Received 18 August 2022, accepted 29 August 2022, date of publication 1 September 2022, date of current version 12 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3203590

RESEARCH ARTICLE

Worst-Case Response Time Analysis of Multitype DAG Tasks Based on Reconstruction

CHEN SHUSHAN[1], XIAO FENG[1], HUANG SHUJUAN[1], ZHANG WENJUAN[2], HAN XINGXING[1], AND LI TIANSEN[1]
[1]School of Computer Science and Engineering, Xi'an Technological University, Xi'an, Shanxi 710072, China
[2]School of Sciences, Xi'an Technological University, Xi'an, Shanxi 710072, China

Corresponding author: Xiao Feng (xffriends207@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 62171361, in part by the General Project of the Key Research and Development Plan of Shaanxi Province under Grant 2022GY-119, in part by the Scientific and Technological Project of Shaanxi Province under Grant 2021JM-440, in part by the Industrial Science and Technology Research Project of Shaanxi Province under Grant 2020GY-066, and in part by the Weiyang District Science and Technology Project under Grant 201925.

ABSTRACT With the wide application of heterogeneous multi-core processor real-time systems, the existing analysis methods of worst-case response time (WCRT) overestimate the blocking information among tasks, resulting in a rather pessimistic estimation. To improve the accuracy of the WCRT, we propose a reconstruction-based WCRT analysis method for multi-type directed acyclic graph (DAG) tasks scheduling algorithm(RMDS). The RMDS algorithm comprises the following steps: First, we unitize all task nodes in the multi-type DAG task; Then, we use key factors as task priorities to schedule tasks and reconstruct the DAG task model into a parallel node segment model; Finally, we estimate the WCRT of multi-type DAG tasks according to the parallel node segment model to assess task schedulability. To verify the performance of our algorithm, we compared it with traditional algorithms. RDMS showed an acceptance rate 6.13% higher and its overall performance increased by 25.95% in comparison with traditional algorithms.

INDEX TERMS Worst-case response time, reconstruction, task scheduling, RMDS, multitype DAG tasks.

I. INTRODUCTION

Real-time embedded systems are increasingly being implemented on heterogeneous multi-core processor platforms to meet their diverse and high computing demands [1] owing to their continuous improvement of processor performance. Real-time systems based on heterogeneous multi-core processors have been successfully applied in the fields of digital signal processing [2], image processing [3], [4], and distributed systems [5], and have achieved remarkable results. Resource allocation [6] and energy consumption [7] in the process of task scheduling based on heterogeneous multi-core processor platforms are always the key issues to be faced. To address these problems, this study adopts a migration method that can avoid cache misses and processor task state transitions. The study focuses on the overhead non-migration scheduling method [8].

The worst-case response time (WCRT) is the maximum time spent by all DAG tasks on a heterogeneous multi-core platform under the corresponding DAG task scheduling policy. This bound is used to judge whether a current DAG task can be scheduled and completed within the specified time and is an important evaluation index to measure the quality of DAG task scheduling algorithms. Current methods to calculate the upper bound of worst response times overestimate the required overhead time and are thus too pessimistic.

The WCRT of each task is the most important indicator to measure the real-time performance of the system. The schedulability of the task is assessed by its WCRT, which provides the basis for real-time scheduling. Since the real-time task scheduling is a non-deterministic polynomial (NP) complete problem [9], and only suboptimal solutions WCRT can be obtained. The WCRT computing method for real-time tasks has become a topic in the field of embedded research.

Directed acyclic graphs (DAG) are widely used to model applications executed on heterogeneous multi-core platforms. Typically, applications are divided into subtasks that

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

are scheduled to be executed on a set of available processor cores. Each vertex of the DAG corresponds to an execution sequence, and each directed edge represents a precedence constraint between vertices. The critical path is the length of the path with the longest execution time in the entire DAG task, which represents the lower limit of the WCRT.

Previous studies [10], [11], [12], [13] analyzed the execution sequence of the tasks, studied the relationship among the execution time of tasks, the length of the critical path, and the WCRT, and helped improve the effect of the task execution time on the WCRT for polynomial solving tasks. Using polynomial needs to consider the influence of different factors on the WCRT, and the calculation method is complicated. Other studies [14], [15], [16], [17], [18], [19], [20] analyzed the tasks latest deadline and earliest start time to calculate the task priority and generated a priority list to obtain the WCRT according to that sequence. Still other studies [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31] used a heuristic algorithm to dynamically simulate the task scheduling problem and generated the optimal scheduling sequence through continuous iterations to obtain the WCRT. This method requires a large amount of computation to iteratively generate the task scheduling sequence. The structure of the task graph is complex, and it is easy to fall into a local optimum whereas ignoring the global optimum scheduling goal.

This paper focuses on the multi-type DAG model, and proposes a reconstruction-based multi-type DAG task scheduling method (RMDS) and a WCRT analysis method.

(1) First, the DAG task graph unitizes the tasks, unifies the execution time of the tasks, and improves the accuracy of obtaining blocking information during task execution;

(2) Second, a key factor calculation method is proposed as the priority index of task nodes, make the task scheduling order globally optimal.

(3) Third, each task node in the DAG is assigned to the corresponding parallel node segment, so that the DAG task model is reconstructed into a parallel node segment [12] model, optimize the scheduling process of tasks and improve the efficiency of task allocation.

(4) Finally, the WCRT of multi-type DAG tasks is estimated based on the parallel node segment information to acquired task WCRT is more accurate.

The experimental results show that the RMDS algorithm proposed here improves the schedulability assessment accuracy by 25.95% compared with the DTA algorithms.

This paper is organized as follows: Section 2 discusses related work; Section 3 introduces the system and task model; Section 4 describes the WCRT analysis method proposed in this paper; Section 5 compares the proposed algorithm with existing algorithms and verifies the superiority and correctness of this method; conclusions are stated in Section 6.

II. RELATED WORK

Many experts and scholars have made significant research on how to efficiently obtain a more accurate WCRT for real-time tasks.

A. POLYNOMIAL CALCULATION

Jaffe *et al.* [10] proposed a calculation method for the WCRT of real-time tasks for the scheduling problem of different types of tasks on different processors, but the obtained WCRT did not consider the self-sustainability problem that get a larger WCRT. Han *et al.* [11] proposed two WCRT analysis methods by studying multi-type DAG tasks, which solved the non-self-sustainability problem of the algorithm [10] and improved the WCRT calculation accuracy. The algorithm overestimates the interference of non-critical path nodes to the execution of critical path nodes. Melani *et al.* [12] analyzed the DAG task model with conditional structure and used the worst-case execution time and longest path length of the task to characterize the complex structure of the conditional task graph and proposed the concept of parallel node segments. The node segment divides the execution interval of the task and calculates its WCRT. The algorithm does not consider the task execution interval defined by the parallel node segment, which will cause a large number of processors to be idle waiting for the execution of other task nodes in the same node segment, and the WCRT obtained is less accurate. Serrano *et al.* [13] proposed a DAG conversion strategy for DAG tasks that support parallel and heterogeneous computing. This strategy avoids the interference between parallel tasks by allocating some of them to the accelerator for execution, reduces the idle waiting time of the processor, and obtains the WCRT of the task, but this method does not consider the possibility that the assignment of different tasks leads to different results.

B. SIMULATION SCHEDULING

Axer *et al.* [14] studied fork-join tasks with fixed priority and proposed a WCRT analysis method based on fixed-priority tasks, considering the interference of tasks with higher priority. The algorithm is a trade-off inadequate consideration of priorities and task selection results in poor accurate WCRTs. Maia *et al.* [15] proposed a schedulability analysis method based on global fixed-priority tasks on multiprocessors, using task decomposition and sliding window to derive the interference information between tasks, and calculated the worst-case response of each task time, but the time complexity of the algorithm is high, and it cannot analyze large-scale real-time tasks. Yang *et al.* [16] proposed an end-to-end response time analysis method by studying DAG tasks that allow parallelism within tasks, calculating the WCRT in the case of priority scheduling of tasks with the earliest deadline. The WCRT obtained by the algorithm is not very accurate. Chang *et al.* [17] proposed a critical assignment strategy for the schedulability problem of multi-type DAG task models, assigning a criticality to each subtask as the priority constraint of task execution to obtain the WCRT. Nogd *et al.* [18] proposed a response time analysis method based on fixed priority and ordered spin lock for the problem of lack of response time analysis for non-preemptive global scheduling of shared resources. However, the WCRT obtained by the algorithm fluctuates greatly in different situations. He *et al.* [19] proposed a response time analysis method for DAG tasks based on arbitrary priority to solve the problem that DAG task scheduling is constrained by priority, high priority tasks are scheduled first, which improves the efficiency of finding the tasks' WCRT, but the accuracy was poor. Sun *et al.* [20] showed that the traditional real-time scheduling algorithm of OpenMP tasks cannot guarantee the upper bound accuracy of the response time and proposed a hierarchical scheduling algorithm (HS) to schedule OpenMP tasks and gave a response time analysis method based on the HS algorithm, but their algorithm has great limitations in scheduling flexibility and response time analysis accuracy.

C. HEURISTIC ALGORITHM

According to the difference of task classification, experts have invested a lot of research on the scheduling problem of related tasks [21], independent tasks [22], and periodic tasks [23], [24], using heuristic algorithms.

