IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 4, 2020, accepted December 9, 2020, date of publication December 14, 2020,
date of current version December 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3044724

Improvement to Semi-Partitioned Cyclic Executives for Mixed-Criticality Scheduling on Multiprocessor Platforms

FENGXIANG ZHANG, (Member, IEEE)
School of Computer and Information Science, Southwest University, Chongqing 400715, China
e-mail: zhangfx@swu.edu.cn

This work was supported by the Fundamental Research Funds for the Central Universities under Grant XDJK2019B025.

ABSTRACT This paper focuses on semi-partitioned cyclic executives for mixed-criticality multiprocessor systems in the case of which a job can be split and execute on different processors, this strategy incorporates most of the advantages of the fully partitioned scheduling while further maximising the total processor utilization. We propose algorithms in order to improve the schedulability of such a system. The length of each frame can be reduced to an optimal (minimum) value, which is a necessary schedulability condition for any schedule under this system model (i.e. every processor's capacity is fully utilized within each frame so that the frame cannot be further shortened), whilst all timing requirements of the system are satisfied. By conducting extensive experiments on a very large number of randomly generated job sets, it is demonstrated that our proposed algorithms significantly reduce the required length of each frame; it is possible for almost 100% of the job sets in the experiments to obtain their optimal values with the exception of a very limited number of specific situations. We also prove that when the complete times of some specific jobs are fixed given values, the strategy to shorten the frame presented in this paper is optimal.

INDEX TERMS Scheduling and task partitioning, schedulability analysis, multi-core/single-chip multiprocessors, cyclic executives, mixed-criticality systems, semi-partitioned scheduling, real-time and embedded systems.

I. INTRODUCTION

Mixed criticality scheduling has represented a hot topic and there are many papers (e.g. [4], [7], [9], [15]–[19], [23], [27], [29]) that have been published recently exploring the nature thereof. In a mixed criticality system, different levels of certification are required for different applications on the same execution platform. Each task can be characterised by many different parameters under different levels of criticality. One important mathematical problem is ensuring that the timing constraints of high-criticality applications will be guaranteed under all circumstances without underutilizing the system resources.

A formal definition of different criticality levels can be obtained with reference to the safety standards that define the development and design processes for real-time embedded systems, and there is a variety of domain specific safety standards [20], such as ISO 26262 [2] for road vehicles, or DO 178C [3] for avionic software and DO 254 [1] for avionics hardware.

Cyclic executive [6], [11], [26] is a simple static scheme for implementation. It consists of continuously repeated frames (or minor cycles) and major cycles, each major cycle consists of fixed number of frames, and the tasks' periods are multiples of a frame up to the value of a major cycle. The schedule of jobs within each frame is computed offline and stored in a table. Every allocated job is required to start and finish within each frame.

The issue of how to map the tasks to frames to best support a set of tasks with given periods is beyond the scope of this paper. In this paper, we assume that all jobs of tasks have been allocated to each frame according to a given policy. Our objective is to reduce the required length of each frame (or the length of time required to complete all jobs in each frame) towards an optimal (minimum) value in a mixed-criticality system. There are two major benefits obtained by reducing the required length of frames: 1) more task sets can be scheduled (i.e. all timing requirements are satisfied) after the tasks

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

are mapped to frames, so that the schedulabiliy of the system can be improved; 2) since the period of each task is restricted to be the multiples of each frame, shorter frames enable the tasks have smaller periods.

The semi-partitioned scheduling [5], [10], [12], [22], [24], [25] allows some jobs to be split and migrate from one processor to another during their execution; once a job's execution is split into two subsections, the first subsection must normally be first executed on a processor in order to ensure this job will not be executed in parallel on two different processors. This scheme almost entails all the advantages of fully partitioned scheduling, while it can further maximise the total processor utilization by fully occupying the spare processor intervals where a whole job's execution cannot fit into.

This paper is based on the similar system model [13], [14] of mixed criticality cyclic executives which follows industrial practice [8] and assumes that all jobs of tasks have been allocated to each frame according to a given policy. The parameters and criticalities of the jobs are given by the system definition. In each frame, there is a switching time point S, before time S, only HI-criticality jobs are executed, if there is no criticality change, only LO-criticality jobs are executed after S; if there is a criticality change before S, the LO-criticality jobs are abandoned in this frame and the unfinished HI-criticality jobs must execute their extra computation times.

Compared with the scheme where all HI-criticality and LO-criticality jobs are executed together at the beginning of each frame, the advantages of this system model are: 1) This strategy enables composable certifiability if all HI-criticality jobs are temporally isolated from the LO-criticality jobs, namely the execution of a HI-criticality job must not be delayed in any way by a LO-criticality job [21]. 2) Even if there is a criticality change in a frame, there will be more HI-criticality jobs to have already finished their executions since they are always the first ones to be executed, hence a reduced number of HI-criticality jobs require to execute their extra computation times after the criticality change time. This can save processor capacity and reduce average response times for jobs in realistic systems.

