $0.03 \le P_k^{\text{ind}} \le 0.07$, $0.8 \le C_k^{\text{ef}} \le 1.4$, $2.5 \le m_k \le 3.0$, and $f_{k,\max} = 1\,\text{GHz}$ (all the frequency values are discrete with an accuracy of 0.01 GHz). These parameters are set exactly by reference to existing work [3, 12, 35]. In this experiment, 20 processors are randomly generated for each experiment.

The random DAGs used in the experiment are generated using a DAG generator based on several parameters, namely, CCR, n, fat, density, regularity, and jump. The parameter settings are chosen based on a previous paper [13] to ensure a diverse set of DAG characteristics. CCR represents the ratio of communication to computation in the DAG. The available values for this parameter are {0, 1, 2}. n determines the total number of tasks in the DAG. The values used in the experiment are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100},

providing a range of DAG sizes. fat influences the width and height of the DAG, thereby affecting its overall structure. The available values are {0.2, 0.4, 0.6}. density impacts the number of edges between different levels of nodes in the DAG. A higher density value indicates a larger number of edges. The values used in the experiment are {0.2, 0.4, 0.6}. regularity affects the uniformity of the number of tasks at each level of the DAG. A higher regularity value indicates a higher similarity in the number of tasks among different levels. The set of values used is {0.2, 0.4, 0.6}. jump represents the number of different levels an edge can connect. A larger jump value allows edges to connect nodes that are farther apart in the DAG. The available values for this parameter are {1, 2, 4}. Table 2 summarizes the parameter settings used in the DAG generator.

To simulate the dynamic arrival of DAGs, an arrival time needs to be set for each DAG. The arrival time represents the time when the DAG becomes available for scheduling, which is defined as

$$a^s = \begin{cases} 0, & \text{if } s = 0 \\ a^{s-1} + \alpha * d^{s-1}, & \text{otherwise} \end{cases} \quad (18)$$

where α is the parameter that can control the spacing and distribution of the DAG arrivals. The α is chosen with the values {5%, 10%, 15%, 20%, 25%, 30%}, where the default value is 15%. The arrival time interval is α multiply the deadline of the last arrived DAG. The larger the value of α, the longer the arrival interval of DAGs. The arrival time of the first DAG is 0.

The deadline of a DAG can be defined based on the calculation of a deadlineBase. deadlineBase represents the completion time of the longest path from the entry task to the exit task in the DAG, without considering the communication cost. By using the deadlineBase, we obtain the minimum overall deadline d^s for the DAG which is defined as

$$d^s = a^s + \beta * \text{deadlineBase}^s \quad (19)$$

where β is selected from [2, 2.5, 3, 3.5, 4, 4.5]. The default value of β is 2. We have procedurally generated the minimum execution times for tasks on an assortment of processors, ensuring a stochastic representation of the system's performance characteristics.

5.2. Results and Discussion

5.2.1. Planning Successful Rate (PSR) and Energy Consumption Analyses. In the experiment, the performance of the algorithms is evaluated based on two metrics: energy consumption and PSR [36]. The PSR metric represents the percentage of successfully scheduled DAGs, which is calculated as

$$\text{PSR} = 100 * \frac{\text{number of successful schedule}}{\text{total number in experiment}} \quad (20)$$

Table 2: The generator parameters of DAG.

Parameters	Value
n	= [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
fat	= [0.2, 0.4, 0.6]
density	= [0.2, 0.4, 0.6]
regularity	= [0.2, 0.4, 0.6]
jump	= [1, 2, 4]
CCR	= [0, 1, 2]

where a successful schedule means that the makespan of a DAG is less than its deadline. The energy consumption metric measures the total energy consumed by the system during the execution of the scheduled DAGs. To analyze the algorithms' performance in a dynamic environment, each algorithm is executed independently 20 times to obtain reliable and statistically significant results, and mean values are calculated. One thousand DAGs are generated using the random parameter values as a DAG pool. From this pool of DAGs, a set of 20 DAGs is randomly selected for each schedule. By running the algorithms multiple times and considering a diverse set of DAGs, the experiment is aimed at providing a comprehensive evaluation of the algorithms' performance, taking into account different scenarios and variations in DAG characteristics.