Moghadam *et al.* [25] proposed a simulation-based response time analysis method for the high complexity of industrial real-time systems, using reinforcement learning to simulate the execution scenario of the program to estimate the WCRT, but the algorithm requires a significant time to train. Sun *et al.* [26] proposed a linear-time dynamic programming algorithm to estimate the WCRT and achieved good results, although the algorithm's high complexity requires significant computational time. He *et al.* [27] proposed to control the execution order of tasks through vertex priority assignment, and designed a heuristic algorithm for proper task priority assignment. In order to improve the schedulability of the system, the algorithm has a large time overhead for DAG tasks with a large number of nodes. Skr *et al.* [28] showed that the heuristic algorithm can only reach the local optimum, and the result deviates from the optimal solution. An ILP-NC algorithm based on integer linear programming is proposed. This algorithm reduces the number of constraints required for task scheduling and improves its performance by modifying the number of processors set by the platform and the deadline for a given task. But the computations needed for this method are highly complex. Roy *et al.* [29] proposed a low-cost heuristic algorithm CC-TMS to solve the problem of high computational complexity of the optimal solution of task scheduling by integer linear programming, which reduces the algorithm for finding the optimal solution of task scheduling. However, when scheduling more complex tasks, the algorithm only considers how to quickly obtain the scheduling order and ignores the scheduling time. Devaraj *et al.* [30] addressed the problem that traditional heuristic task scheduling algorithms cannot consider all necessary conditions, and proposed a fault-tolerant scheduling method based on supervisory control and designed a search strategy to maximize fault-tolerance, thereby improving task scheduling. But this algorithm is very time consuming.

Current research uses different scheduling strategies to solve the task scheduling problem, and has achieved great results, but the results of the existing methods to obtain the worst response time of the task are pessimistic, and they do not consider the impact of other tasks in the task scheduling process. In view of the above, this study proposes a worst-case response time analysis method for multi-type DAG tasks based on reconstruction, which improves the accuracy of task response time and reduces the time overhead of task scheduling.

III. SYSTEM MODEL AND RELATED DEFINITIONS

A. PROCESSOR MODEL

Here we assume that the heterogeneous multi-core platform consists of K different types of processors, represented by $H = \{H_1, H_2, \ldots, H_K\}$. The number of processor cores of each type is not necessarily the same, so let m_k represent the number of processor cores of the kth type, that is $m_k = |H_k|$. All cores can execute tasks in parallel. The computing speed at which each processor core executes the corresponding task node is one unit of time.

B. TASK MODEL

The multi-type DAG task model can be represented as a quadruple $G = \{V, E, P, c\}$, where $V = \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$ is the set of nodes, each node v_i represents the i^{th} task in the DAG, and $n = |V|$ is the number of tasks in the DAG. E is the set of DAG edges, whose elements are two-dimensional matrix, (e_{ij}) for $1 \leq i, j \leq 2$, and $e_{ij} = 1$ indicates that there is a predecessor dependency between the nodes (v_i, v_j), where v_i is called the predecessor node of v_j and v_j the successor node of v_i, that is, v_j can only be executed after v_i is executed. The relationship between task nodes and processors is denoted by P, all tasks are to be executed on a fixed type of processor. In the processor modeling stage, each type of processor is marked, and tasks are assigned to the processor with the fastest execution speed by default, and each node has a type of processor. For example, $P(v_i) = k, (k \in [1, K])$ means that the node v_i must be executed on set H_k of processor cores of type k. The execution time of each task node is denoted by c, and $c(v_i)$ is the maximum execution time of the node v_i on its corresponding type of processor H_k.

Definition 1: Let $prep(v_i)$ denote the set of direct predecessor tasks of the task v_i in the multiple-type DAG task. If $pred(v_i) = \emptyset$, then the node is called the start node and is denoted v_{start}. The set $succ(v_i)$ denotes the set of tasks that directly succeed v_i. If $succ(v_i) = \emptyset$, then the node is called the termination node and denoted v_{end}.

In a DAG task, if there are multiple start and end nodes, a virtual start node and end node with an execution time of 0 is added.

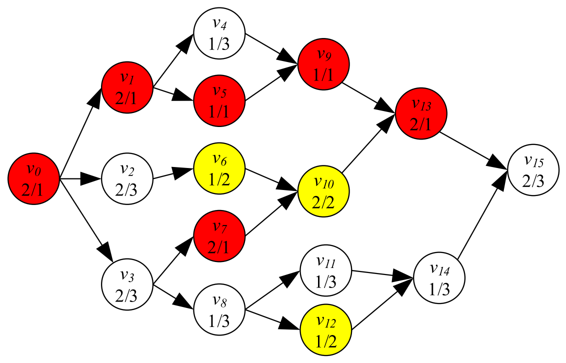


FIGURE 1. DAG task graph example *G*.

When a continuous execution sequence includes both the start and the end nodes it is called a complete path l. $L = \{l_1, l_2, l_3, \ldots, l_M$ represents the set of all complete paths, and l_{max} is the longest complete path, which is called the critical path, $len(l_m)$ represents the time cost of executing the complete path l_m, and $len(G)$ is expressed as the time overhead of the longest path l_{max} in the DAG, which determines the lower bound of the WCRT. We have:

$$len\left(l_m\right) = \sum\nolimits_{v_i \in l_m} c(v_i) \tag{1}$$

$$len(G) = \max_{l_m \in L} \{len\left(l_m\right)\} \tag{2}$$

Definition 2: Let $des(v_i)$ be the set of descendent nodes of v_i. If node $v_j \in des\left(v_i\right)$, then v_j must be executed after v_i and v_j has no other predecessors. All nodes, except v_{start}, are descendants of v_{start}.

Definition 3: If we consider all the connected nodes from the node v_i to the terminal node v_{end} as a subgraph of the DAG task, the longest path of the subgraph is called the local critical path $PCP(v_i)$, and $L_{PCP}(v_i)$ the local key. The local critical paths of nodes on the critical path are a subset of the critical paths of the DAG. The length of the path is:

$$L_{PCP}\left(v_i\right) = \max_{v_j \in succ(v_i)} c\left(v_i\right) + \left(L_{PCP}\left(v_j\right)\right) \tag{3}$$

Definition 4: Let's be the set of multiple parallel tasks where all tasks in the node segment are executed at the same time and different node segments are serially connected through directional arcs. By converting the DAG task into an execution sequence composed of multiple node segments, the task model is simplified. The worst execution time of all parallel node segments depends on the task node with the longest execution time:

$$c\left(segment_s\right) = \max_{v_i \in segments_i} c\left(v_i\right) \tag{4}$$

Fig. 1 shows a multi-type DAG example G running on three types of processors, where each node represents a task of the DAG, the node name of the current task and the execution time of the task are given in the node, different colors indicate the different processor cores in which they are executed; v_i is the name of the node, and the digital sub-table represents the execution of the node on the corresponding processor type

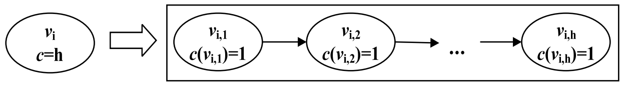


FIGURE 2. Schematic diagram of node splitting.

and the size of its execution time. In this example, $m_1 = 2$, and $m_2 = m_3 = 1$. From Definition 1 and Definition 2, we know that $prep\left(v_0\right) = \{v_1, v_2, v_3\}$, since $succ\left(v_0\right) = \emptyset$ and $prep\left(v_{15}\right) = \emptyset$, so $v_{start} = v_0$, $v_{end} = v_{15}$. The critical path is $l_{max} = \{v_0, v_3, v_7, v_{10}, v_{13}, v_{15}\}$ whose length is $len\left(G\right) = 12$. From definition 3 it can be concluded that $L_{PCP}\left(v_3\right) = 10$.

IV. RMDS ALGORITHM

To obtain a more accurate WCRT, we propose a reconstruction-based WCRT analysis method for multi-type DAG tasks (RMDS).

A. TASK PREPROCESSING

The execution time of each task node in the DAG is randomly generated. Since the execution time of task nodes is not uniform, when calculating the WCRT many processors will be idle when the task node divides the execution interval and the WCRT of the task cannot be accurately calculated. To solve the above problems, this study unitizes each task node of the DAG according to the execution time and divide all task nodes except the start node and all end nodes into multiple nodes with the same execution time preserving their sequential connection. We end up with a collection of unit nodes. To reduce the number of node splits, the execution time of a unit node is set to the greatest common divisor d of all nodes that need to be split. We normalize the tasks whose execution time is less than one unit and then perform the unitization operation of the tasks to ensure that to each of them is assigned at least one unit node.

Fig. 2 is a schematic diagram of how a node with an execution time h and $d = 1$ is split into h nodes with an execution time of d. Through node splitting, each split unit node $v_{i,j}$ is represented as the j^{th} unit node of the i^{th} node.

The Fig. 3 shows the task graph G' after unitizing the task graph G. All nodes in G except the start node v_{start} and the end node v_{end} are split into unit nodes. The other attributes of each unit node are the same as those before the split, and all unitized nodes are connected by edges.