The main contributions of this paper are summarized as follows. In order to improve the schedule within a frame, we present complete algorithms in this paper with the purpose of reducing the length of each frame to an optimal value, which is a necessary schedulability condition for any schedule under this system model (i.e. every processor's capacity is fully utilized within each frame). These algorithms provide solutions to the following issues: (1) how to partition the jobs among processors before and after time S; (2) how to implement the executions of jobs in each frame when the schedule is being remapped; (3) how to rearrange the schedule in order to reduce each frame's length to a minimum possible value. By conducting experiments on a large number of randomly generated job sets, we demonstrate that the proposed algorithms significantly reduce the length of frames. With the exception of a limited number of cases, 100% of these task sets get optimal frames after the improvements. We also prove that, when the completion times of some specific jobs before the time S are fixed values, the results presented in this paper are optimal.

II. SYSTEM MODEL AND EXISTING RESULTS

A cyclic executive system consists of continuously repeated frames (or minor cycle) T_F and major cycles T_M, and the values satisfy $T_M = kT_F$, where k is a positive integer, defining the number of frames in each major cycle. The jobs allocated to each frame execute repeatedly within a continuous execution of a series of frames. Each task's period is constrained to be multiple of T_F up to the value of T_M.

The mathematic problem of how to choose T_F and T_M to best support a given task set is beyond the scope of this paper. This paper is based on the same system model [13], [14] which follows industrial practice [8], it assumes that the jobs of tasks are allocated to each frame with fixed parameters by the system definition. The time units of all jobs' executions are mapped within a frame $[0, D)$, and they must complete by the end of each frame, with some additional timing constraints to support mixed criticality systems which will be detailed later in this section. Therefore, D is also the relative deadline of all allocated jobs in this frame. Our purpose of this paper is to reduce the length (or shorten the required completion time of all jobs) of any given frame by rearranging and partitioning the jobs' executions in this frame, under the assumption that all jobs have already been allocated to each frame according to a given policy, so that the schedulability of such a system can be improved.

The system executes on an m identical processor platform with two criticality levels. Let i denote the i-th task τ_i in the system, and j_i represents a job of τ_i. Each job j_i of task τ_i has a criticality attribute χ_i which can either be HI-criticality ($\chi_i = HI$) or low LO-criticality ($\chi_i = LO$); χ_i is determined by the designer of a system and never changes during the operation of a system. Each HI-criticality job has two worst-case execution times (WCETs) or computation times $C_i(HI)$ and $C_i(LO)$, and each LO-criticality job has only one WCET $C_i(LO)$.

A switching time point S ($0 < S < D$) is set in the time interval $[0, D)$ of each frame. During the interval $[0, S)$, all processors are executing ONLY HI-criticality jobs' $C_i(LO)$. At the start of each frame, the system is operating under a low criticality model. If each HI-criticality job executes for no more than $C_i(LO)$, in the next interval $[S, D)$, all processors are only executing LO-criticality jobs.

If any HI-criticality job j_i executes for more than $C_i(LO)$ in the interval $[0, D)$, the criticality of the system changes from low to high, then all HI-criticality jobs must execute their $C_i(HI)$ and complete by the end of the frame D. In this case, the switching time point S becomes meaningless and all LO-criticality jobs are abandoned in this frame. In summary, the timing constraints for each frame are described as follows.

(1) S is the deadline of all HI-criticality jobs' $C_i(LO)$ if there is no criticality change.
(2) D is the deadline of all LO-criticality jobs which start their executions at time S if there is no criticality change.
(3) D is also the deadline of all HI-criticality jobs' $C_i(HI)$ if there is any criticality change before or at time S; in this case, all LO-criticality jobs are abandoned and restrictions (1) and (2) are no longer valid.

Most existing literatures on mixed criticality systems (e.g. [4], [7], [9], [16]–[19], [23], [27], [29]) do not consider cyclic executives with semi-partitioned schemes. In the remainder of this section, we describe some existing results from [13], [14]; our improvements in the next section will be based on these results.

Let $C_i(EX)$ be the extra computation time required by a HI-criticality job when a criticality change occurs, i.e. $C_i(EX) = C_i(HI) - C_i(LO)$. The original schedule is to let all HI-criticality jobs' $C_i(LO)$ fit into the interval $[0, S)$, and to let all $C_i(EX)$ fit into the interval $[S, D)$.

In order to allow all HI-criticality jobs' $C_i(LO)$ to fit into $[0, S)$ on an m identical processor platform, according to the optimal scheme of McNaughton [28],

$$S = \max\left(\sum_{\chi_i=HI} C_i(LO) \Big/ m, \max_{\chi_i=HI}\{C_i(LO)\}\right). \quad (1)$$

In order to let all HI-criticality jobs' $C_i(EX)$ fit into the interval $[S, D)$, the length of $[S, D)$ must be greater than

$$\Delta^{HI} = \max\left(\sum_{\chi_i=HI} C_i(EX) \Big/ m, \max_{\chi_i=HI}\{C_i(EX)\}\right)$$

Let D^{HI} be a frame's deadline while considering only the length of the time interval when all HI-criticality jobs can complete their executions in the case of a criticality change occurring. Therefore, for the original schedule,

$$D^{HI} = S + \Delta^{HI}.$$

Let Δ^{LO} be the minimum length of the time interval required by all LO-criticality jobs in order to execute their $C_i(LO)$. Additionally, the value of Δ^{LO} can be calculated using the same approach outlined in equation (1). Since all LO-criticality jobs must start and finish their executions in $[S, D)$ in the case of a criticality change occurring, the real deadline of a frame is given by

$$D = \max\left(D^{HI}, S + \Delta^{LO}\right). \quad (2)$$

This paper addresses the problem of improvements to D^{HI}. Due to the definitions of the system model, the values of S and Δ^{LO} are fixed and they cannot be improved. The frame's real deadline can be simply calculated by equation (2).