The first experiment, depicted in Figure 2, encompasses the evaluation of both PSR and average energy consumption across varying arrival time intervals. In Figure 2(a), the PSR demonstrates a discernible upward trend as the arrival time interval increases. This increase in the arrival factor, denoted by α, implies a reduction in the number of applications that the system must process within a unit of time. In the system under consideration, the number of processors is fixed. This means that the computing power of the system is limited. Consequently, the reduction of α leads to an enhancement in the successful completion rate of applications. However, it is important to note that the FCFS algorithm prioritizes minimizing the completion time of applications at the expense of significantly higher energy consumption, as illustrated in Figure 2(b). As the value of α increases, the urgency to complete applications within the specified deadline diminishes, leading to a greater emphasis on energy consumption performance, as depicted in Figure 2(b) when α exceeds 0.2. Remarkably, the OMECDS algorithm surpasses the MSLECC algorithm achieves notable energy savings and PSR improvements. In comparison to the LESA algorithm, the OMECDS algorithm exhibits a 30.55% increase in average energy consumption, but a remarkable 204.5% improvement in average PSR. When compared to the FCFS algorithm, the OMECDS algorithm achieves a 58.57% reduction in energy consumption and a 7.12% increase in average PSR.

Figures 3(a) and 3(b) present the PSR and average energy consumption of the four algorithms under varying deadline factors. With an increase in the value of β, the deadlines for each application are extended, resulting in

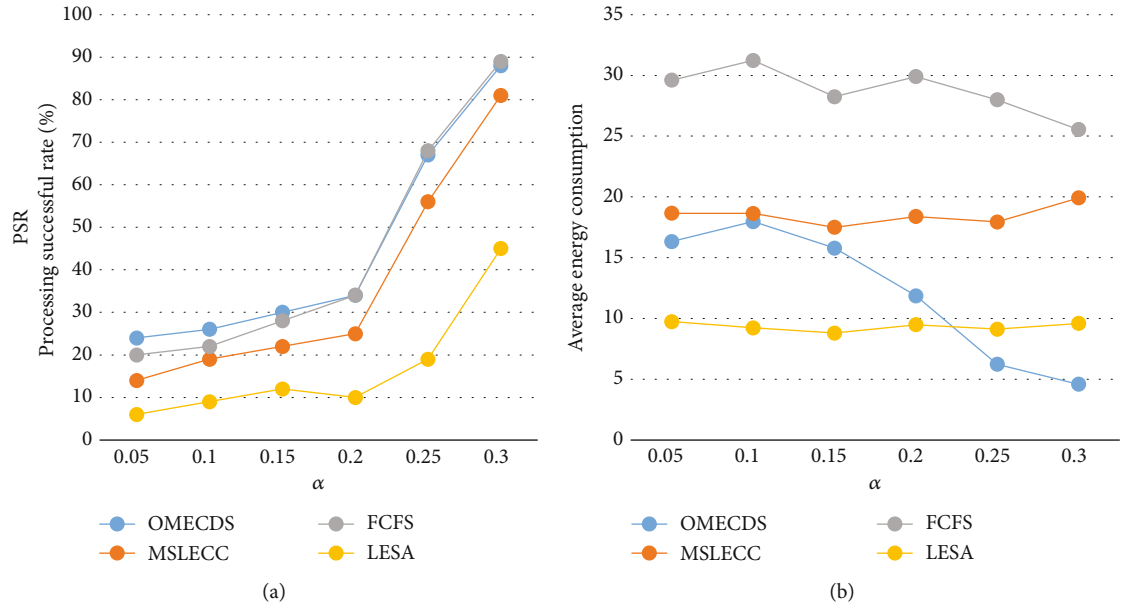

Figure 2: PSR values and average energy consumption of four algorithms with different interval arrival times.