The number of nodes $|V'|$ of the DAG after unitization is:

$$\left|V'\right| = \sum\nolimits_{i=1}^{|V|-2} \frac{c\left(v_i\right)}{d} + 2 \tag{5}$$

Equation (5), $|V|$ is the number of nodes in the DAG before unitization, and d is the greatest common divisor of the execution time of the split nodes.

B. TASK PRIORITIES IN RMDS

The accuracy of improving the WCRT of task scheduling depends largely on assigning priorities to tasks. If a task is scheduled first, resulting in a shortened WCRT, the task must

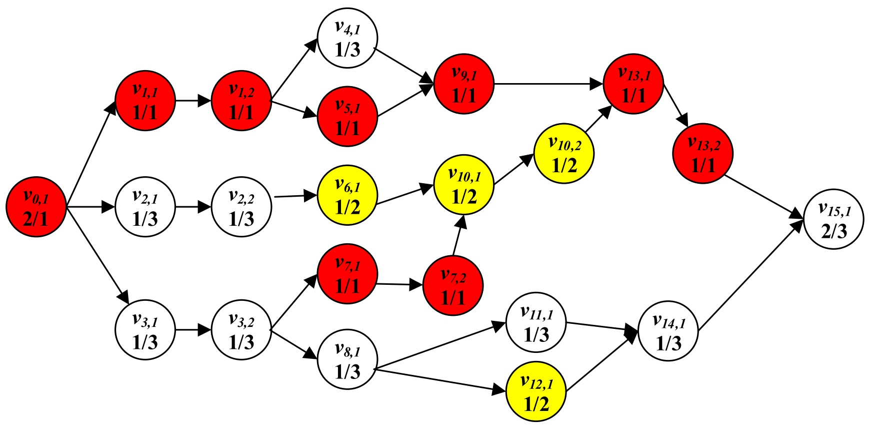


FIGURE 3. Unitized task graph G'.

have a high priority. This study proposes key factors as task priority indicators and evaluates the impact of current node priority allocation on subsequent nodes by calculating key factors and prioritizes tasks with large key factors. The key factor is mainly composed of two parts.

(1) The local critical path length L_{PCP} from the current node v_i to the terminal node v_{end}, as shown in Definition 3, which represents the WCRT lower limit of the DAG subgraph formed by the current node and the terminal node. It is used as the main reference factor for task scheduling.

(2) The criticality $\mu(v_i)$ of the current node to subsequent nodes. If the current node has many descendant nodes, preferentially assigning this node can effectively reduce the idle waiting time of the processor. To ensure the minimum overall response time, $\mu(v_i)$ is used as a secondary indicator of task priority assignment, as shown in (6):

$$\mu(v_i) = \max_{v_j \in succ_{v_i}} \mu(v_j) + \max_{k\in[1,K]} \left(\sum_{\substack{v_j \in succ(v_i) \\ P(v_j)=k}} \frac{c(v_j)}{m_k} \right) \tag{6}$$

where $succ(v_i)$ is the direct successor node set of task node v_i, $P(v_j) = k$ indicates that task node v_j is scheduled on the processor core of type k, and m_k is the number of processor cores of type k. $\mu(v_{end})$ is defined as 0.

In summary, for the task scheduling priority assignment problem, the calculation method of the key factor is shown in the following equation:

$$Iprio(v_i) = L_{PCP}(v_i) + \mu(v_i) \tag{7}$$

Table 1 is a list of task priorities based on key factors.

According to Table 1, the key factors proposed in this paper expand the range of priorities and subdivide the degree of influence of each task on subsequent task scheduling.

C. TASK GRAPH RECONSTRUCTION

Based on the priority list of DAG tasks shown in Table 1, the DAG is reconstructed into a parallel node segment model, the execution interval of each task node is limited, and the priority selection strategy is used to ensure that tasks are in a relatively optimal time period scheduling, thereby reducing the execution time of task nodes. The reconstruction of the DAG is divided into the following steps.

(1) Obtain nodes with 0 in-degree nodes from the DAG, and form a ready node set, and execute step (2).

(2) Allocate the ready node set to the node segments according to the size of the key factor and assign the node with the largest key factor first. If the key factor is the same, go to step (3), otherwise, delete the currently allocated node and go to step (6).

(3) The nodes with the long local critical paths are preferentially allocated to the node segments. If the local critical

TABLE 1. Task priority list.

Node	$L_{PCP}(v_i)$	$\mu(v_i)$	$Iprio(v_i)$
$v_{0,1}$	12	11.5	23.5
$v_{1,1}$	8	5.0	13.0
$v_{1,2}$	7	4.5	11.5
$v_{2,1}$	9	7.0	16.0
$v_{2,2}$	8	6.0	14.0
$v_{3,1}$	10	9.5	19.5
$v_{3,2}$	9	7.5	16.5
$v_{4,1}$	6	3.5	9.5
$v_{5,1}$	6	3.5	9.5
$v_{6,1}$	7	5.0	12.0
$v_{7,1}$	8	5.5	13.5
$v_{7,2}$	7	5.0	12.0
$v_{8,1}$	5	5.0	10.0
$v_{9,1}$	5	3.0	8.0
$v_{10,1}$	6	4.0	10.0
$v_{10,2}$	5	3.0	8.0
$v_{11,1}$	4	3.0	7.0
$v_{12,1}$	4	4.0	8.0
$v_{13,1}$	4	2.5	6.5
$v_{13,2}$	3	2.0	5.0
$v_{14,1}$	3	2.0	5.0
$v_{15,1}$	2	0	2.0

Algorithm 1 RMDS

```
Input: $G = \{V, E, P, C\}$, System
Output: Segment
1  Initialize all nodes of DAG
2  $Seglength = 0$
3  For each $v_i \in G$
4      If there are multiple nodes that degree is 0
5          Create a node $v_{input} (v_{output})$ with $c(v) = 0$
6          $succ(v_{input}) = v_i$ or $prep(v_{output}) = v_i$
7      End if
8  End for
9  Calculate the greatest common divisor d of all the nodes
10 For each node $v_i \in G$ && $v_i \neq v_{start}, v_{end}$
11     If $c(v_i) > d$
12         $v_i \rightarrow \{v_{i,1}, v_{i,2}, \ldots, v_{i,c(v_i)/d}\}$
13     End if
14 End for
15 calculate the priority of each node Eq. 6
16 While $G \neq NULL$
17     Initialize the usage of each processor core and clear readyList
18     For each $v_{i,j} \in G and prep(v_{i,j}) = \emptyset$
19             Input $v_{i,j}$ to readyList
20     End for
21     Rank(readyList) according to priority
22     For each $v_{i,j} : readyList$
23             If $P(v_{i,j}) = k$ and Type of k processor idle
24                 Input $v_{i,j}$ into $Segment[Seglength]$ and delete $v_{i,j}$ from G
25             End if
26     End for
27     $Seglength = Seglength + 1$
28 End while
29 Return Segment
```

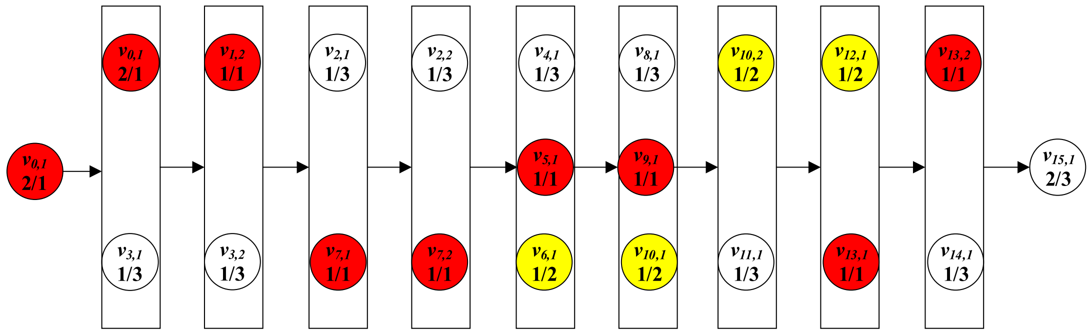


FIGURE 4. Segment model.

paths have the same length, step (4) is performed; otherwise, the currently allocated nodes are deleted, and step (6) is performed.

(4) Priority is given to node segments with the largest number of direct successor nodes; if the number of direct successor nodes is the same, perform step (5); otherwise, perform step (6).

(5) Allocate nodes in the order of the node sequence, delete the currently allocated nodes, and execute step (6).

(6) If the current node segment has no idle processor core or the current ready node set is empty, repeat steps (1-6) until the node in the DAG is empty.

