Distributed Integrated Modular Avionics Resource Allocation and Scheduling Algorithm Supporting Task Migration

Qing Zhou[1(✉)], Kui Li[1], Guoquan Zhang[1], and Liang Liu[2]

[1] National Key Laboratory of Science and Technology on Avionics Integration, China Aeronautical Radio Electronics Research Institute, Shanghai 200233, China
zhouqingavic@163.com
[2] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 29 Jiangjun Avenue, Nanjing 210016, China

Abstract. At present, the avionics system tends to be modularized and integrated, and the distributed integrated modular avionics system (DIMA) is proposed as the development direction of the next generation avionics system. In order to support the operation of complex tasks, DIMA needs to have an effective resource allocation and scheduling algorithm for task migration and reorganization to achieve reconstruction. However, many current resource allocation and scheduling algorithms, used in traditional avionics systems, are not available for DIMA. In view of the above problems, the paper analyzes the characteristics of the DIMA avionics system architecture model and builds abstract models of the computing resources, computing platforms and tasks. Based on the established model, an efficient task scheduling algorithm, resource allocation algorithm and task migration algorithm for DIMA avionics architecture are designed. And we do simulation experiments to establish the model, and compare the designed EWSA algorithm with the mainstream algorithm JIT-C. The results show better performance in terms of workflow average completion time, successful scheduling completion rate and optimization rate. In addition, considering the failure in the process of executing the mission, we proposed a mission migration and reorganization algorithm WMA and set different time and number of fault resources of the aircraft in the simulation experiments to evaluate the performance of WMA algorithm.

Keywords: Distributed integrated modular avionics systems · Resource allocation · Scheduling algorithm · Task migration

1 Introduction

The avionics system is the critical system to the mission for aircrafts. It covers various electronic systems such as communications, radar, surveillance, and flight control. If the engine is the heart of the aircraft, then the avionics system is the brain [1]. It can be said that there is no advanced aircraft without advanced avionics systems [2]. Recently,

Q. Li et al. (Eds.): BROADNETS 2019, LNICST 303, pp. 176–197, 2019.
https://doi.org/10.1007/978-3-030-36442-7_12

the avionics system tends to be modular and integrated, subsystems such as flight management, navigation, display control, radar, photoelectric detection, fire control, and mission control, are all implemented with universal function module [3]. Distributed Integrated Modular Avionics (DIMA) system has been proposed as the direction of the development of next-generation avionics system [4].

The DIMA is essentially a parallel distributed computer system that integrates heterogeneous hardware resources, which uses switched network to interconnect hardware facilities of different physical areas of the aircraft and applies application software system to share underlying hardware devices, to realize highly generalized hardware devices and lower degree of software and hardware binding coupling [5].

However, compared to other distributed system architectures, DIMA architecture has many special features owing to its requirements of strong real time, high security and reliability [6]. First of all, the energy consumption of the computing platform service is very small compared to the braking system, thus the energy consumption caused by the algorithm is basically negligible [7]. Secondly, the aeronautical system has a very high real-time demand for task scheduling, thus the cost factor can be ignored while sharing internal resources of the aircraft [8]. In addition, in order to ensure the high safety and reliability of the avionics system, it is necessary to consider the emergency measures of the aircraft in the case of computational resource failure [9]. For aircraft clusters, we need to consider the differences between different types of aircrafts for the different sensor information they carried. Moreover, location information also needs to be considered when assigning tasks in the process of dynamic flight of aircrafts [10].

Therefore, in the face of complex DIMA avionics architecture, how to redistribute integrated heterogeneous computing resources, how to efficiently implement task scheduling on shared resources, and how to improve the reliability and security of application systems under DIMA system are the urgent problems to be solved in the current aviation research. However, with the strong real-time, high security and reliability requirements of DIMA avionics system, related scheduling strategies of traditional aeronautical systems applying on DIMA show poor performance [11].

In view of the above problems, the paper analyzes the characteristics of the DIMA avionics system architecture model and abstracts the computing resources, computing platforms and tasks. Based on the established model, an efficient task scheduling algorithm, resource allocation algorithm and task migration algorithm for DIMA avionics architecture are designed.

2 Related Works

At present, although few scholars have studied task scheduling and migration algorithms for DIMA avionics systems, many excellent task scheduling algorithms have been proposed in other distributed fields such as cloud computing.

Gupta et al. [12] proposed an online-aware job scheduling algorithm named MPA2OJS for multi-platform applications. To improve job execution time and optimize throughput, MPA2OJS uses dynamic heuristic job scheduling mapping schemes and efficiently schedules new high-performance cloud computing (HPC) workflows and load fluctuations for heterogeneous computing resource platforms based on internal application characteristics, job requirements, platform capabilities and dynamic requirements. The algorithm takes the availability of platform resources, the applicability and sensitivity of jobs on the platform into consideration, which balances the load of multiple computing platforms and distributes job streams to the most profitable computing platform efficiently.

Dong et al. [13] proposed a task scheduling algorithm with the most efficient priority named MESF to reduce energy consumption by limiting the number of active servers and the time of response. It takes advantage of the integer programming optimization solution to trade off active server and the time of response. Experimental simulation results show that MESF can save about 70% of energy consumption compared with the random task scheduling scheme.

Panigrahy et al. [14] proposed a geometric heuristic algorithm by sorting VM request queues of virtual machines. Li et al. [15] proposed a method based on multi-dimensional knapsack problem to solve the virtual machine VM placement. By using the virtual machine VM placement schedule, the total working time requested by the virtual machine on the same physical machine was reduced.

Khanna et al. [16] proposed a dynamic host management algorithm, which is activated when the physical machine becomes underloaded or overloaded. At the same time, the algorithm reduces SLA violations, minimizes migration costs and optimizes the number of physical servers. Beloglazov and Buyya et al. [17] analyzed historical data of resource utilization of virtual machines and proposed dynamic virtual machine integration to achieve the effect of power saving.