In order to let each HI-criticality job's $C_i(EX)$ fit into $[S, D^{HI})$, the calculation of D^{HI} depends on the value of $\sum_{\chi_i=HI} C_i(EX) \Big/ m$. There are two cases:

Case (1). When $\max_{\chi_i=HI}\{C_i(EX)\} \le \sum_{\chi_i=HI} C_i(EX) \Big/ m$, according to McNaughton's scheme, the amount of the $\sum_{\chi_i=HI} C_i(EX) \Big/ m$ units of computation time is equally allocated to each processor. There is no possibility to improve schedulability under this case, and the deadline of each frame is given by

$$D^{HI} = S + \sum_{\chi_i=HI} C_i(EX) \Big/ m. \quad (3)$$

It is obvious that the deadline given by equation (3) is a necessary schedulability condition under this system model, and it is an optimal/minimum deadline for any schedule when S is a fixed value.

Case (2). When $\max_{\chi_i=HI}\{C_i(EX)\} > \sum_{\chi_i=HI} C_i(EX) \Big/ m$,

As pointed out in [13], according to the system model and the optimal scheme of McNaughton [28], the deadline of a frame without improvements is given by

$$D^{HI} = S + \max\left(\sum_{\chi_i=HI} C_i(EX)/m, \max_{\chi_i=HI}\{C_i(EX)\}\right) \quad (4)$$

Note that according to the definition of cyclic executives, every allocated job is required to start and finish within each frame, so that the deadline of a frame is also the required length of a frame.

The purpose of this paper is to improve the schedulability of the HI-criticality jobs for Case (2) so that the length of each frame can be reduced, while all timing requirements are satisfied and the switching point S remains the same value.

III. IMPROVEMENT TO SCHEDULABILITY

This section improves the schedulability of the system by reducing the original deadline of the frame D^{HI} given by equation (4).

Our objective of this section is to improve the implementation and to reschedule the HI-criticality jobs' executions in $[0, D^{HI}]$ so that the required length of each frame can be reduced from equation (4) towards the optimal value given by equation (3), while the value of remains unaffected.

The algorithms in this section are introduced as follows. In each frame, all HI-criticality jobs' $C_i(EX)$ are initially allocated by Algorithm 1 in the interval $[S, D^{HI}]$; all HI-criticality jobs' $C_i(LO)$ are initially allocated by Algorithm 2. The implementation of the jobs for the improvement is described in Algorithm 3. Finally, Theorem 2 indicates the extend of the improvement to D^{HI} that can be achieved by Algorithm 3 and Algorithm 4.

A. INITIAL ASSIGNMENT IN $[S, D^{HI}]$ AND $[0, S]$

This section describes our initial partition strategies for HI-criticality jobs in the intervals $[S, D^{HI}]$ and $[0, S]$. The purpose of this section is to let the HI-criticality jobs satisfying $C_i(EX) > \sum_{\chi i=HI} C_i(EX/m)$ finish as soon as possible in $[S, D^{HI}]$ in order to execute the $C_i(LO)$ of the HI-criticality jobs, such that more workload in $[S, D^{HI}]$ could be brought forward before time S.

Considering that the jobs with

$$C_i(EX) > \sum_{\chi_i=HI} C_i(EX)/m$$

Algorithm 1 The Allocation Policy of All $C_i(EX)$ in the Interval $[S, D]^{HI}$ Is Described as Follows:

1) Since each job satisfying $C_i\,(EX) \geq Avg\,(EX)$ must hold one processor, allocate each Γ_{sub}'s job to a processor starting from processor P_1 to P_α, where α is the number of jobs in Γ_{sub} given by Definition 1; each processor executes only one such a job.
2) Equally distribute the amount of all other jobs' $C_i\,(EX)$ to the remaining processors from $P_{\alpha+1}$ to P_m (by splitting the last job which cannot completely fit into a processor), where m is the number processors in the system. As a result, each processor receives the execution time amount of

$$\sum\nolimits_{C_i(EX)<Avg(EX)} C_i\,(EX) \Big/ (m-\alpha),$$

are the "problem" jobs that need to be improved, the following definition is used throughout all the algorithms