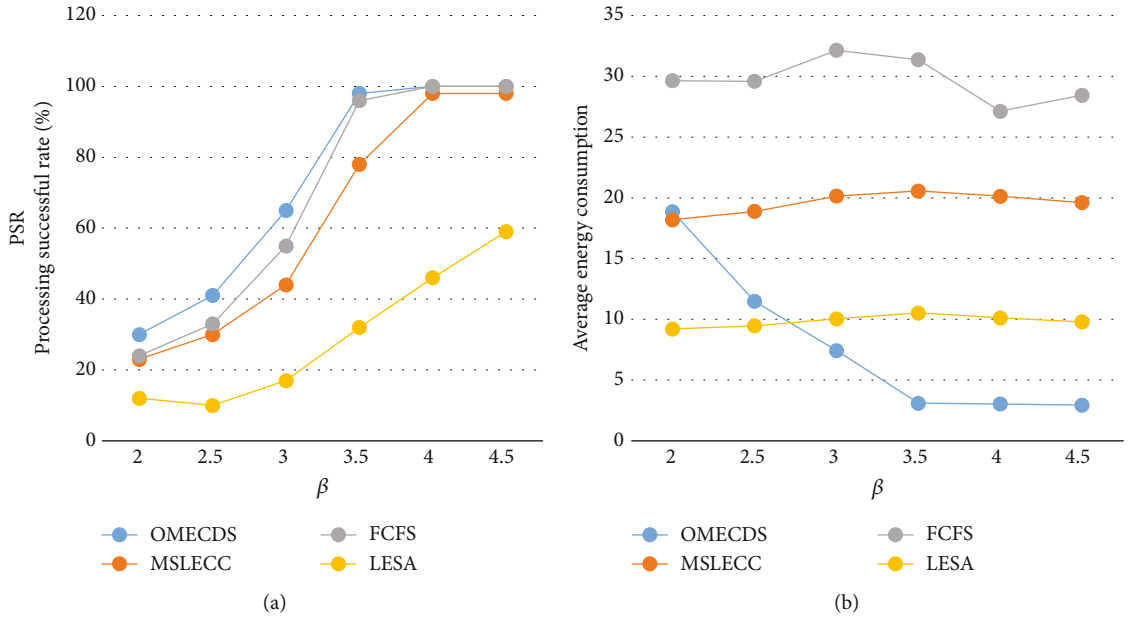

Figure 3: PSR values and average energy consumption of four algorithms with different deadline factors.

an improvement in the PSR for all algorithms. When β surpasses 3.5, OMECDS exhibits the capability to schedule all arrived applications while meeting their respective deadlines. OMECDS demonstrates a more remarkable ability to minimize energy consumption when the deadline is relaxed, as evidenced in Figure 3(b). In contrast, MSLECC and LESA algorithms allocate energy for each task, leading to similar energy consumption even under different deadline constraints, because their energy consumption constraints are fixed. Based on the experimental results, it is evident that OMECDS outperforms both FCFS and MSLECC algorithms in terms of both PSR and energy consumption across different values of β. Additionally, OMECDS surpasses LESA in terms of PSR across varying β values. Notably, when compared to LESA, in general, OMECDS still achieves a significant 18.41% reduction in average energy consumption.

In Figure 4, we experiment with different numbers of applications. Notably, the number of tasks in an application is still randomly generated. The number of applications is 10, 20, 30, 40, 60, 80, and 100. The α is 0.15 and β is 2.5. Obviously, OMECDS exhibits superior performance compared to FCFS and MSLECC algorithms. Compared with LESA, the average energy consumption of OMECDS experiences a slight increase of 7.97%, but the PSR shows a substantial improvement of 198%.