Taheri and Zamanifar et al. [18] introduced a two-stage virtual machine integration mechanism to deal with incomplete VM migration. Perplex VMs are virtual machines that should be integrated but have no suitable host placement. As a result, the system terminates the migration and VMs is replaced in its previous location. This problem will lead to a waste of CPU capacity and power consumption, increasing network overhead. According to the proposed framework, VMs are migrated from the overused host to other hosts in the first phase, and VMs from the low-load host are sent to other hosts in the second phase.

Sahni and Vidyarthi [19] etc. proposed a workflow scheduling algorithm JIT-C with deadline constraint. JIT-C algorithm is the core technology in the workflow task until the execution and ready to make a proper scheduling decisions, in order to make proper scheduling decisions/supply, this algorithm considering the performance of the virtual machine VM cloud computing platform change, loop controller monitors the task execution progress monitoring, and according to the latest information resources allocation/scheduling decisions. JIT-C algorithm scheduling workflow tasks has good

performance, but because of the avionics system's strong real-time, high security and reliability, JIT-C algorithm can not meet the requirements of avionics system.

JIT-C has some shortcomings used in DIMA system platform. We model the computing resources, the computing platform and tasks by analyzing the characteristics of DIMA avionics system. Based on the established model, an efficient workflow-based task scheduling algorithm EWSA is proposed and compared with JIT-C algorithm, which focused on DIMA avionics system architecture. Through the simulation experiments, we evaluate these algorithms in terms of the average completion time and the optimization rate. In addition, considering the failure in the process of executing the mission, we proposed a mission migration and reorganization algorithm WMA and set different time and number of fault resources of the aircraft in the simulation experiment to evaluate the performance of WMA algorithm.

3 DIMA Avionics System Resource and Task Model

Based on the requirements of strong real-time performance, high security and reliability of DIMA avionics system, we modeled the computing resources, and proposed the computing platform and workflow task model for the computing resources model.

3.1 Computing Resource Model

A computing resource has a variety of parameter information such as processor, memory, bandwidth, etc., and each processor has one or more cores, each core has a fixed computing capacity. Each computing resource can run one or more subsystem tasks that share the underlying hardware facilities such as memory, processors, and so on. In order to ensure that the tasks of each subsystem are not interfered with each other when sharing resources, the concept of partition is proposed in the standard specification of ARINC653. By dividing different tasks into different partition systems, the task systems running on the same computing resources are not interfered with each other.

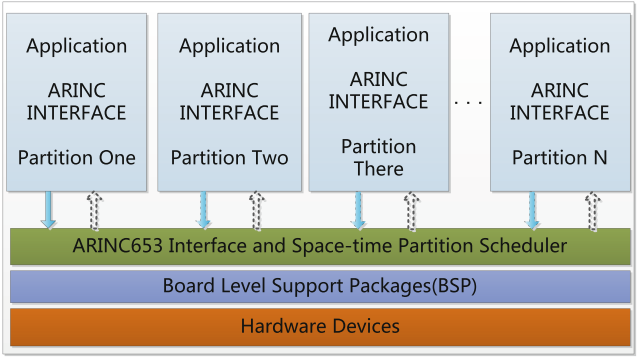

Fig. 1. ARINC653 partition software diagram

Figure 1 shows ARINC653 partition software, the core concept is to propose time and space partition isolation [20]. On the same computing resource, time-division or space-division scheduling schemes are implemented between partitions, and partitions are isolated from each other, which improve fault tolerance and security of the system. In the partition, multiple task processes share the hardware resource information obtained by the partition system, and the scheduling between task processes can be specified by the partition system designer, such as FCFS, MFS, etc. Therefore, in order to comply with the ARINC653 standard specification, the task process application is run by creating a VmPartition during the abstract modeling of computing resources of DIMA avionics system.

Figure 2 shows the model diagram of computing resources. It is assumed that an aircraft has multiple computing resources, and different computing resources have different parameters such as the number of processors, processing speed and memory configuration, etc., and the external service is composed of virtualized resources, including computing services and storage services. All computing resources inside the aircraft are interconnected through a strong real-time communication network to share sensor interfaces. Assuming that all computing and storage services are provided by the same aircraft, the average bandwidth between computing services is roughly equal. In addition, the storage service is implemented by the unified allocation of the local storage service of the aircraft, and the computing service is provided in the form of different types of VmPartition. Virtual partition has different CPU types, memory RAM and other configuration information resources, and virtual partition is dynamically submitted by the user to create and destroy. Different process task types are running between virtual partitions. If two tasks t_{i} and t_{j} are running on different virtual partitions and there is data transmission association between tasks, the calculation formula of data transmission time $TT\left(e_{ij}\right)$ is as follows: $\frac{d_{t_i}}{\beta}$, where d_{t_i} is the data file size output from task t_{i} to task t_{j}, and β is the average bandwidth of data transmission within the aircraft. If two tasks t_{i} and t_{j} are scheduled to run on the same virtual partition t_{j}, the data transfer time between them is zero. In the research work of this paper, the scheduling scheme between virtual partitions on the same computing resource is Time Shared or Space Shared strategy, and the intra-partition tasks uses the first-come-first-served (FCFS) strategy while scheduling between processes.

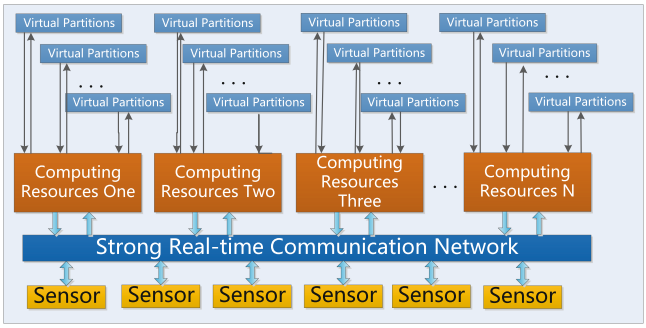

Fig. 2. The model diagram of calculation resource

3.2 Computing Platform Model

The computing platform model in this paper is similar to the one used in literature [20], as shown in Fig. 3. The resources of the aircraft provide services externally with virtualized service resources, which mainly involve three stages in planning workflow task execution. The first stage completes resource provisioning, which involves identifying and mapping the number of computational resources required to perform this task and the configuration of the VmPartition. The second stage dynamically creates the VmPartition onto the appropriate physical computing resources and generates an appropriate schedule for task execution in that partition. The third stage is a process that exists from the start of operation of the aircraft to the end of the aircraft, that is, to scan whether the physical computing resources of the aircraft fail. In case of failure, the failure resource processing module is called to implement the emergency plan to ensure the execution of subsequent tasks.