According to the above method, the DAG is transformed into a parallel node segment model without affecting the

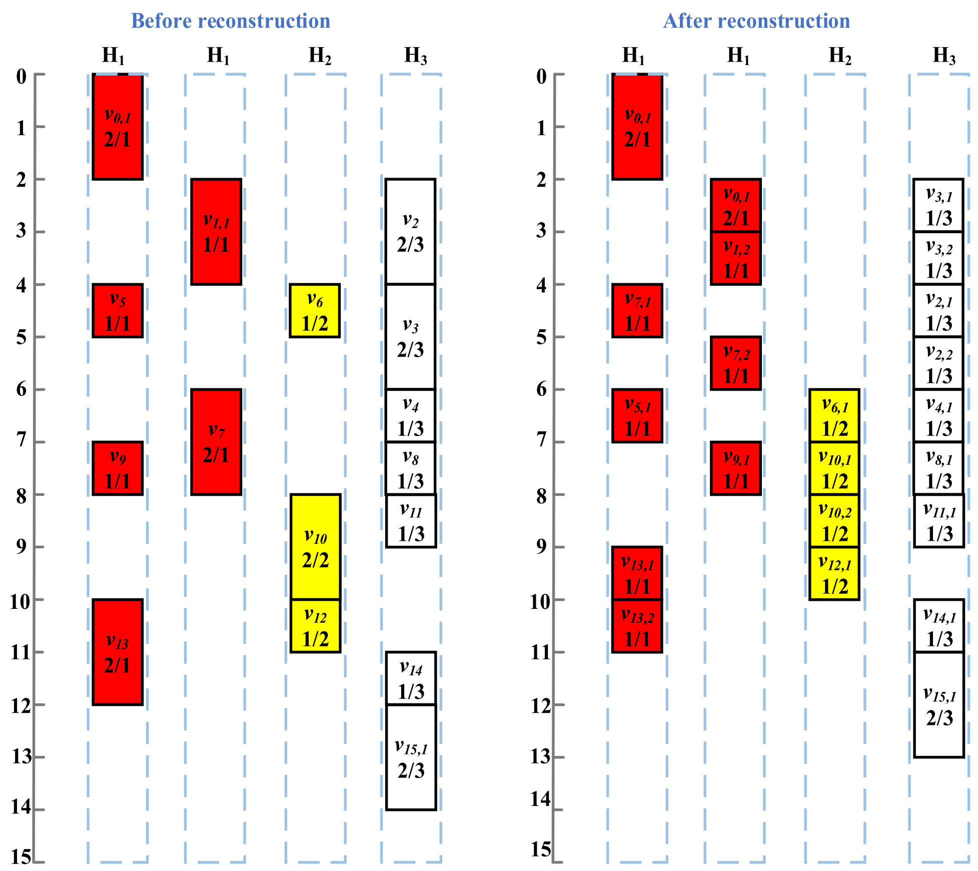


FIGURE 5. Comparison of task simulation scheduling before and after algorithm processing.

dependencies among task nodes. The execution interval of each task is specified to avoid unnecessary blocking between tasks, thereby reducing the WCRT of the task.

Pseudocode of the RMDS algorithm.

Lines 1-2 of the algorithm initialize all nodes in the DAG input by the algorithm, that is, perform zero-setting operations on indicators such as the priority of each node, and initialize the length of the node segment to 0. Lines 3-8 normalize the DAG G and transform the input irregular task graph into a standard DAG with only one start node and end node. Line 9 obtains the greatest common divisor d of the execution time of all nodes except the start node and the termination node by calculation. Lines 10-14 unitize the task graph G with the information obtained earlier. Line 15 calculates the priority of each node using 6. Lines 16-28 converts the DAG model into a parallel node segment model. Line 17 initializes the processor and clears the ready list. Lines 18-20 get the current list of ready tasks. Line 21 orders the list of ready tasks by priority. Lines 22-28 place the node into the current node segment and delete the corresponding node in the DAG until no node that meets the requirements exist or the current node segment has no corresponding idle processor core. By analyzing the algorithm, during its execution the nodes will be split into unit nodes, and then the corresponding operations are performed on the tasks (lines 16-28). Through analysis, the outermost loop is in the worst case, the entire DAG task is connected in series, it needs to loop V' times to complete the task scheduling. In this case, the inner loop time is 1, the total time complexity is $O(\left|V'\right| + 1)$, and becomes $O(\left|V'\right|)$ after eliminating the constant. In the best case of the outer loop, all nodes are scheduled in parallel. At this time, the time complexity of the task is $\left[\frac{|V'|}{\sum_{k\in[1,K]} m_k} + 1, \frac{|V'|}{m_k} + 1\right]$, so the time complexity is $O(\left|V'\right|)$. The task node is divided into V' unit nodes, so the space complexity is $O(\left|V'\right|)$.

The task graph is converted into a segment model by the RMDS algorithm, and the result is shown in fig. 4.

Fig. 5 shows the comparison of task simulation scheduling time after and before model conversion. It can be seen that the WCRT obtained by converting G using the RMDS algorithm one unit shorter of the traditional algorithm.

D. WCRT ANALYSIS

The RMDS algorithm proposed in this paper converts the DAG task model into a segment model and divides the execution interval of each task node to obtain the WCRT of the task according to the information of the node segment. Compared with the WCRT analysis method that has been proposed so far, this paper reconstructs the DAG task model into a parallel node segment model. During the reconstruction process, the number of parallel nodes in each node segment is set, thereby delaying start time. Changing the start execution time of some nodes optimizes the scheduling sequence of DAG tasks, thereby reducing the WCRT. Through the model conversion strategy, the WCRT of this paper can be obtained using the following equations:

$$\boldsymbol{WCRT} = \sum\nolimits_{seg \in Segment} \boldsymbol{c}\,(\boldsymbol{seg}) \tag{8}$$

(8) can be simplified into the equation:

$$\boldsymbol{WCRT} = \boldsymbol{d} * (\boldsymbol{lenth}\,(\boldsymbol{Segment}) - \mathbf{2}) + \boldsymbol{c}\,(\boldsymbol{v}_{start}) + \boldsymbol{c}\,(\boldsymbol{v}_{end}) \tag{9}$$

where d is the greatest common divisor of the execution time of all task nodes in the unitized DAG, and $lenth\,(Segment)$ represents the total length of the parallel node segment transformed by the DAG.

Theorem: The WCRT of a DAG task is bounded by

$$\boldsymbol{R}\left(\boldsymbol{G}'\right) \leq \boldsymbol{d} * (\boldsymbol{length}\,(\boldsymbol{Segment}) - \mathbf{2}) + \boldsymbol{c}\,(\boldsymbol{v}_{start}) + \boldsymbol{c}\,(\boldsymbol{v}_{end}) \tag{10}$$

Proof: According to the calculation method of WCRT, the WCRT of the DAG task depends on the execution time d of the unitized task and the number of node segments after task transformation. Assuming that the actual execution time of a node in the multi-type DAG is less than the WCRT of the task node, then $c(v_i) \geq c'(v_i)$.

If the number of node splits remains the same, then $d > d'$ and $length\,(Segment) = length\left(Segment'\right)$. According to (9), it can be known that $WCRT \geq WCRT'$ holds.

To sum up, when the actual execution time of the task node is less than the worst execution time of the task, the actual execution time never exceeds the upper bound of the WCRT proposed in this paper. The theorem is proven.

This paper proposes a WCRT analysis method for multi-type DAG tasks based on reconstruction. By adjusting the scheduling order of different task nodes and limiting the scheduling interval of different nodes, the interference between tasks is reduced, thus reducing the task scheduling time and obtaining a more accurate upper bound of the WCRT.

V. EVALUATION

In order to verify the performance and accuracy of the RMDS algorithm. A comparative analysis with four algorithms: JEF [10], HAN-1 [11], HAN-2 [11], and DTF [17] is presented to verify the performance of the RMDS algorithm. The DAG is randomly generated using the layer-by-layer method [31]. The WCRT of each algorithm on different task graphs is obtained through experimental simulation. The results are compared using the following metrics.

(1) Acceptance Ratio (AR): Specifies the WCRT upper bound as the ratio between the number of tasks that can be scheduled, and the total number of tasks generated. The higher the acceptance rate, the more accurate the worst-case response of the algorithm.

$$\boldsymbol{AR} = \sum\nolimits_{G_i \in set} \frac{\boldsymbol{count}\,(\boldsymbol{WCRT}\,(\boldsymbol{G}_i) < \boldsymbol{D}_i)}{|\boldsymbol{set}|} \tag{11}$$

In (11), the set is the DAG task set, $WCRT\,(G_i)$ is the WCRT of the i^{th} task graph, D_i is the deadline of the current task graph G_i, and $|set|$ represents the amount of data in the current task set.

(2) Average WCRT ($AWCRT$): The experiment is random, and multiple sets of data are averaged to obtain the average WCRT of the task. The performance of the algorithm is assessed by comparing the average WCRT obtained by different algorithms.

$$\boldsymbol{AWCRT} = \sum\nolimits_{G_i \in set} \frac{\boldsymbol{WCRT}\,(\boldsymbol{G}_i)}{|\boldsymbol{set}|} \tag{12}$$

(3) Normalized upper bound (NR): This performance index is based on the JEF algorithm as a reference and performs a normalized comparison of the other algorithms based on the WCRT obtained by the JEF algorithm. If the normalized result is smaller, the better the algorithm is.

$$\boldsymbol{NR} = \sum\nolimits_{G_i \in set} \frac{\boldsymbol{WCRT}\,(\boldsymbol{G}_i)}{\boldsymbol{JEF}\,(\boldsymbol{G}_i)} \tag{13}$$

(4) *Speedup*: The ratio of the task sequence execution time to the WCRT, to obtain the acceleration of the current algorithm for task scheduling.

$$\boldsymbol{Speedup} = \frac{\boldsymbol{vol}\,(\boldsymbol{G})}{\boldsymbol{WCRT}} \tag{14}$$

In (14), $vol\,(G)$ represents the time overhead of sequential execution of tasks.