Definition 1: Let $Avg(EX) = \sum_{\chi_i=HI} C_i(EX)/m$ *and let the subset of the jobs satisfying* $C_i(EX) \geq Avg(EX)$

$$\Gamma_{sub} = \{j_{s1}, j_{s2}, \ldots, j_{s\alpha}\},$$

where $j_{s\kappa}$ *represents any given job in* Γ_{sub}, $1 \leq \kappa \leq \alpha$, α *is the number of jobs in* Γ_{sub}, *and the jobs in* Γ_{sub} *have been sorted by their* $C_{s\kappa}\,(EX)$ *values in descending order, namely*

$$C_{s1}\,(EX) \geq C_{s2}\,(EX) \geq \ldots \geq C_{s\alpha}\,(EX).$$

therefore, we let

$$t_2 = S + \frac{\sum_{C_i(EX)<Avg(EX)} C_i\,(EX)}{m-\alpha}. \tag{5}$$

$$\text{Let } t_3 = S + Avg(EX), \tag{6}$$

and the initial D^{HI} was given by equation (4). It is clear that $t_2 \leq t_3 \leq D^{HI}$, as shown in Figure 1.

In order to reduce D^{HI}, it is necessary to bring forward $C_{s\kappa}\,(EX)$ from any job of Γ_{sub} as much as possible (such as the interval $[t_3, D^{HI}]$ on P_1 in Figure 1, so that if there is a criticality change before the completion time of a job's $C_{s\kappa}\,(LO)$, its $C_i\,(EX)$ can partially get executed before time S. Resultantly, the deadline of each frame is reduced.

Because any amount of $C_{s\kappa}\,(EX)$ that is brought forward before S must be executed after its $C_{s\kappa}\,(LO)$'s completion time, in order to bring back the time units as much as possible, we must arrange a schedule for all HI-criticality jobs' $C_i(LO)$ in the interval $[0, S]$, in order to allow each $C_{s\kappa}(LO)$ from Γ_{sub} to complete as early as possible.

The following algorithm provides such a schedule for the interval $[0, S]$ ensuring that each $C_{s\kappa}\,(LO)$ from Γ_{sub} gets executed and finished as early as possible, and the job with the greatest value of $C_{s\kappa}\,(EX)$ can always be executed as a priority, hence more time units can be brought forward.

Note that since case (b) is a very special case of this step, the implementation of step (6) could be simplified as: if $\beta + \alpha > m$, no improvement is made.

Algorithm 2 Each HI-Criticality Job's $C_i(LO)$ in the Interval $[0, S]$ Is Allocated and Implemented by the Following Steps

(1) Distribute each job satisfying $C_i\,(LO) \geq \sum_{\chi_i=HI} C_i\,(LO) \Big/ m$ to different processors starting from processor P_m towards P_1, as shown in Figure 2.
(2) Distribute each $C_{s\kappa}\,(LO)$ of Γ_{sub}'s jobs to different processors from P_1 to P_α, where $\kappa = 1, \ldots, \alpha$ given by Definition 1. Note that each $j_{s\kappa}$'s $C_{s\kappa}(EX)$ has been in fact assigned to the same processor with its $C_{s\kappa}\,(LO)$, as described in Algorithm 1.
According to Definition 1, the job with the greatest value of $C_{s\kappa}\,(EX)$ will be firstly allocated to P_1, then the second largest one is allocated to P_2, and so on, as illustrated in Figure 2. If there are more than two jobs having the same $C_{s\kappa}\,(EX)$ values, the job with the minimum $C_{s\kappa}\,(LO)$ value is allocated first.
(Step (2) ensures each $C_{s\kappa}\,(LO)$ of Γ_{sub} finishes as soon as possible, although the first subsection of a split job will execute firstly on the same processor if provided; this will be described in step (5).)
(3) Allocate all other HI-criticality jobs (not from Γ_{sub}) to P_1 in any order, until a job j_ℓ (coloured by yellow in Figure 3) cannot be completely fit into P_1. Then split j_ℓ's $C_\ell\,(LO)$ into two subsections, let the second subsection occupy all remaining time on processor P_1 before S, and let the first subsection of j_ℓ execute firstly on P_2, such that the first subsection of j_ℓ has the highest priority on the designated processor, as illustrated in Figure 3. This means that if there is an assigned job from Γ_{sub} already on P_2, this job has to wait until the first subsection of j_ℓ completes its execution.
(4) If the first subsection of j_ℓ cannot be completely fit into on P_2 (due to a job of Γ_{sub} that has been allocated to P_2), then find another processor to execute. If the first subsection of j_ℓ cannot fit into any processor, then no improvement is made.
(5) Allocate the remaining HI-criticality jobs to P_2 in the same manner of steps (3)(4), and then $P_3, \ldots, P_m$, until all HI-criticality jobs' $C_i(LO)$ have been partitioned.
Note that based on step (5) of Algorithm 2, in all the figures of this section, any empty space between 0 and S does not represent the processor idle time, there could also be executions of jobs which are not specified in each figure. This point will be emphasised again in Algorithm 3.