Workflow tasks submitted to the computing platform contain associated high-quality service (QOS) requirements, such as deadline constraints and resource specification requests. The deadline limit refers to the latest completion time for executing the workflow, and the resource specification required to perform the task refers to the computing resource requirements (such as memory, computing power, I/O, etc.). Based on these input requests, the Workflow Management System (WMS) automatically identifies the required resources and maps the tasks to the corresponding VmPartition.

WMS is mainly composed of three modules: resource allocation mapping module, workflow scheduling module and execution management module. Resource allocation mapping module consists of two sub-modules: resource capability assessment module and resource mapping management module. Resource capacity assessment module analyzes workflow structure to determine the number of resource requests, and resource mapping management module maps the corresponding resource requests to the corresponding VmPartition. Workflow scheduling management module and execution management module work together to dynamically create VmPartition on aircraft physical computing resources. Workflow scheduling management module mainly involves the actual scheduling of any workflow node to the VmPartition execution. The execution management module is used to track the execution status of the task node to record the Real Start Time (RST) and Real Finish Time (RFT) of the task node and update the status of resource information. This computing model is task-driven and the size of the fetched resources may vary at runtime.

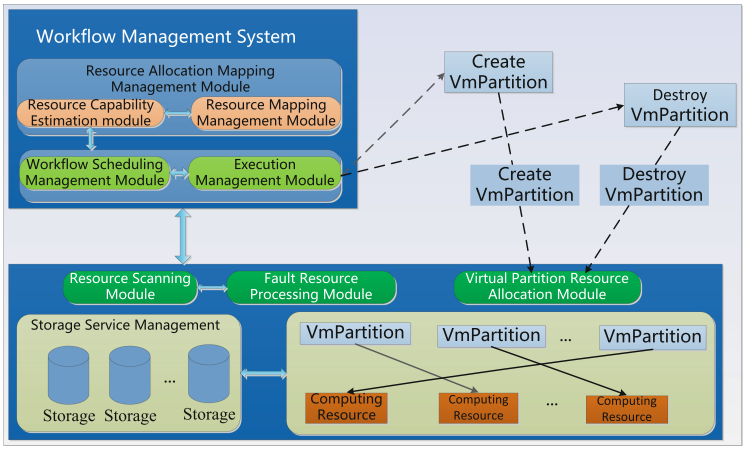

Fig. 3. Computing platform model diagram

The aircraft is mainly composed of resource scanning module, fault resource processing module and virtual partition resource allocation module to manage various computing resource information of the aircraft. The resource scanning module works in cooperation with the fault resource processing module. When the fault resource is scanned, the fault resource processing module executes the corresponding emergency plan. The virtual partition resource allocation module actually creates/destroys VmPartition on the physical computing resources.

3.3 Task Model

In the field of real aviation, the combat mission of aircraft is realized step by step. As shown in Fig. 4, when the aircraft strikes the target, it must first discover the target, identify and track the target, and then carry out anti-reconnaissance to prevent its exposure in the process of tracking. When a series of pre-task execution is completed, the target is hit. In the execution of a series of predecessor tasks, there are data transfer association requests between the subtasks, and subsequent tasks cannot begin execution until all prior tasks have been completed. In addition, task nodes that are simultaneously executable can be executed concurrently.

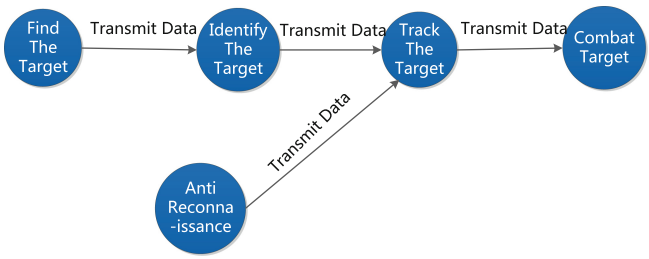

Fig. 4. Operational task execution diagram of the aircraft.

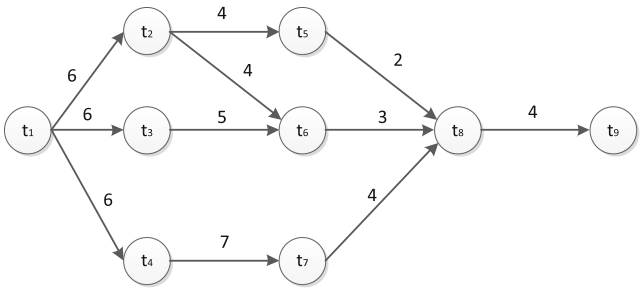

Fig. 5. Workflow task model diagram

According to the description above, we abstract similar aviation missions into a workflow task model, as shown in Fig. 5, a workflow application W = (T, E) be modeled as a directed acyclic graph (DAG), where $T = \{t_1, t_2, \ldots, t_n\}$ means task node set, E is a set of tasks with data transmission between or control set of dependencies to the edge. Dependency e_{ij} is a set of priority constraints of task $(t_i,\ t_j)$, where $t_i,\ t_j \in T$ and $t_i \neq t_j$, indicate that task t_j (subtask) cannot start execution before the end of task t_i (parent task) execution, because there are relevant data transfer dependencies between task t_i and task t_j. Subtask nodes are not able to start executing until all parent task nodes have finished executing and all dependencies (including data transfer and control) have been preprocessed. The workflow deadline D is defined as the latest execution time limit for executing the workflow. In this paper, each task node is modeled as a node with a fixed number of instructions, and each node has a corresponding output data file size. As shown in the figure, each directed edge represents the time required to transfer data files between tasks.