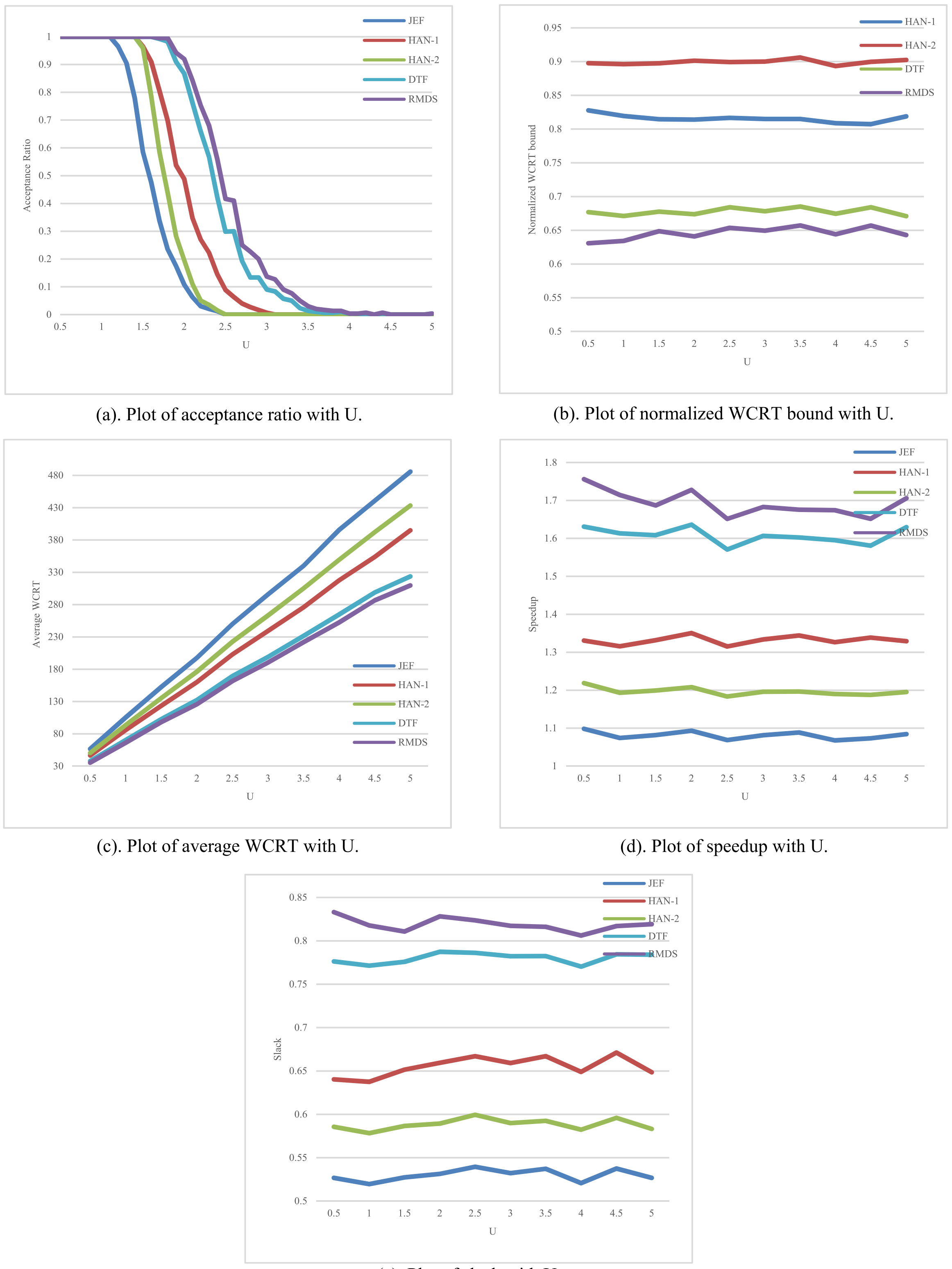
(5) *Slack*: *Slack* is a measure of the robustness of a task scheduling algorithm, which reflects the uncertainty of the WCRT of a task generated by an algorithm scheduling. The definition of Slack is shown in eq. (15).

$$\boldsymbol{Slack} = \frac{\sum_{i=1}^{n} \boldsymbol{WCRT} - \boldsymbol{Inter}\,(\boldsymbol{v}_i) - \boldsymbol{Exit}\,(\boldsymbol{v}_i)}{\boldsymbol{n}} \tag{15}$$

where n is the number of task nodes, $Inter(v_i)$ denotes the length of the longest path from the entry node v_{start} to the task node v_i (excluding the task v_i), and $Exit(v_i)$ represents the length of the longest path from the task node v_i to the termination node v_{exit}.

The algorithm considers the influence of different numbers of processor types and the number of cores. When creating a processor model, the number of processor types K is randomly selected in the range of [2, 10], and the number of cores for each processor, m_k, is randomly selected as a number between 2 and 8.

The first parameter is the parallelism factor Pr [17], which controls the probability of the existence of an edge between the current node and its subsequent nodes in the DAG task, to judge the influence of the parallelism of the task on the algorithm; the parallelism factor Pr is randomly selected between [0.04, 0.2].



(a). Plot of acceptance ratio with U.

(b). Plot of normalized WCRT bound with U.

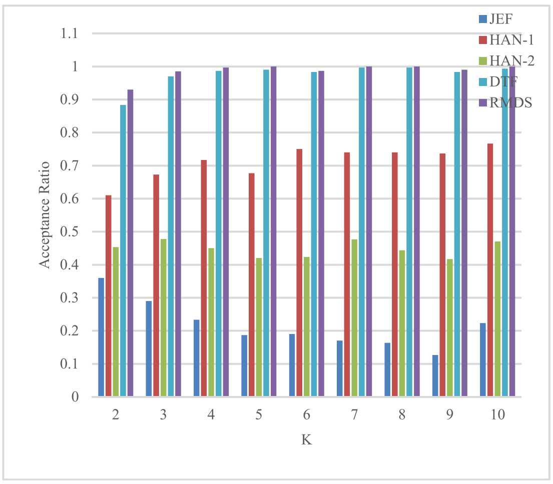
(c). Plot of average WCRT with U.

(d). Plot of speedup with U.

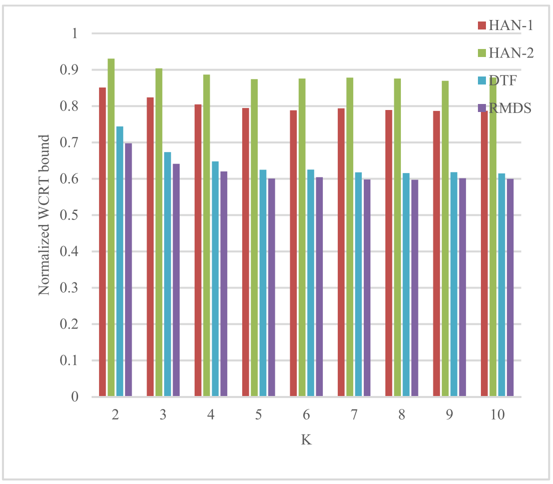
(e). Plot of slack with U.

FIGURE 6. Variation of indicators of different algorithms with U.

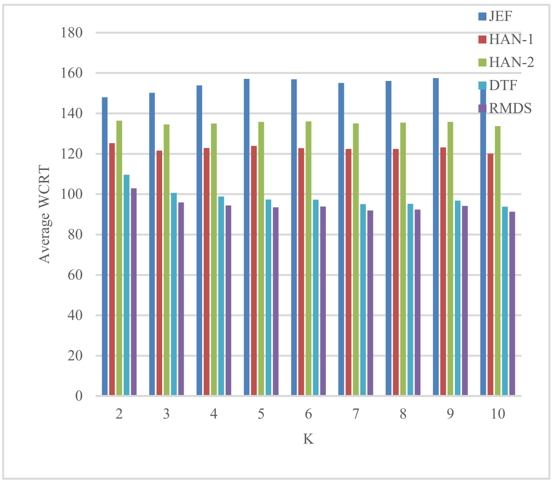
The second parameter is the number of task nodes |V|, which is used to control the impact of the scale of DAG tasks on the upper bound of the WCRT. The number of task nodes |V| is randomly selected between [10, 50].



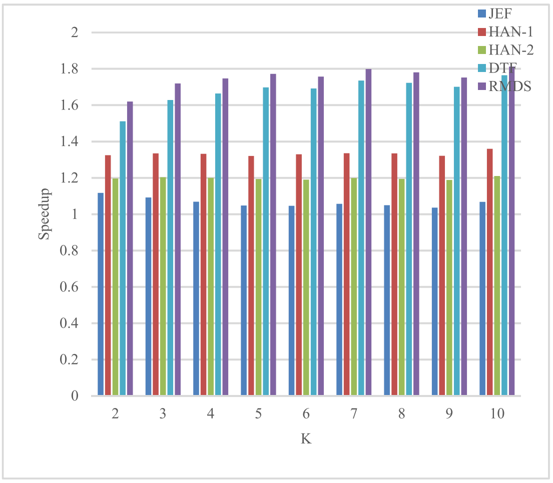
(a). Plot of acceptance ratio with K.