Algorithm 2 (*Continued*) Each HI-Criticality Job's $C_i(LO)$ in the Interval $[0, S]$ Is Allocated and Implemented by the Following Steps

(6) Let β denote the number of HI-criticality jobs with $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO) \Big/ m$. If $\beta + \alpha > m$, where α is the number of jobs in Γ_{sub}, there are two cases:
Case (a): There are no jobs satisfying both conditions $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO) \Big/ m$ and $C_i(EX) > Avg(EX)$, as shown in Figure 4. In this case, we have to allocate a job satisfying $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO) \Big/ m$ to a different job satisfying $C_i(EX) > Avg(EX)$ on the same processor (e.g. processor 3 in Figure 4) before time S. This could result in the failure of deadline S if there is no criticality change. Therefore, no improvement is made in this case.
Case (b): There is at least one job satisfying $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO) \Big/ m$ and $C_i(EX) > Avg(EX)$. In this case we always allocate such a job's $C_i(LO)$ and $C_i(EX)$ to the same processor, as shown in Figure 5. Let the number of such jobs be χ, if $\beta + \alpha - \chi \leq m$, the improvement is still possible; if $\beta + \alpha - \chi > m$, no improvement can be made.

B. SWAP-IN/OUT THE TIME UNITS

While some workloads of $C_{s\kappa}(EX)$ are brought forward to the interval $[0, S]$, the same amount of $C_{s\kappa}(LO)$ may have to be brought out to be executed after time S. In order to ensure the timing requirements of the system model can be satisfied and the value of S is not affected, the following regulations are presented for the implementation of such an exchange in time units.

Based on regulation (1), in all the figures of this section, any empty space in $[0, S]$ on each processor does not represent the processor idle time, and furthermore there are also jobs executing.

Regulation (2) of Algorithm 3 guarantees that the value of S is not affected by the swap-in/out process, and all the timing restrictions of section 2 are satisfied.

Then it is necessary to establish how many time units of $C_{s\kappa}(EX)$ for each Γ_{sub}'s job can be swapped in, and how to "swap out" the time units of $C_i(LO)$ upon executing them after S.

Lemma 1: There are always at least $m - \alpha$ processors not executing any Γ_{sub}'s job at any instance of time.

Proof: Since the number of jobs in Γ_{sub} is α, and any job can execute on only one processor at any instance of time, the conclusion holds.

Theorem 1: By Algorithm 3 and Algorithm 4, when a criticality change occurs in a system, the deadline of any given job $j_{s\kappa}$ from Γ_{sub} can be reduced to

$$d_{s\kappa} = S + C_{s\kappa}(EX) - \min\{S - t_{s\kappa}, S + C_{s\kappa}(EX) - t_3\}; \quad (7)$$

Algorithm 3 The Implementation of the HI-Criticality Jobs Obeys the Following Regulations

(1) We assume that at the beginning, the interval $[0, S)$ is fully occupied by the HI-criticality jobs' $C_i(LO)$ on every processor. Therefore any time units of $C_{s\kappa}(EX)$ brought forward before S potentially result in the same amount of $C_i(LO)$ swapping out after S.
(2) Any "swapped-out" time units of $C_i(LO)$ are not really out, they are still placed before S sharing the exact same time interval (e.g. $[x_1, x_2]$, where $0 < x_1 < x_2 \leq S$) with the corresponding "swapped-in" time units of $C_{s\kappa}(EX)$ on the same processor. As a result, there are two cases:
   (a) If there is no criticality change before the finish time of a "swapped-in" $j_{s\kappa}$'s $C_{s\kappa}(LO)$, job $j_{s\kappa}$ finishes in the anticipated time. The "swapped-in" $j_{s\kappa}$'s execution time units are abandoned since $j_{s\kappa}$ no longer needs to be considered even if a criticality change occurs after $j_{s\kappa}$'s finish time. Therefore the executions of the "swapped-out" jobs are not affected at all, they get executions before time S simply by following the allocation policy of Algorithm 2.
   (b) If there is a criticality change before the finish time of a "swapped-in" $j_{s\kappa}$'s $C_{s\kappa}(LO)$, the "swapped-in" $C_{s\kappa}(EX)$ is executed in the shared time interval $[x_1, x_2]$ (after its $C_{s\kappa}(LO)$'s finish time). Therefore the "swapped-out" job cannot be executed in $[x_1, x_2]$ in this case, and it has to execute these $x_2 - x_1$ time units' $C_i(LO)$ after S in a specified time interval on a designated processor.

equation (7) can also be presented as

$$\textbf{\textit{if}}\ (S - t_{s\kappa} \geq S + C_{s\kappa}(EX) - t_3) d_{s\kappa} = t_3;$$
$$\textbf{\textit{else}}\ d_{s\kappa} = C_{s\kappa}(EX) + t_{s\kappa};$$

where S is calculated by equation (1), t_3 is given by equation (6), and $t_{s\kappa}$ is the completion time defined in step (1) of Algorithm 4.

Proof: Algorithm 4 has provided details of the steps indicating how to swap-in the time units of $C_{s\kappa}(EX)$ placed before time point S whilst swapping-out the same amount of time units of $C_i(LO)$ placed after S. By Algorithm 3, the "swapped out" time units are still placed before S and share the same time interval of the "swapped in" units of $C_{s\kappa}(EX)$. Therefore, the value of S is not affected at all. In step (3) of Algorithm 4, we have proved that case (a) is the only situation in which $j_{s\kappa}$'s finish time cannot be further moved to an earlier instance. In other words, we can always find processors to swap-in the computation time of $C_{s\kappa}(EX)$ if there is sufficient space between $t_{s\kappa}$ and S.