4 Efficient Task Scheduling Algorithm Under DIMA Avionics System

Based on the computing resources, computing platform and task model of DIMA avionics system in Sect. 4, this section introduces an efficient task scheduling algorithm. Firstly, several variables used in the algorithm and their meanings are introduced, as shown in Table 1.

Table 1. Definition of basic mathematical symbols

Symbol	Meaning
D	Workflow W deadline
$T(t_i, VmPartition_v)$	Task t_i execution time on virtual partition type $VmPartition_v$
$TT(e_{ik})$	Data transfer time between task t_i and task t_k
$MET(t_i)$	The minimum execution time for task t_i
$\{t_{entry}\}$	A task without a parent node
$\{t_{exit}\}$	A task without child nodes
$EST(t_i)$	The earliest execution time for task t_i
$EFT(t_i)$	The earliest time task t_i ends
$RST(t_i)$	Task t_i actually starts execution time
$RFT(t_i)$	Task t_i actual end time
$EXST(t_i)$	Task t_i is expected to start time
$EXFT(t_i)$	Task t_i expected end time
$LST(t_i)$	Task t_i starts execution at the latest
$LFT(t_i)$	Task t_i end at the latest
MET_W	Workflow W minimum execution time
RATE	Workflow W completion time optimization rate

The operational equations of the above basic mathematical symbols are shown as follows, some of which are similar to literature [20]:

(1) Firstly, the proposed algorithm should satisfy the real-time requirement of avionics system. It is necessary to find a scheduling scheme for workflow W and optimize the total completion time TCT (Total Completed Time) of workflow W under the limitation of deadline D. The optimization objective is shown in Eq. (1).

$$Optimize\, TCT = \max_{t_i \in W} \{(RFT(t_i))\}, TCT \leq D \tag{1}$$

Where TCT is the total completion time of workflow W, t_i is the task node in W, and RFT (t_i is the actual completion time of task node t_i.

(2) $MET(t_i)$: the minimum execution time of task t_i, assuming that task t_i can be executed on $VmPartition_{type}$ and has a minimum execution time $ET(t_i, VmPartition_v)$ on a certain VmPartition type. The calculation formula is shown in (2):

$$MET(t_i) = \min_{VmPartition_v \in VmPartition_{type}} \{ET(t_i, VmPartition_v)\} \tag{2}$$

(3) $TT(e_{ij})$: the time for transferring data between task t_i and task t_j, assuming that task t_i and task t_j are assigned to execute on different VmPartition. The average bandwidth between the VmPartition of the aircraft is β, the data size from task t_i

to task t_j is d_{t_i}, and the calculation formula of data transmission time is shown in (3):

$$TT(e_{ij}) = \frac{d_{t_i}}{\beta} \tag{3}$$

(4) $EST(t_i)$: the earliest start time of task t_i, when the tasks of the predecessor nodes of task t_i are all executed with the minimum execution time, and after the relevant data dependence transferring to t_i, t_i can start execution. The calculation formula is shown in (4):

$$\begin{cases} EST(t_{entry}) = 0 \\ EST(t_i) = \underset{t_p \in t_{i's^{parent}}}{max} \{EST(t_p) + MET((t_p) + TT(e_{pi})\} \end{cases} \tag{4}$$

(5) $EFT(t_i)$: the earliest end time of task t_i, the calculation formula is shown in (5):

$$EFT(t_i) = EST(t_i) + MET(t_i) \tag{5}$$

(6) $EXFT(t_i)$: the expected end time of task t_i, assuming that the number of scheduled tasks at the same level is N, the minimum execution time of all task nodes are at the same level. The calculation formula is shown in (6):

$$\mathrm{E}XFT(t_i) = EXST(t_i) + \frac{\sum_{j=1}^{N}(MET(t_j))}{N} \tag{6}$$

(7) $EXST(t_i)$: the expected start time of task t_i, which is defined as the expected start time that t_i can start execution after all previous task nodes of task t_i have been scheduled and executed. The calculation formula is shown in (7):

$$\begin{cases} EXST(t_{entry}) = 0 \\ EXST(t_i) = \underset{t_p \in t_{i's^{parent}}}{\max} \{EXFT(t_p) + TT(e_{pi})\} \end{cases} \tag{7}$$

(8) $LFT(t_i)$: the latest completion time of task t_i with the limit of workflow deadline D. The calculation formula is shown in (8):

$$\begin{cases} LFT(t_{exit}) = D \\ LFT(t_i) = \underset{t_c \in t_{i's^{children}}}{min} \{LFT(t_c) - MET(t_c) - TT(e_{ic})\} \end{cases} \tag{8}$$

(9) $LST(t_i)$: the latest start time of task t_i with the limit of workflow deadline D. The calculation formula is shown in (9):

$$LST(t_i) = LFT(t_i) - MET(t_i) \tag{9}$$

(10) *MET_W*: The minimum execution time of workflow W, which is defined as the total time required when all task nodes on the critical path (the longest execution path) complete execution on the fastest virtual machine. The calculation formula is shown in (10):

$$MET\_W = \max_{t_i \in W} \{EFT(t_i)\} \tag{10}$$

(11) *RATE:* the optimization rate of completion time with the constraint of deadline D. The calculation formula is shown in Eq. (11):

$$RATE = \frac{TET - D}{D} \tag{11}$$

4.1 Preprocessing Algorithm

In order to save the time of data transfer between tasks, the serial task nodes in workflow W need to be combined into one task node in advance. The pre-processing shown in Algorithm 1, is implemented by tksqueue, which firstly queues the entry task, queues all its sub-task nodes when it exits the queue, merges the existing serial task nodes, and changes the direction of the merged task node and data transmission time.