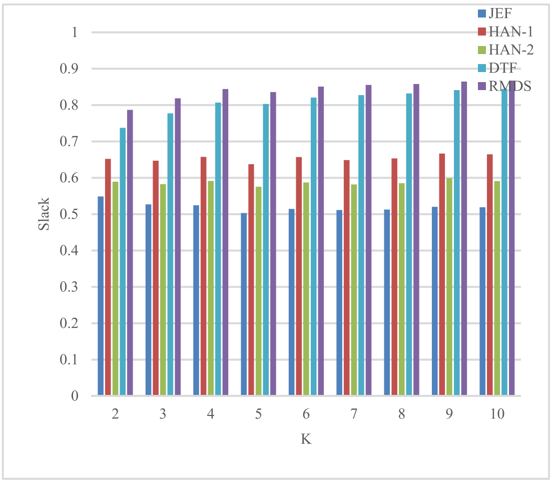
(b). Plot of normalized WCRT bound with K.



(c). Plot of average WCRT with K.



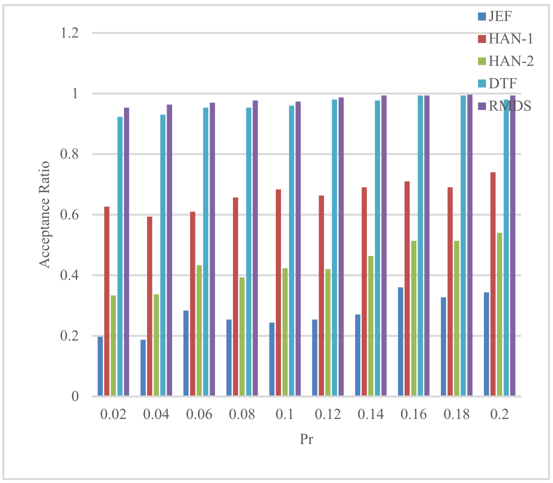
(d). Plot of speedup with K.



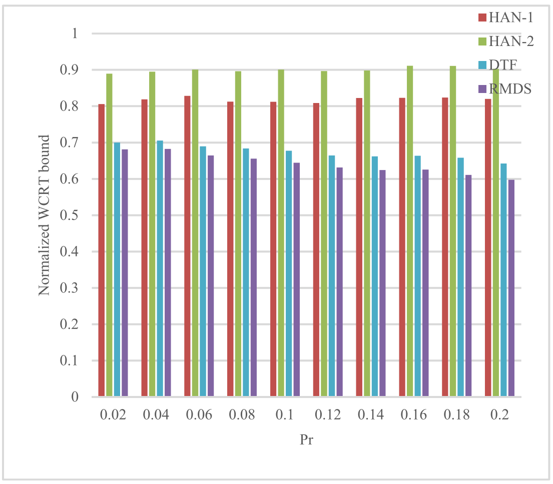
(e). Plot of slack with K.

FIGURE 7. **Variation of indicators of different algorithms with K.**

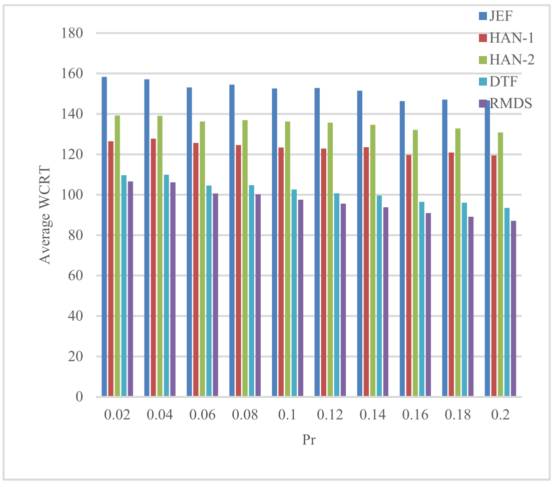
The third parameter is the task total utilization rate U. The task utilization rate of all task nodes is randomly assigned by the UuniFast method [32], and $U = \sum_{v_i \in G} u(v_i)$, to obtain the WCRT, $u(v_i)$ is the utilization of node v_i. The total task utilization U is randomly selected from [0.5, 5], and the sum of the worst execution times of all



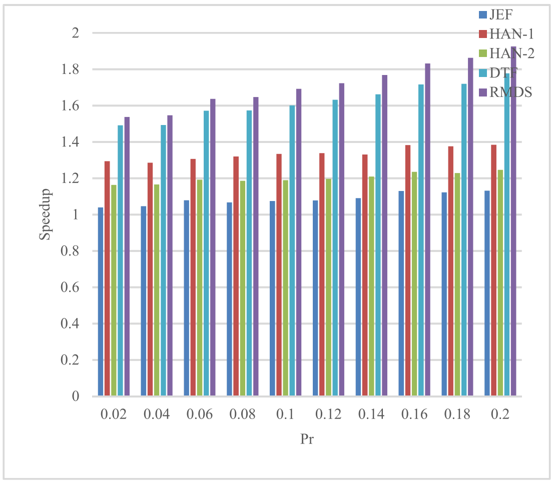
(a). Plot of acceptance ratio with Pr.



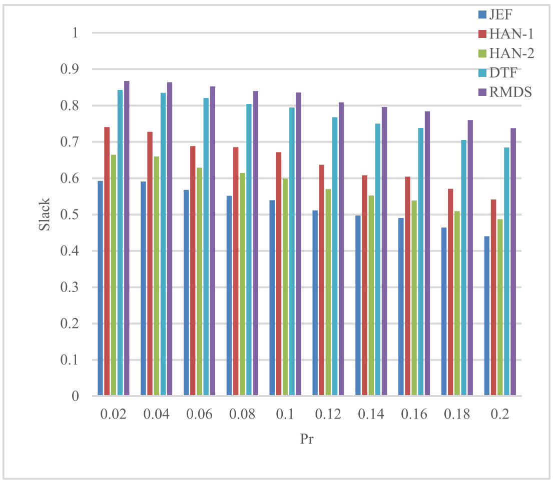
(b). Plot of normalized WCRT bound with Pr.



(c). Plot of average WCRT with Pr.



(d). Plot of speedup with Pr.

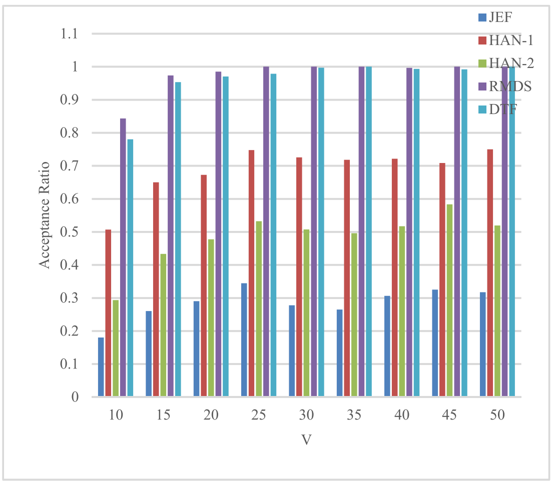


(e). Plot of slack with Pr.

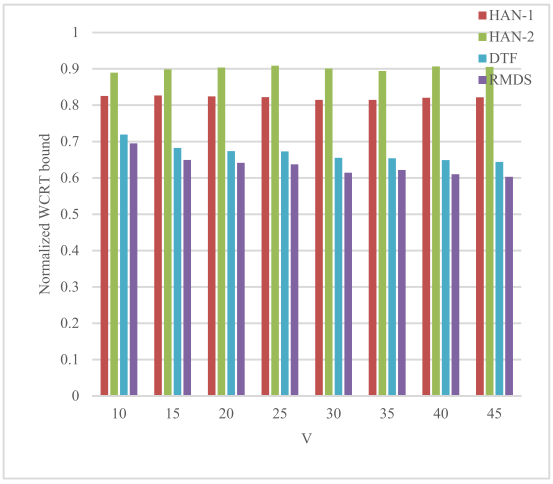
FIGURE 8. Variation of indicators of different algorithms with *Pr*.

nodes of the DAG task is $vol(G) = U \times T$, where the deadline $D \leq T$.

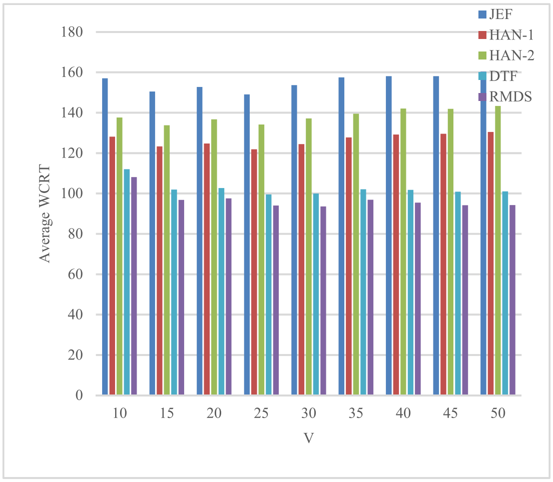
According to three parameters, the layer-by-layer method is used to randomly generate 37,500 DAGs as the dataset for the control experiment. During the experiment, the algorithm was used to conduct comparative experiments for each task graph. For example, a total of 45 groups of experiments with a step size of 0.1 were carried out for the parameter U,



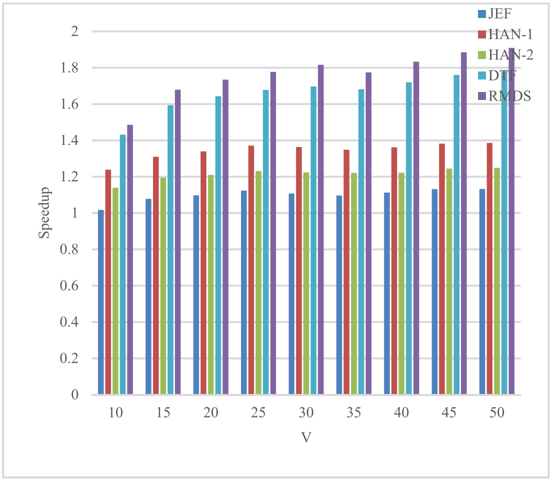
(a). Plot of acceptance ratio with V.