Algorithm 4 The Processing of Swap-in/Out Can Be Carried Out According to the Following Steps

Step (1). Partition all HI-critical jobs' computation times in the interval $[0, D^{HI}]$ according to Algorithm 1 and Algorithm 2, and then let the current completion times of $C_{s\kappa}$ (LO) from the Γ_{sub}'s jobs be $\{t_{s1}, t_{s2}, \ldots, t_{s\alpha}\}$, where Γ_{sub} and α are given by Definition 1, as illustrated in Figure 1.

Step (2). Let $L^{\max}_{mov.s1} = S - t_{s1}$, $L^{\max}_{mov.s2} = S - t_{s2}$, …, $L^{\max}_{mov.s\alpha} = S - t_{s\alpha}$, which represent the upper bounds of the move-in time units for jobs $j_{s1}, j_{s2}, \ldots, j_{s\alpha}$ in Γ_{sub}. Starting from processor P_1, "swap in" the $L^{(1)}_{mov.s1}$ time units of C_{s1} (EX) to the interval $\left[t_{s1}, t_{s1} + L^{(1)}_{mov.s1}\right]$ on the same processor, where

$$L^{(1)}_{mov.s1} = \min\left\{t_3 - t_2, S + C_{s1}(EX) - t_3, L^{\max}_{mov.s1}\right\};$$

and "swap out" the $L^{(1)}_{mov.s1}$ time units of C_i (LO) to the interval $\left[t_2, t_2 + L^{(1)}_{mov.s1}\right]$ on processor $P_{\alpha+1}$, as illustrated in Figure 6. Note that the swap-out $L^{(1)}_{mov.s1}$ time units of C_i (LO) could come from more than one job.

Step (3). This step studies the possibility of bringing forward C_{s1} (EX) of j_{s1} to a further extend. Denote $C^{(2)}_{s1}$ (EX) to be the remainder of C_{s1} (EX) after step (2), i.e. $C^{(2)}_{s1}(EX) = C_{s1}(EX) - L^{(1)}_{mov.s1}$. If $S + C^{(2)}_{s1}(EX) = t_3$ (i.e. $L^{(1)}_{mov.s1} = S + C_{s1}(EX) - t_3$), the finish time of j_{s1} has been moved up to t_3, then this step ends. If $S + C^{(2)}_{s1}(EX) > t_3$ (i.e. $L^{(1)}_{mov.s1} < S + C_{s1}(EX) - t_3$), as shown in Figure 6, there are two cases:

(a) $L^{(1)}_{mov.s1} = L^{\max}_{mov.s1} < S + C_{s1}(EX) - t_3$; or
(b) $L^{(1)}_{mov.s1} = t_3 - t_2 < S + C_{s1}(EX) - t_3$.

If case (a) is true, j_{s1}'s finish time can no longer be improved, and step (3) ultimately ends.

If case (b) is true, this means j_{s1}'s $C^{(2)}_{s1}$ (EX) can be further brought in,

let

$$L^{(2)}_{mov.s1} = \min\left\{t_3 - t_2, S + C^{(2)}_{s1}(EX) - t_3, L^{\max}_{mov.s1} - L^{(1)}_{mov.s1}\right\},$$

then swap in the $L^{(2)}_{mov.s1}$ time units of $C^{(2)}_{s1}$ (EX) to the interval

$$\left[t_{s1} + L^{(1)}_{mov.s1}, t_{k1} + L^{(1)}_{mov.s1} + L^{(2)}_{mov.s1}\right].$$

Therefore, when $S - t_{s\kappa} \geq S + C_{s\kappa}(EX) - t_3$, the completion time of $j_{s\kappa}$ can be reduced to t_3. When $S - t_{s\kappa} < S + C_{s\kappa}(EX) - t_3$, since the swap-in time units of $C_{s\kappa}(EX)$ must be placed after $t_{s\kappa}$, the maximum amount of swap-in time units is $S - t_{s\kappa}$, so the completion time of $j_{s\kappa}$ becomes

$$S + C_{s\kappa}(EX) - (S - t_{s\kappa}) = C_{s\kappa}(EX) + t_{s\kappa}.$$

Algorithm 4 (*Continued*) The Processing of Swap-in/Out Can Be Carried Out According to the Following Steps

Since $L^{(1)}_{mov.s1} = t_3 - t_2$ in case (b), the "swapped out" $L^{(2)}_{mov.s1}$ time units must be placed on another processor, as shown in Figure 7, so $L^{(1)}_{mov.s1}$ and $L^{(2)}_{mov.s1}$ must NOT come from the same job. Therefore, the "swapped in" $L^{(2)}_{mov.s1}$ time units have to be placed on another processor or more than one processor which has no Γ_{sub}'s job execution in this interval. Whilst all the "swapped out" time units must not come from the Γ_{sub}'s jobs, and they are placed in $[t_2, t_3]$ from processor $P_{\alpha+1}$ to P_m. In line with Lemma 1, at any instance of time, there are always at least $m - \alpha$ processors are available to make this swapping in/out possible. (It is recommended to place the initial "swapped in" time units on the same processor if it is possible (e.g. P_1 in Figure 6) to reduce job migrations, and search processors for the secondary "swapped in" time units starting from $P_{\alpha+1}$ (e.g. P_3 in Figure 7).)