Algorithm 1. *Pre-processing (W)*
Input: A workflow of N tasks DAG W(T，E)
1. Begin
2. *tksqueue* ← $\{t_{entry}\}$
3. While *tksqueue* not null then
4. t_p ← *tksqueue(front)* // Retrieves the queue header element
5. $S_c \leftarrow \{t_c | t_c \in W \;\&\&\; t_c$is a subtask node of $t_p\}$
6. If $|S_c| == 1 \;\&\&\; t_c$ has only one parent node t_p
7. Combine task t_p and t_c into task t_{p+c} and replace $_p$ nodes
8. Set the parent of the $t_c{}'s$ node to t_{p+c}
9. Change the t_{p+c} transfer time to t_c to $t_c{}'s$
10. Update $MET(t_{p+c})$
11. Add t_{p+c} to *tksqueue*
12. Else
13. Add child node task $t_p{}'s$ to *tksqueue*
14. End if
15. End While
16. End

4.2 Resource Allocation Mapping Algorithm

The resource allocation mapping algorithm is used to allocate VmPartition for task mapping at the same level in workflow W. In addition, the corresponding task node binding is assigned to VmPartition.

Specific resource allocation algorithm OptimalVmPartitionMapping, shown in Algorithm 2, firstly calculates the average expected with hierarchical task execution time and start time by using Eqs. (5) and (6) with AverageExcutionTime method (line 2). The allocateResourceType method (line 8) is then used to allocate appropriate computing resources to each task using the expected end time and expected start time difference of the task. When mapping VmPartition resource configuration, it may occur that the expected end time of the task node is greater than the expected start time, that is, EXFT $\geq$ EXST. In this case, it is unreasonable to continue scheduling the task at the current expected end time. Therefore, we need to appropriately extend the expected end time of the task node to appropriately shrink the VmPartition resource configuration of the task node.

Algorithm 2. *OptimalVmPartitionMapping* (*taskList*)

1. Begin
2. *averageExcutionTime* ←*caculateAverageExcutionTime*(*taskList*)
3. For each $t_i \in taskList$
4. *taskVmPartitionMap* = ∅
5. Sets the expected end time$EXFT(t_i) \leftarrow averageExcutionTime$ - TT(t_i)
6. If $EXFT(t_i) \leq LFT(t_i)$ then
7. $tempTime \leftarrow EXFT(t_i)$ - $EXST(t_i)$
8. *taskVmPartitionMap* ← *allocateResourceType*(*tempTime*)
9. Else
10. *taskVmPartitionMap* ← Assigns a virtual partition configuration 11. of a standard configuration type
11. Bind the task Id of taskVmPartitionMap
12. Set the creation time of taskVmPartitionMap to $EXFT(t_i)$
13. Add VmPartitionMa to VmPartitionList*p*
14. End For
15. End

4.3 Ewsa Algorithm

When workflow W is in the existing vehicle scheduling on computing resources platform, first initializes the workflow W with the Initialize method, then respectively using Eqs. (2), (3) and (4) to calculate the minimum execution time MET in the

workflow W task node, the data transmission time TT and the earliest start time EST, and identify critical path task node in workflow W. Equation (10) is used to calculate the minimum execution time in the workflow W, and judge the workflow scheduling can be. The Algorithm is shown in Algorithm 3.

Algorithm 3.*Efficient Workflow Schedule Algorithm* (EWSA)
Input:
Workflow DAG W(T, E) composed of N task nodes
The deadline D defined by workflow W
1. Begin
2. *Initialize*(W)
3. lculate the minimum execution time of workflow W MET_W
4. utilization (11)
5. If D ≥ MET_W then
6. Invoke the preprocessor pre-processing (W) to preprocess the workflow
8. Calculate the workflow task nodes MET, LFT, LST using equations (2), (8), and (9)
9. *analysisLevel*(W)
10. $level \leftarrow max\{\text{level(W)}\}$
11. for($i \leftarrow 1; i \leq level; i \leftarrow i+1$)
12. $taskList \leftarrow \{t_i | t_i \in W \;\&\&\; leval(t_i) == level\}$
13. *OptimaVmPartitionMapping*(*taskList*)
14. end for
15. {tentry}←All entry nodes in workflow W
16. *SheduleAviationCloudlet*($\{t_{entry}\}$)
17. While $t_i \in W \;\&\&\; t_i$ unscheduled execution then
18. t_i is sent to the scheduling manager to perform the scheduling
19. Update task $t_i's$ time start execution time RST and actual end time RFT *to_be_scheduled*← $\{t_i \in \text{W} \mid \forall t_p \in t_i' sparent \&\& t_p$ 24. completed }
20. *SheduleAviationCloudlet* (*to_be_scheduled*)
21. end while
22. Else
23. It is recommended to modify the
24. deadline D && D ≥MET_W
25. end If
26. End

When it is judged that the workflow W can be scheduled within the deadline D, the Pre-processing algorithm is used to pre-process the workflow. Equations (2), (8) and (9) were respectively used to update the task nodes MET, LFT and LST in workflow

W. The scheduling hierarchy of each task node in workflow task W and corresponding task nodes at the same level are analyzed by *analysis Level* method. The resource allocation mapping algorithm OptimaVmPartitionMappin maps one type of *VmPartition* separately to generate the corresponding expected scheduling plan. The *SheduleAviationCloudlet* method (lines 14–20) is in the task scheduling management module, which performs the actual task scheduling and works with the execution manager to dynamically create the corresponding task's VmPartition on the aircraft's computing resources and execute the task scheduling on VmPartition. During this process, the execution manager monitors the execution status of the scheduling task, and records the actual start time RST and the actual end time RFT of the task, which are used by the scheduling manager to decide the scheduling plan for the next task. When the execution manager finds that the task has finished executing, it works with the scheduling manager to destroy the task's VmPartition and release system resources.