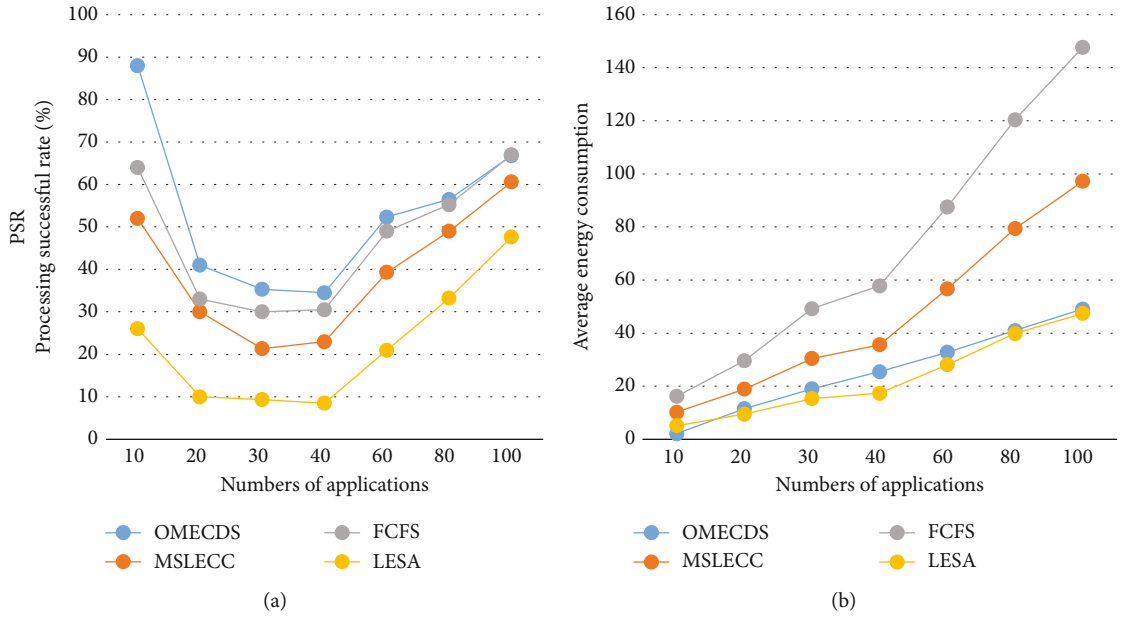

Figure 4: PSR values and average energy consumption of four algorithms with different numbers of applications.

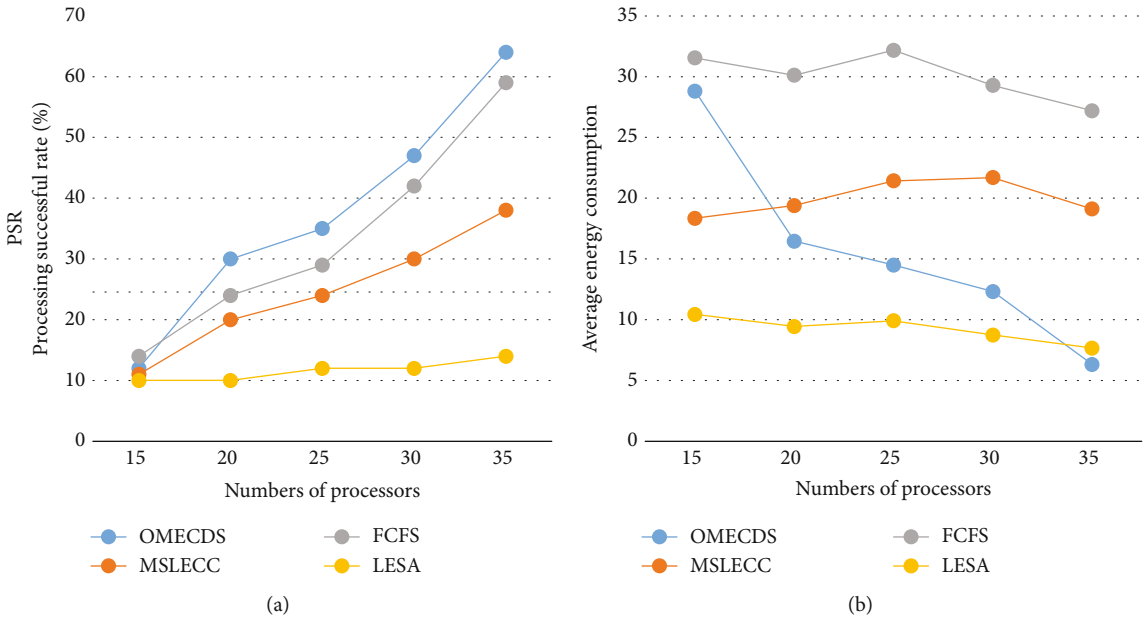

Figure 5: PSR values and average energy consumption of four algorithms with different numbers of processors.