Step (4). The process of step (3) continues until the finish time of j_{s1} becomes t_3
(i.e. $L^{(1)}_{mov.s1} + L^{(2)}_{mov.s1} + \ldots + L^{(N)}_{mov.s1} = S + C_{s1}(EX) - t_3$, where (N) is the number of movements for j_{s1}), or until there is no space between t_{s1} and S in order to accommodate any new "swap-in" time units
(i.e. $L^{(1)}_{mov.s1} + L^{(2)}_{mov.s1} + \ldots + L^{(N)}_{mov.s1} = L^{\max}_{mov.s1}$).

Step (5). Repeat steps (2) and (3) for other jobs in Γ_{sub}, as shown in Figure 8. Fully fill in the interval $[t_2, t_3]$ on each processor from $P_{\alpha+1}$ to P_m by the "swapped out" time units. Finally we may get an improved deadline in the form of t_3 which is the greatest lower bound for any scheduling algorithm.

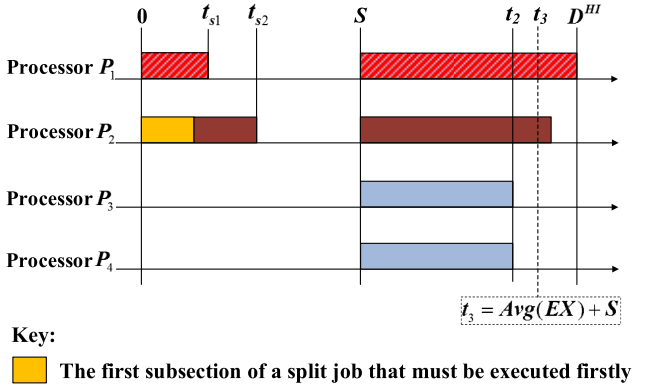

FIGURE 1. Illustrative example of Algorithm 1 and step (1) of Algorithm 4.

Theorem 2: By Algorithm 3 and Algorithm 4, the deadline of a frame D^{HI} can be reduced to

$$\max_{j_{s\kappa} \in \{j_{s1}, j_{s2}, \ldots, j_{s\alpha}\}} \{S + C_{s\kappa}(EX) - \min\{S - t_{s\kappa}, S + C_{s\kappa}(EX) - t_3\}\}$$

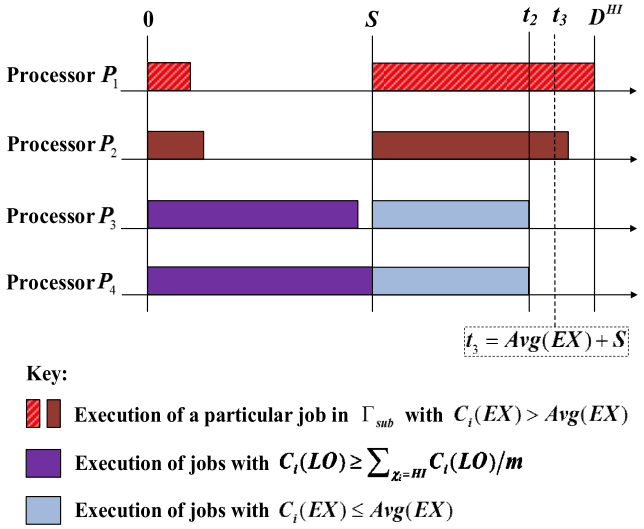

FIGURE 2. Illustrative example of steps (1) and (2) of Algorithm 2.

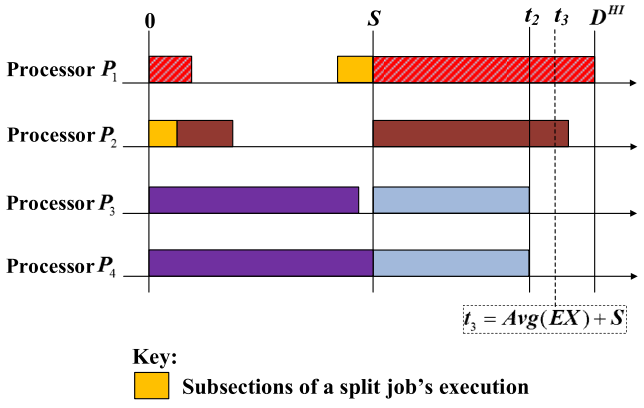

FIGURE 3. Illustrative example of step (3) of Algorithm 2.