When the task scheduling management module dynamically creates the *VmPartition* where the task resides, there may be insufficient computing resources available for the aircraft to meet VmPartition requirements. At this point, we need to dynamically adjust the configuration type of VmPartition based on the available computing resources of the current aircraft. In the process of adjusting the virtual partition configuration VmPartition, the following conditions must be met:

(1) Task t_i executing on VmPartition is expected to finish no later than task t_i, that is, $\mathrm{EXFT}(t_i) \leq \mathrm{LFT}(t_i)$.
(2) Task t_i binded on *VmPartition* executes on virtual partitions, there is no such node t_c subtasks, $t_c \in t_i'$ s children, making the expected end time of t_c is greater than the start time at the latest.

4.4 Task Migration Wma Algorithm

The fault resource processing module works together with the resource scanning module. When the resource scanning module finds that there is a fault computing resource, *VmPartition* on the fault resource is sent to the fault resource processing module. The fault resource processing module calls the migration algorithm to realize task migration and reorganization, as shown in **Algorithm 4.**

Algorithm 4. *WorkflowMigrationAlgorithm* (*VmPartitionList*)

1. Begin
2. If *VmPartitionList* not null Then
3. $R \leftarrow \{R_k | R_k$ is the computing resource available in the current aircraft$\}$
5. For each $\text{VmPartition}_i \in VmPartitioonList$
6. For each $R_k \in R$
7. If The remaining compute resources available in R_i satisfy the VmPartition_irequest Then
8. Move VmPartition_i to compute resource R_i
9. Restart the task node in Restart the task node in 11.$\text{VmPartition}_i$and Break
10. End If
11. End For
12. If VmPartition_i can't find moving computing 15.resources && (The task on VmPartition_i belongs to the 16.critical path task || VmPartition_i has special priority tasks) Then
13. The maximum number of remaining computing 19.resources available in $R_j \leftarrow R$
14 *selectVmPartition* ← 21. *selectVmPartitionMigrateChoice*(VmPartition_i, R_j, choice)
15. Suspend selectVmPartition and join the wait queue 23.*VmPartitionWaitingList*
16. Free the system resources occupied by select VmPartition
17. Moving VmPartition_i to compute resource R_j and restart 26.the task
18. Else
19. Add VmPartition_i to the wait queue 29.*VmPartitionWaitingList*
20. End If
21. End For
22. End

This algorithm realizes *VmPartition* movement by traversing (lines 5–9) other computing resources available to the aircraft. If it finds the computing resource R_i available to accommodate VmPartition's request, move VmPartition directly to the computing resource R_i and restart the task on *VmPartition*.

If it traverses all available computing resources and find that no computing resources are suitable for VmPartition's resource requirements, we select part of the virtual partition in a preemptive manner to temporarily suspend and join the waiting queue. As shown in 12–16 lines, we select a virtual partitions with selectVmPartitionMigrateChoice method, which realizes two selection strategy as follows:

(1) Select the selectVmPartition in the compute resource R_j that occupies the least computing resources and is applicable.
(2) select the task node in the compute resource R_j with the longest execution end time and is applicable for selectVmPartition.

During processing the task migration, whenever the task node finishes execution and releases the system computing resources, the fault resource processing module will give priority to the tasks pending execution queue to ensure that the suspended tasks can be executed as soon as possible. Although the migration and reorganization of tasks in a preemptive way will lead to the delay of the entire workflow W, it can ensure that the aircraft can still perform the workflow tasks normally in the case of resource failure. Simulation results show that in workflow with relatively simple relational complexity, the probability of delayed task caused by failure of fault resource is obvious. Moreover, delay rates are relatively low or even better in complex workflows.

5 Experimental Simulation and Result Analysis

In the paper, we use the simulation tool CloudSim [11] to build the resource model, task model and computing platform model for the proposed distributed integrated modular avionics DIMA system. the proposed algorithm EWSA is compared with the algorithm JIT-C. Moreover, in order to evaluate the performance of WMA algorithm, we set different time and number of fault resources of the aircraft in the simulation experiment.

5.1 Simulation Experiment Configuration Ewsa Algorithm

In the simulation environment of CloudSim, we assume that the computing resources of the aircraft have both isomorphic and heterogeneous types. The isomorphic computing resources are consistent with all kinds of computing resources processing capacity and memory configuration within the aircraft. For heterogeneous computing resources, the internal computing resources of the aircraft are inconsistent and the processing capabilities are different. The isomorphic computing resource information is shown in Table 2, and the other is shown in Table 3. We assume that there are 10 physical computing resources inside an aircraft, and the average network bandwidth between internal virtual partitions VmPartition is 200 MBps. In addition, as shown in Table 4, this paper sets up virtual partition VmPartition of three different reference configuration types in the simulation process. At the same time, this paper sets the physical computing resources to scan time interval of 200 s.

Table 2. Isomorphic computing resource information table

Quantity	Cores	Capacity (GHz)	RAM (GB)
10	2	4	8

Table 3. Heterogeneous computing resource information table

Serial number	Cores	Capacity (GHz)	RAM (GB)
#1	1	2	4
#2	1	4	8
#3	1	6	16
#4	2	4	8
#5	2	8	16
#6	2	12	32
#7	4	8	16
#8	4	16	32
#9	8	16	32
#10	8	32	64

Table 4. VmPartition baseline configuration type

VmPartition type	Cores	Capacity (GHz)	RAM (GB)	Storage (GB)
Small	1	1	1.7	160
Medium	1	2	3.75	410
Large	2	4	7.5	840

When evaluating algorithm performance, you need to define an expiration date for each workflow W. If the deadline is very loose, there is enough time to schedule the workflow in real time under the constraints of the deadline. Therefore, it is necessary to comprehensively evaluate the performance analysis of the algorithm for all possible periods: emergency, moderate and loose deadlines.