Figures 5(a) and 5(b) show the performance of the four algorithms with a varied number of processor, while keeping the number of applications fixed at 20. All processors are randomly generated. The value of α is set to 0.15 and the value of β is set to 2. As the number of processors increases, the PSR of all algorithms tends to increase. When compared to MSLECC, OMECDS achieves an average PSR improvement of 46% and reduces average energy consumption by 20.11%. In comparison to FCFS, OMECDS shows a 10.35% improvement in PSR and a 48.72% reduction in energy consumption. Compared with LESA, OMECDS experiences a 63.94% increase in energy consumption but exhibits an impressive average PSR improvement of 212.09%. Overall, when considering both the successful completion rate of applications and energy consumption, OMECDS performs better than the other algorithms.

5.2.2. Runtime Analyses. This subsection is aimed at evaluating the time efficiency of the four algorithms. The running time represents the duration from the start of the algorithm to the point of finding a solution. Time performance is one of the important indicators to measure the algorithm.

In the first comparison, we analyze the average runtime of all algorithms under different deadline factors β, as illustrated

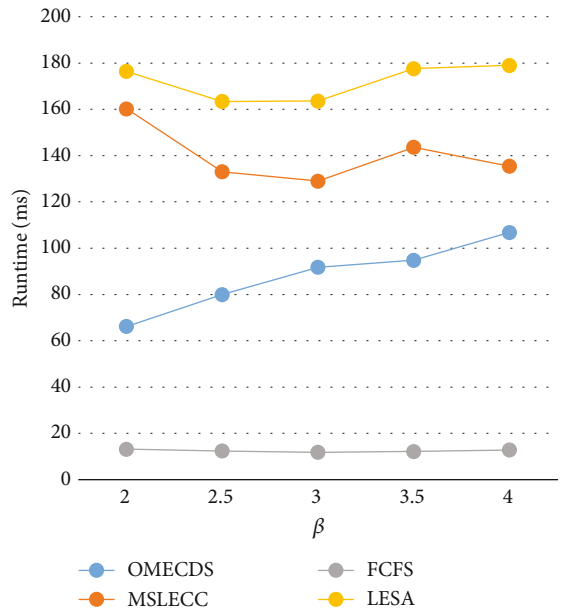

Figure 6: Running time of four algorithms with different deadline factors.

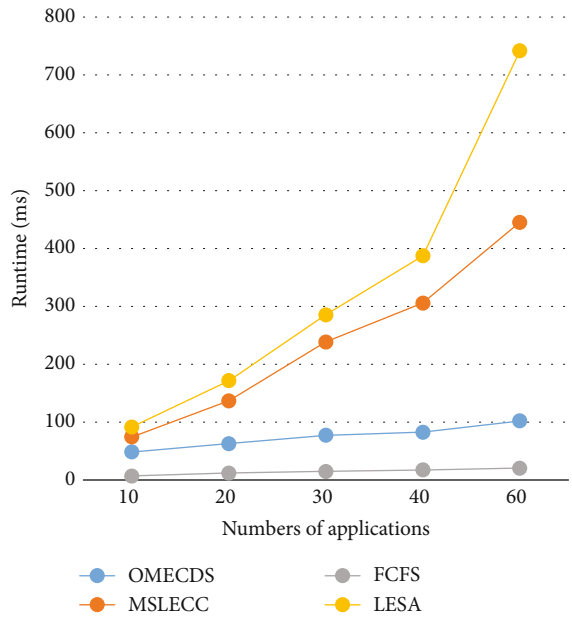

Figure 7: Running time of four algorithms with different numbers of applications.

in Figure 6. As mentioned previously, when the deadline constraint becomes looser, OMECDS algorithm tends to prioritize solutions with lower energy consumption. Consequently, as β increases, the running time of OMECDS experiences a slight increase. On the other hand, FCFS exhibits better time performance compared to OMECDS, LESA, and MSLECC due to its utilization of the maximum frequency for processors without dynamic adjustments. However, it is important to note that OMECDS still demonstrates better time performance compared to LESA and MSLECC. Both MSLECC and LESA are heuristic-based algorithms with low time complexity, indicating that OMECDS maintains good execution efficiency while achieving its objectives.