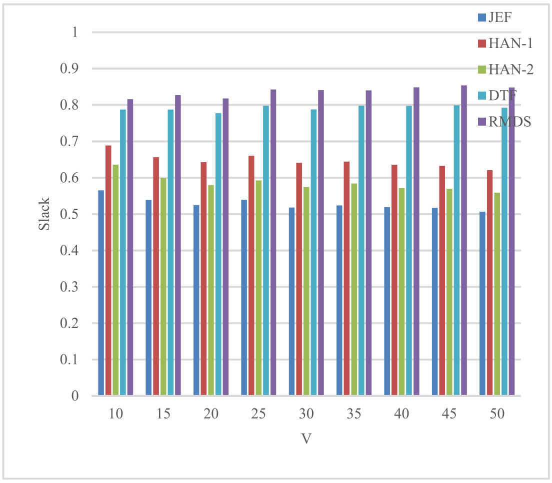
(b). Plot of normalized WCRT bound with V.



(c). Plot of average WCRT with V



(d). Plot of speedup with V.



(e). Plot of slack with V.

FIGURE 9. Variation of indicators of different algorithms with V.

for values of U in the range [0.5, 5], and each group of experiments was randomly generated. The same experiment was performed on 300 different DAG task graphs, and the experimental results were averaged.

TABLE 2. Summary of the comparison between the RMDS algorithm and other algorithms.

Algorithm	JEF	HAN-1	HAN-2	DTF
Acceptance Ratio	234.8074%	39.4602%	101.1992%	6.1293%
Average WCRT	58.7477%	28.2046%	40.9967%	4.867%
Normalized WCRT bound	∅	28.1035%	41.2983%	4.8292%
Speedup	59.4710%	29.1931%	43.7133%	5.239%
Slack	57.1461%	26.8674%	41.1600%	4.8808%

Fig. 6 shows the curves of each performance index of different algorithms as a function of the total task utilization U. Fig. 6(a) is a graph of the variation of the acceptance rate of different algorithms with U. From the experimental results in Fig. 6(a), the RMDS algorithm is more accurate than the traditional algorithm in judging the schedulability of tasks, and it shows an average improvement over the DTF algorithm up 6.13%. When U is in the range of (1.5, 4), the task acceptance rate of the RMDS algorithm is 18.04% higher than that of the DTF algorithm. Fig. 6(b) shows the variation of the normalized upper bound with U for different algorithms. From the experimental results in Fig. 6(b), the RMDS algorithm greatly reduces the WCRT of the task. Compared with the DTF algorithm, the response time obtained by the RMDS algorithm is reduced by 4.63%. Fig. 6(c) shows the variation of the average WCRT of different algorithms with U. From the experimental results in Fig. 6(c), as the total utilization of the task increases, the WCRT of the task increases linearly, From Fig. 6(c), it can be seen that the WCRT of the RMDS algorithm slightly increases compared with the traditional algorithm. The average WCRT is 4.49% lower than that of the DTF algorithm. Fig. 6(d) shows the speedup of different algorithms changing with U. From the experimental results in Fig. 6(d), the RMDS algorithm has a higher speedup for task scheduling than the existing algorithm under different task utilization rates. The MDS algorithm achieves a better performance for heterogeneous multi-core processors, and with the increase of U, the speedup of the algorithm has remained stable. By comparing the results, the speedup of the RMDS algorithm is 5.01% higher compared with the DTF algorithm. Fig. 6(e) is a comparison chart of Slack with U of different algorithms. From the experimental results in Fig. 6(e), the stability of the RMDS algorithm is higher than that of the existing algorithm. The stability of the RMDS algorithm is 4.69% higher than that of the DTF algorithm.

Fig. 7 shows the graphs of variation of the performance indicators of different algorithms varying with the number of processor types K. Fig. 7(a) is a graph showing the variation of the acceptance rate of different algorithms with K. From the experimental results in Fig. 7(a), as K increases, the algorithm has higher scheduling performance for tasks. Fig. 7(b) shows the variation of the normalized upper bound with K for different algorithms. From the experimental results in Fig. 7(b), the WCRT of the RMDS algorithm is always relatively stable with the increase of K. Solved non-self-sustaining problems. Compared with the DTF algorithm, the normalized WCRT obtained by the RMDS algorithm is reduced by 4.01%. Fig. 7(c) shows the variation of the average WCRT of different algorithms with K. From the experimental results in Fig. 7(c), the average WCRT of the RMDS algorithm is 3.98% lower than that of the DTF algorithm. Fig. 7(d) shows the variation of speedup with K for different algorithms. From the experimental results in Fig. 7(d), the speedup of the RMDS algorithm under different processor environments is always higher than that of the existing algorithm. By comparing the results, the speedup of the RMDS algorithm is 4.25% higher than that of the DTF algorithm. Fig. 7(e) shows the comparison of Slack with K for different algorithms. From the experimental results in Fig. 7(e), with the increase in the number of processor types, the stability of the RMDS algorithm remains balanced, and higher than the current one. The stability of the RMDS algorithm is 3.98% higher than that of the DTF algorithm.

Fig. 8 shows the variation of the performance index of different algorithms where Pr denotes the number of types of processors. Fig. 8(a) is a graph of the variation of the acceptance rate of different algorithms with Pr. From the experimental results of Fig. 8(a), as the number of parallel nodes of the task increases, the acceptance rate of the algorithm gradually increases. Fig. 8(b) shows the variation of the normalized upper bound with Pr for different algorithms. From the experimental results in Fig. 8(b), the WCRT obtained by the RMDS algorithm with the increase of Pr is the WCRT obtained by the RMDS algorithm which is lower than the results of traditional algorithms. Compared with the DTF algorithm, the normalized WCRT obtained by the RMDS algorithm is reduced by 5.14%. Fig. 8(c) shows the average WCRT of different algorithms as a function of Pr. From the experimental results in Fig. 8(c), tasks with high parallelism can be scheduled in a shorter time. The RMDS algorithm's average WCRT is 5.07% lower than that of the DTF algorithm. Fig. 8(d) shows the variation of speedup with Pr for different algorithms. From the experimental results in Fig. 8(d), with the increase in task parallelism, the algorithm accelerates task scheduling, and the speedup of the RMDS algorithm is 5.67% higher than the DTF algorithm. Fig. 8(e) is a comparison chart of slack with Pr of different algorithms. From the experimental results of Fig. 8(e), the higher the task parallelism, the worse the stability of the algorithm. The stability of the RMDS algorithm is 5.13% higher than that of the DTF algorithm.

Fig. 9 shows the graphs of the performance indicators of different algorithms varying with the number of tasks V. Fig. 9(a) shows the variation of the acceptance rate of different algorithms with the number of tasks V. From the experimental results in Fig. 8(a), as the number of tasks increases, the acceptance rate of the algorithm gradually increases. Fig. 9(b) shows the variation of the normalized upper bound with V for different algorithms. From the experimental results in Fig. 9(b), as the number of tasks increases, the WCRT obtained by the RMDS algorithm is always lower than the results of traditional algorithms. Compared with the DTF algorithm, the normalized WCRT obtained by the RMDS algorithm is reduced by 5.82%. Fig. 9(c) shows the variation of the average WCRT with V for different algorithms. From the experimental results in Fig. 9(c), the average WCRT of the RMDS algorithm is 5.63% lower than that of the DTF algorithm. Fig. 9(d) shows the speedup of different algorithms as a function of V. From the experimental results in Fig. 9(d), the Speedup of the RMDS algorithm is 6.03% higher than that of the DTF algorithm. As the number of tasks increases, the RMDS algorithm can also stabilize the performance of the processor. Fig. 9(e) is a comparison chart of the variation of Slack with V for different algorithms. From the experimental results in Fig. 9(e), the stability of the RMDS algorithm is always in a balanced state, which is higher than that of the existing algorithm. The stability of the RMDS algorithm is 5.72% higher than that of the DTF algorithm.

Table 2 summarizes the results of the comparison of various indicators between the RMDS algorithm and the JEF [9], HAN-1 [10], HAN-2 [10], and DTF [16] algorithms. Each value in the table represents the current RMDS algorithm. The indicator's performance improvement for the current algorithm. Based on the above experiments, the RMDS algorithm proposed in this paper has improved performance in all aspects compared with the existing algorithms. It can be seen from Table 2 that the RMDS algorithm proposed in this paper has a comprehensive performance improvement of 25.9453% compared with the DTF algorithm.