Thus, we use the rules in Eq. (12) to set the deadline [20]:

$$Deadline\ D = (1 + \mu) \times MET\_W \tag{12}$$

Among them, MET_W is the shortest execution time of the workflow, and μ is the deadline date factor, which is defined as follows:

Strict DeadLines: $0 \leq \mu < 1.5$
Moderate DeadLines: $1.5 \leq \mu < 3$
Relaxed DeadLines: $3 \leq \mu < 4.5$

During the experiment, the workflow is generated by random values. The number of task node instructions is among the range of [2.5×10^5, 5×10^5], and the data size between task nodes is among the range of [2×10^2 MB, 5×10^2 MB]. The workflow W is randomly generated, which has one or more ingress nodes and one egress node. The workflow W is incremented with the number of its task nodes among [5, 10, 15, 20, 25, 30, 35], and its workflow relationship is more and more complex. The workflow algorithm with the same task node performs 100 times each time, and we take the average to evaluate the performance of the algorithm.

5.2 Comparison of Experimental Results

In this paper, we compare EWSA with JIT-C in terms of the average completion time of workflows, the rate of successful scheduling completion and the rate of optimization. In addition, by setting different time and number of fault resources of the aircraft, we evaluate the performance of the WMA algorithm in terms of the rate of the task migration delay.

(1) Comparison of average completion time of workflows of different algorithms

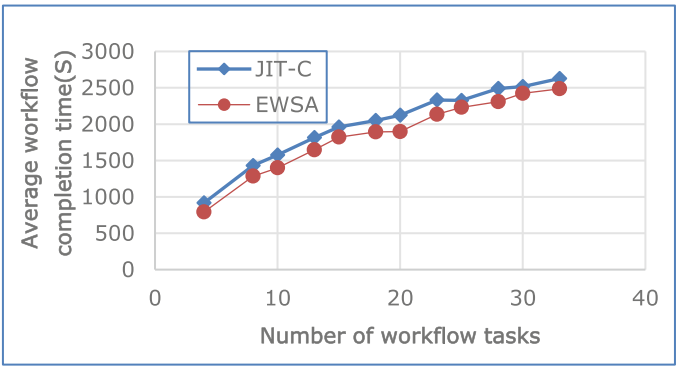

Fig. 6. Workflow completion time

Figure 6 is the comparison of the average completion time of different algorithms under different task numbers. As the number of workflow task nodes increases, the average completion time also increases, and the results show that the EWSA algorithm is always slightly better than the JIT-C algorithm. The reason is that when the EWSA algorithm schedules the workflow task nodes, the task nodes on the critical path acquire more computing resources by evaluating the amount of resources required for each level of the task node. At the same time, tasks on non-critical paths get longer execution time without affecting the execution of the next-level task nodes. The EWSA algorithm optimizes the rational allocation of resources for the entire workflow under the distributed integrated modular DIMA avionics system, thereby reducing the execution time of critical paths.

(2) Comparison of successful scheduling completion rate and optimization rate of different algorithms

We set the deadline limit of workflow tasks under different constraints as STRICT, MODERATE, RELAXED, to evaluate the performance of the algorithm. As shown in Table 5, JIT-C algorithm meets real-time requirements of workflow tasks under STRICT constraints, and the successful scheduling rate is about 75%, while EWSA

algorithm is as high as 90%. This is because EWSA algorithm dynamically adjusts the resource requests of workflow task nodes based on the number of resources available on the current platform, and macroscopically enables workflow task nodes to meet the requirements every time they request resources. Both JIT-C algorithm and EWSA algorithm can achieve 100% under MODERATE and RELAXED deadline constraint, caused by the extended deadline limit that leads workflow tasks to having more relaxed time to implement scheduling on limited resources without delay.

Table 5. Completion rate and optimization rate of workflow successful scheduling

Tasks deadline limit			Workflow					
			5	10	15	20	25	30
STRICT	JIT-C	Success rate	78%	76%	74%	73%	74%	75%
		Optimize rate	5%	5%	5%	5%	4%	5%
	EWSA	Success rate	90%	89%	91%	86%	88%	87%
		Optimize rate	17%	14%	15%	15%	12%	12%
MODERATE	JIT-C	Success rate	100%	100%	100%	100%	100%	100%
		Optimize rate	41%	42%	42%	42%	41%	42%
	EWSA	Success rate	100%	100%	100%	100%	100%	100%
		Optimize rate	44%	45%	44%	45%	46%	44%
RELAXED	JIT-C	Success rate	100%	100%	100%	100%	100%	100%
		Optimize rate	70%	70%	70%	69%	67%	68%
	EWSA	Success rate	100%	100%	100%	100%	100%	100%
		Optimize rate	70%	72%	70%	70%	70%	70%

The rate of optimization is calculated according to the deadline defined by the workflow, and the calculation formula is shown in Eq. (11). In order to see the optimization effect of workflow completion time more intuitively, we present the optimization results in the form of bar chart, which is shown in Fig. 7. The optimization rate of EWSA algorithm is higher about 10% than that of JIT-C algorithm under STRICT constraints. Under MODERATE and RELAXED constraints, they are nearly stable and the optimization rate of EWSA algorithm is about 3% higher than that of JIT-C algorithm. As the deadline constraint is more relaxed, JIT-C algorithm can always find the appropriate resources while EWSA algorithm optimization degree is close to saturation. In addition, with the increase of the number of workflow task nodes, the complexity of workflow relationship becomes higher and higher. Many tasks are close to serial execution, and the concurrent execution rate is lower, resulting in the lower degree of optimization.

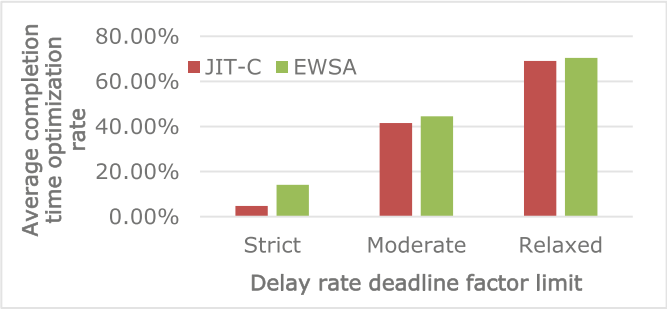

Fig. 7. Completion time optimization rate

(3) Migration task delay rate of wma algorithm

We set different time and numbers of failure resources of the aircraft in our simulation to verify the efficiency of task migration and reorganization, and to evaluate the performance of WMA algorithm. The computational resource failure time is set as 400 s after the simulation is started, and the number of computational resource failures is randomly selected among [1, 8]. Experimental results, in Fig. 8, show that when the number of workflow task nodes is small and their relationship is simple, the rate of workflow delay is high at about −11%. As the workflow task node number increases, the relationship is more and more complex, thus the delay rate is gradually decreasing, the completion time optimized slightly at around 2%. Because the more complex the relationship between task nodes within the workflow, the number of concurrent execution of task nodes within the same hierarchy decreases, making the execution of the entire workflow task nodes close to serial.