The last experiment is aimed at investigating the average running time of all algorithms under varying numbers of randomly generated applications. In Figure 7, it can be observed that as the number of applications increases, the average execution time of all algorithms also increases. However, our proposed algorithm demonstrates significantly shorter average execution times when compared to LESA and MSLECC. Although FCFS has a shorter execution time than OMECDS, it is important to highlight that OMECDS excels in application completion and energy savings. In summary, OMECDS exhibits outstanding performance in terms of execution efficiency, energy savings, and successful application scheduling.

6. Conclusions

To tackle the dynamic scheduling problem associated with parallel applications in heterogeneous distributed systems, we have introduced the OMECDS algorithm, which is aimed at minimizing energy consumption while adhering to deadline constraints. The proposed algorithm boasts a low computational complexity, making it efficient in practical implementation. Our devised task priority method effectively manages the spontaneous arrival of applications, while the formulation of subdeadlines and the selection mechanism for processors strike an optimal balance between meeting deadlines and minimizing energy consumption. Experimental results have demonstrated the superior performance of our approach compared to alternative algorithms. However, for future work, we plan to delve into distributed methods to tackle existing challenges. Centralized approaches are susceptible to single points of failure. We aspire to explore distributed solutions that can augment reliability and robustness in scheduling parallel applications.

Data Availability Statement

Data will be made available on request.

Conflicts of Interest

The authors declare no conflicts of interest.

Author Contributions

Y.L. and J.C. designed the resource. Y.L. wrote the first draft of the paper. Y.L., X.D., J.C., and C.D. analyzed the results, read the manuscript, and approved the final version. All authors have read and agreed to the published version of the manuscript.

Funding

This work was supported by the National Natural Science Foundation of China (No. 62106202), the Key Research and Development Program of Shaanxi (No. 2024GX-YBXM-118), the Aeronautical Science Foundation of China (No. 2023M073053003), and the Fundamental Research Funds for the Central Universities.

References

[1] J. Chen, C. Du, Y. Zhang, P. Han, and W. Wei, "A clustering-based coverage path planning method for autonomous heterogeneous UAVs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 25546–25556, 2022.

[2] J. Chen, Y. Zhang, L. Wu, T. You, and X. Ning, "An adaptive clustering-based algorithm for automatic path planning of heterogeneous UAVs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16842–16853, 2022.

[3] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li, "Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3426–3442, 2017.

[4] Q. Zhang, S. Geng, and X. Cai, "Survey on task scheduling optimization strategy under multi-cloud environment," *CMES-Computer Modeling in Engineering & Sciences*, vol. 135, no. 3, pp. 1863–1900, 2023.

[5] Y. Zhang, "DVFS-based energy-aware scheduling of imprecise mixed-criticality real-time tasks," *Journal of Systems Architecture*, vol. 137, article 102849, 2023.

[6] X. Tang, W. Cao, H. Tang et al., "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2079–2092, 2022.

[7] S. Chang, X. Zhao, Z. Liu, and Q. Deng, "Real-time scheduling and analysis of parallel tasks on heterogeneous multi-cores," *Journal of Systems Architecture*, vol. 105, article 101704, 2020.

[8] J. Zhou, J. Yan, K. Cao et al., "Thermal-aware correlated two-level scheduling of real-time tasks with reduced processor energy on heterogeneous MPSoCs," *Journal of Systems Architecture*, vol. 82, pp. 1–11, 2018.

[9] S. Potluri, S. Mohanty, and S. Mohanty, "QoS-driven hybrid task scheduling algorithm in a cloud computing environment," *International Journal of Grid and Utility Computing*, vol. 14, no. 4, pp. 311–319, 2023.

[10] J. Chen, P. Han, Y. Zhang, T. You, and P. Zheng, "Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems," *Journal of Systems Architecture*, vol. 142, article 102938, 2023.

[11] J. Chen, Y. He, Y. Zhang, P. Han, and C. Du, "Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems," *Journal of Systems Architecture*, vol. 129, article 102598, 2022.

[12] N. Gao, C. Xu, X. Peng, H. Luo, W. Wu, and G. Xie, "Energy-efficient scheduling optimization for parallel applications on heterogeneous distributed systems," *Journal of Circuits, Systems and Computers*, vol. 29, no. 13, article 2050203, 2020.

[13] G. Wang, Y. Wang, M. S. Obaidat, C. Lin, and H. Guo, "Dynamic multiworkflow deadline and budget constrained scheduling in heterogeneous distributed systems," *IEEE Systems Journal*, vol. 15, no. 4, pp. 4939–4949, 2021.