VI. CONCLUSION

This paper studies the WCRT analysis method for multi-type DAG tasks on heterogeneous platforms and proposes a reconstruction-based WCRT analysis (RMDS) for multi-type DAG tasks. In the RMDS algorithm, each task is processed as a unit, and the blocking information of each task scheduling is accurate. To determine the scheduling order of task nodes, a priority calculation method based on key factors is proposed. Finally, the DAG task graph is reconstructed into a parallel node segment model to schedule tasks. In this paper, a new scheduling method is proposed to obtain the WCRT of tasks more accurately. The experimental results show that the WCRT accuracy of the DAG task obtained by the RMDS algorithm is 25.9453% higher than that of the traditional algorithm. Due to the complexity of the structure of multi-type DAG tasks, the selection of task priority determines the accuracy of the WCRT finally obtained by the algorithm. How to choose an appropriate task priority calculation method is the focus of our next research.

ACKNOWLEDGMENT

The authors would like to thank AiMi Academic Services (www.aimieditor.com) for the English language editing and review services.

REFERENCES

[1] N. Zhou, J. Hu, and H. M. Hu, "Development trend and key technologies of multi-core processors," *Comput. Eng. Des.*, vol. 39, no. 2, pp. 393–399 and 467, 2018.

[2] W. Huang, L. Ding, G. Zhai, X. Min, J.-N. Hwang, Y. Xu, and W. Zhang, "Utility-oriented resource allocation for 360-degree video transmission over heterogeneous networks," *Digit. Signal Process.*, vol. 84, pp. 1–14, Jan. 2019.

[3] D. Shin, J. Lee, J. Lee, J. Lee, and H.-J. Yoo, "An energy-efficient deep learning processor with heterogeneous multi-core architecture for convolutional neural networks and recurrent neural networks," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS)*, Apr. 2017, pp. 1–2, doi: 10.1109/COOLCHIPS.2017.7946376.

[4] D. Kang, J. Oh, J. Choi, Y. Yi, and S. Ha, "Scheduling of deep learning applications onto heterogeneous processors in an embedded device," *IEEE Access*, vol. 8, pp. 43980–43991, 2020, doi: 10.1109/ACCESS.2020.2977496.

[5] H. B. Hu, X. Li, and J. Liang, "Research and implementation of heterogeneous distributed database system integration," *Appl. Res. Comput.*, vol. 10, pp. 67–70, May 2002.

[6] S. Moulik, R. Devaraj, and A. Sarkar, "Hetero-Sched: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun., IEEE 16th Int. Conf. Smart City, IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Jun. 2018, pp. 659–666.

[7] S. Moulik, Z. Das, R. Devaraj, and S. Chakraborty, "SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems," *J. Syst. Archit.*, vol. 114, Mar. 2021, Art. no. 101953.

[8] R. Devaraj, "A solution to drawbacks in capturing execution requirements on heterogeneous platforms," *J. Supercomput.*, vol. 76, no. 9, pp. 6901–6916, Sep. 2020.

[9] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[10] J. M. Jaffe, "Bounds on the scheduling of typed task systems," *SIAM J. Comput.*, vol. 9, no. 3, pp. 541–551, Aug. 1980.

[11] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, and W. Liu, "Response time bounds for typed DAG parallel tasks on heterogeneous multi-cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2567–2581, Nov. 2019.

[12] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 211–221, doi: 10.1109/ECRTS.2015.26.

[13] M. A. Serrano and E. Quiñones, "Response-time analysis of DAG tasks supporting heterogeneous computing," in *Proc. 55th Annu. Design Autom. Conf.*, Jun. 2018, pp. 1–6.

[14] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Dobel, and H. Hartig, "Response-time analysis of parallel fork-join workloads with real-time constraints," in *Proc. 25th Euromicro Conf. Real-Time Syst.*, Jul. 2013, pp. 215–224.

[15] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-time analysis of synchronous parallel tasks in multiprocessor systems," in *Proc. 22nd Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2014, pp. 3–12.

[16] K. Yang, M. Yang, and J. H. Anderson, "Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 349–358.

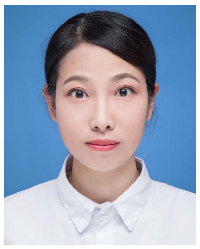
[17] S. Chang, X. Zhao, Z. Liu, and Q. Deng, "Real-time scheduling and analysis of parallel tasks on heterogeneous multi-cores," *J. Syst. Archit.*, vol. 105, May 2020, Art. no. 101704.

[18] S. Nogd, G. Nelissen, M. Nasri, and B. B. Brandenburg, "Response-time analysis for non-preemptive global scheduling with FIFO spin locks," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2020, pp. 115–127, doi: 10.1109/RTSS49844.2020.00021.

[19] Q. He, M. Lv, and N. Guan, "Response time bounds for DAG tasks with arbitrary intratask priority assignment," in *Proc. 33rd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2021, pp. 1–10.
[20] J. Sun, N. Guan, F. Li, H. Gao, C. Shi, and W. Yi, "Real-time scheduling and analysis of OpenMP DAG tasks supporting nested parallelism," *IEEE Trans. Comput.*, vol. 69, no. 9, pp. 1335–1348, Sep. 2020.
[21] J. Chen, Y. He, Y. Zhang, P. Han, and C. Du, "Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems," *J. Syst. Archit.*, vol. 129, Aug. 2022, Art. no. 102598.
[22] J. Chen, P. Han, Y. Liu, and X. Du, "Scheduling independent tasks in cloud environment based on modified differential evolution," *Concurrency Comput., Pract. Exper.*, vol. 21, Mar. 2021, Art. no. e6256.
[23] J. Chen, C. Du, P. Han, and X. Du, "Work-in-progress: Non-preemptive scheduling of periodic tasks with data dependency upon heterogeneous multiprocessor platforms," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 540–543.
[24] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.
[25] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Learning-based response time analysis in real-time embedded systems: A simulation-based approach," in *Proc. 1st Int. Workshop Softw. Qualities Dependencies*, May 2018, pp. 21–24.
[26] J. Sun, N. Guan, Z. Guo, Y. Xue, J. He, and G. Tan, "Calculating worst-case response time bounds for openMP programs with loop structures," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2021, pp. 123–135, doi: 10.1109/RTSS52674.2021.00022.
[27] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2283–2295, Oct. 2019, doi: 10.1109/TPDS.2019.2910525.
[28] S. K. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *J. Syst. Archit.*, vol. 105, May 2020, Art. no. 101706.
[29] S. K. Roy, R. Devaraj, and A. Sarkar, "Contention cognizant scheduling of task graphs on shared bus-based heterogeneous platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 2, pp. 281–293, Feb. 2022.
[30] R. Devaraj and A. Sarkar, "Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.
[31] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proc. 3rd Int. ICST Conf. Simulation Tools Techn.*, 2010, pp. 1–10.
[32] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005.

CHEN SHUSHAN was born in Xinyang, Henan, China, in 1998. He received the master's degree in computer software and theory from the School of Computer Science and Engineering, Xi'an Technological University. His research interests include embedded real-time systems and multicore computing.

XIAO FENG was born in Jiaozuo, Henan, China, in 1976. He received the B.S. and M.S. degrees in computer science from Xi'an Technological University, China, in 2000 and 2003, respectively, and the Ph.D. degree from Northwest University, in 2012. From 2003 to 2006, he was a Teacher with Xi'an Technological University. Since 2006, he has been an Assistant Professor with the Computer Science and Engineering College, Xi'an Technological University, where he has been a Professor with the School of Computer Science and Engineering, since 2016. He is the author of three books and more than 70 articles. His research interests include intelligent information processing, pattern recognition, and computer vision.



HUANG SHUJUAN was born in Weinan, Shanxi, China, in 1975. She received the B.S. degree from the Computer Software Department, North Western University, Xi'an, Shanxi, in 1996, and the M.S. and Ph.D. degrees from the Computer College, Northwestern Polytechnical University, Xi'an, in 2005 and 2014, respectively. From 1996 to 2002, she was a Research Assistant at the Tenth Research Institute of Telecommunication Science and Technology. From 2005 to 2013, she was a Lecturer at the Software and Microelectronic College, North Western Polytechnical University. Since 2014, she has been an Assistant Professor with the School of Computer Science and Engineering, Xi'an Technological University, Shanxi. Her main research interests include the internet of things, multicore computing, and embedded systems.

ZHANG WENJUAN was born in 1980. She received the Ph.D. degree in applied mathematics from Xidian University. From November 2018 to November 2019, she was a Visiting Scholar at the Department of Mathematics, University of Florida, USA. She is currently an Associate Professor and a master's Supervisor with Xi'an Technological University, Xi'an, China. Her research interests include image processing, computer vision, and machine learning.

HAN XINGXING was born in Shangluo, Shanxi, China, in 1996. She received the master's degree in electronic information from the School of Computer Science and Engineering, Xi'an Technological University. Her research interests include embedded real-time systems and multicore computing.

LI TIANSEN was born in Wuwei, Gansu, China, in 1996. He received the master's degree in software engineering from the School of Computer Science and Engineering, Xi'an Technological University. His research interests include embedded real-time systems and mixed-criticality.

• • •