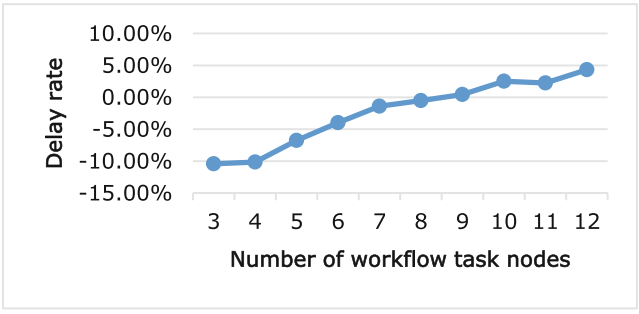

Fig. 8. Migration task delay rate

6 Conclusion and Future Work

Under the current trend of integrated modularization of avionics systems, by analyzing the characteristics of the future distributed avionics system DIMA architecture model, the paper focuses on the task scheduling and resource allocation of DIMA and uses simulation tool CloudSim to model its computing resources, tasks and computing platforms. Based on the established model, an efficient workflow-based task scheduling algorithm EWSA is proposed and compared with JIT-C algorithm, which shows better performance in terms of average workflow completion time and the rate of optimization. In addition, considering the failure in the process of executing the mission, we proposed a mission migration and reorganization algorithm WMA and set different time and number of fault resources of the aircraft in the simulation experiment to evaluate the performance of WMA algorithm.

The related scheduling algorithms and models proposed in the paper are currently applicable to single aircraft platforms. Considering multi-aircrafts cluster is the next research direction.

Acknowledgments. This work was supported in part by the Aeronautical Science Foundation of China under Grant 20165515001.

References

1. Wang, T., Qingfan, G.: Research on distributed integrated modular avionics system architecture design and implementation. In: IEEE AIAA Digital Avionics Systems Conference, pp. 1–53 (2013)
2. Annighofer, B., Thielecke, F.: A systems architecting framework for optimal distributed integrated modular avionics architectures. CEAS Aeronaut. J. **6**(3), 485–496 (2015)
3. Swanson, D.L.: Evolving avionics systems from federated to distributed architectures. In: Proceedings of the 17th DASC Digital Avionics Systems Conference 1998. The AIAA/IEEE/SAE, 1: D26/1-D26/8, vol. 1. IEEE (1998)
4. Han, P., Zhai, Z., Nielsen, B., et al.: A modeling framework for schedulability analysis of distributed avionics systems. arXiv: Software Engineering, pp. 150–168 (2018)
5. Li, X., Xiong, H.: Modeling and analysis of integrated avionics processing systems. In: 2010 IEEE/AIAA 29th Digital Avionics Systems Conference (DASC), pp. 6.E.4-1–6.E.4-8. IEEE (2010)
6. Bao, L., Bois, G., Boland, J., et al.: Model-based method to automate the design of IMA avionics system based on cosimulation. SAE Int. J. Aerosp. **8**(2), 234–242 (2015)
7. Yunsheng, W., Savage, S., Hang, L., et al.: The architecture of airborne datalink system in distributed integrated modular avionics. In: Integrated Communications, Navigation and Surveillance Conference (2016)
8. Robati, T., Gherbi, A., Mullins, J., et al.: A modeling and verification approach to the design of distributed IMA architectures using TTEthernet. Procedia Comput. Sci. **83**, 229–236 (2016)
9. Wang, H., Niu, W.: A review on key technologies of the distributed integrated modular avionics system. Int. J. Wirel. Inf. Netw. **25**(3), 358–369 (2018)

10. Zhou, Q., Xiong, Z., Zhan, Z., et al.: The mapping mechanism between distributed integrated modular avionics and data distribution service. In: Fuzzy Systems and Knowledge Discovery, pp. 2502–2507 (2015)
11. Calheiros, R.N., Ranjan, R., Beloglazov, A., et al.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. **41**(1), 23–50 (2011)
12. Gupta, A., Faraboschi, P., Gioachin, F., et al.: Evaluating and improving the performance and scheduling of HPC applications in cloud. IEEE Trans. Cloud Comput. **4**(3), 307–321 (2016)
13. Dong, Z., Liu, N., Rojas-Cessa, R.: Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers. J. Cloud Comput. **4**(1), 5 (2015)
14. Panigrahy, R., Talwar, K., Uyeda, L., et al.: Heuristics for vector bin packing. research. microsoft.com (2011)
15. Li, K., Zheng, H., Wu, J.: Migration-based virtual machine placement in cloud systems. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), pp. 83–90. IEEE (2013)
16. Khanna, G., Beaty, K., Kar, G.: Application performance management in virtualized server environments. In: 10th IEEE/IFIP, IEEE 2006 Network Operations and Management Symposium, 2006. NOMS 2006, pp. 373–381 (2006)
17. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput.: Pract. Exp. **24**(13), 1397–1420 (2012)
18. Taheri, M.M., Zamanifar, K.: 2-phase optimization method for energy aware scheduling of virtual machines in cloud data centers. In: International Conference for Internet Technology and Secured Transactions, pp. 525–530 (2011)
19. Sahni, J., Vidyarthi, D.: A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans. Cloud Comput. **6**, 2–18 (2015)
20. Wang, Y., Cui, L., Wang, J., et al.: Spatial and temporal partitioning validation for ARINC635-based avionics software. In: International Conference on Electronics and Information Engineering (2015)