[14] J. Liu, J. Ren, W. Dai et al., "Online multi-workflow scheduling under uncertain task execution time in IaaS clouds," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1180–1194, 2021.

[15] X. Xiao, G. Xie, R. Li, and K. Li, "Minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 1471–1476, Tianjin, China, 2016.

[16] Y. Liu, C. Du, J. Chen, and X. Du, "Scheduling energy-conscious tasks in distributed heterogeneous computing systems," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 1, article e6520, 2022.

[17] J. Chen, T. Li, Y. Zhang et al., "Global-and-local attention-based reinforcement learning for cooperative behaviour control of multiple UAVs," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 3, pp. 4194–4206, 2024.

[18] J. Chen, F. Ling, Y. Zhang, T. You, Y. Liu, and X. Du, "Coverage path planning of heterogeneous unmanned aerial vehicles based on ant colony system," *Swarm and Evolutionary Computation*, vol. 69, article 101005, 2022.

[19] L. Zhang, K. Li, C. Li, and K. Li, "Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems," *Information Sciences*, vol. 379, pp. 241–256, 2017.

[20] J. Li, H. Sang, Y. Han, C. Wang, and K. Gao, "Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions," *Journal of Cleaner Production*, vol. 181, pp. 584–598, 2018.

[21] A. Khiat, M. Haddadi, and N. Bahnes, "Genetic-based algorithm for task scheduling in fog–cloud environment," *Journal of Network and Systems Management*, vol. 32, no. 1, p. 3, 2024.

[22] A. Rehman, S. S. Hussain, Z. ur Rehman, S. Zia, and S. Shamshirband, "Multi-objective approach of energy efficient workflow scheduling in cloud environments," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 8, article e4949, 2019.

[23] Y. Liu, C. Du, J. Chen, and X. Du, "Online deadline-constrained scheduling of parallel applications in CPSs," 2023.

[24] N. Zhou, F. Li, K. Xu, and D. Qi, "Concurrent workflow budget-and deadline-constrained scheduling in heterogeneous distributed environments," *Soft Computing*, vol. 22, no. 23, pp. 7705–7718, 2018.

[25] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[26] V. Arabnejad, K. Bubendorfer, and B. Ng, "Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds," *Future Generation Computer Systems*, vol. 100, pp. 98–108, 2019.

[27] L. Li, W. Xu, Y. Tan, Y. Yang, J. Yang, and D. Tan, "Fluid-induced vibration evolution mechanism of multiphase free sink vortex and the multi-source vibration sensing method," *Mechanical Systems and Signal Processing*, vol. 189, article 110058, 2023.

[28] L. Li, Q. Li, Y. Ni, C. Wang, Y. Tan, and D. Tan, "Critical penetrating vibration evolution behaviors of the gas-liquid coupled vortex flow," *Energy*, vol. 292, article 130236, 2024.

[29] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004*, pp. 35–40, San Jose, CA, USA, 2004.

[30] G. Xie, G. Zeng, J. Jiang, C. Fan, R. Li, and K. Li, "Energy management for multiple real-time workflows on cyber–physical cloud systems," *Future Generation Computer Systems*, vol. 105, pp. 916–931, 2020.

[31] G. Xie, G. Zeng, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel

applications in heterogeneous cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 62–75, 2017.

[32] E. Nogues, M. Pelcat, D. Menard, and A. Mercat, "Energy efficient scheduling of real time signal processing applications through combined DVFS and DPM," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pp. 622–626, Heraklion, Greece, 2016.

[33] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–21, 2013.

[34] B. Hu, Z. Cao, and M. Zhou, "Scheduling real-time parallel applications in cloud to minimize energy consumption," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 662–674, 2019.

[35] J. Li, G. Xie, K. Li, and Z. Tang, "Enhanced parallel application scheduling algorithm with energy consumption constraint in heterogeneous distributed systems," *Journal of Circuits, Systems and Computers*, vol. 28, no. 11, article 1950190, 2019.

[36] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control," *Journal of Grid Computing*, vol. 11, no. 4, pp. 633–651, 2013.