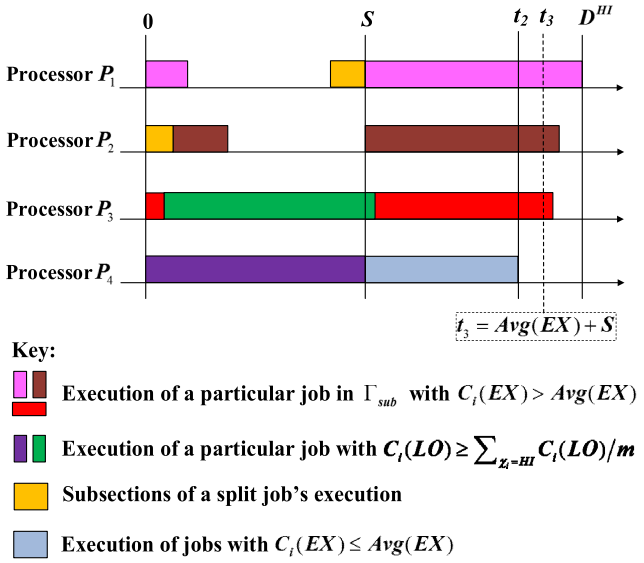

FIGURE 4. Illustrative example of Algorithm 2 when $\alpha + \beta > m$, and there are no jobs satisfying both conditions $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO)/m$ and $C_i(EX) > Avg(EX)$.

Proof: Since all HI-criticality jobs with the exception of Γ_{sub} finish at or before time t_3, only the jobs of Γ_{sub} could

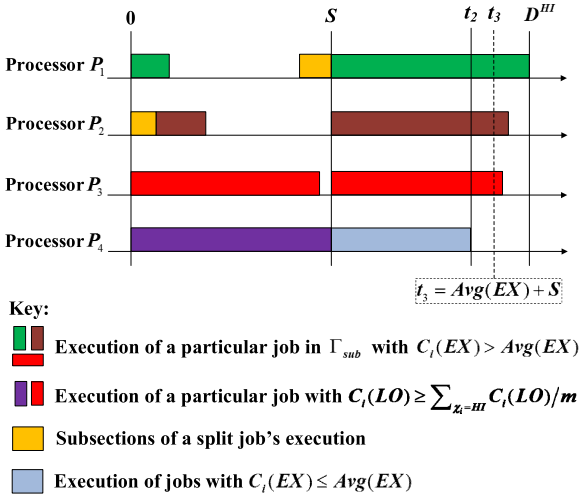

FIGURE 5. Illustrative example of Algorithm 2 when $\alpha + \beta > m$, and there is at least one job satisfying both conditions $C_i(LO) \geq \sum_{\chi_i=HI} C_i(LO)/m$ and $C_i(EX) > Avg(EX)$.

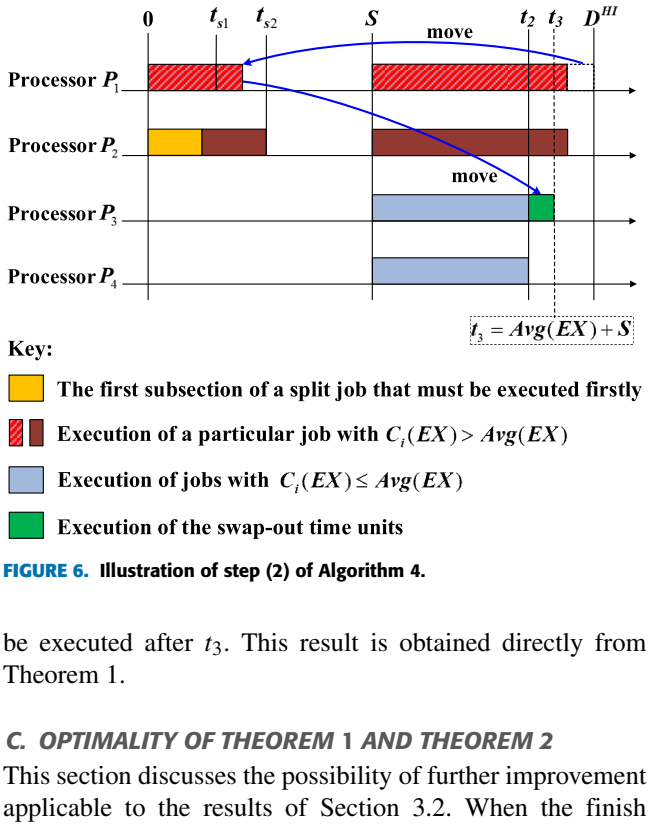

FIGURE 6. Illustration of step (2) of Algorithm 4.

be executed after t_3. This result is obtained directly from Theorem 1.

C. OPTIMALITY OF THEOREM 1 AND THEOREM 2

This section discusses the possibility of further improvement applicable to the results of Section 3.2. When the finish times of C_{sK} (LO) from Γ_{sub}'s jobs (i.e. the set of time points $\{t_{s1}, t_{s2}, \ldots, t_{s\alpha}\}$ defined in Algorithm 4) are a fixed values, the following theorem proves that the approaches of Section 3.2 provide the best schedulability of the system.

Theorem 3: When the finish times of $C_{sK}(LO)$ from Γ_{sub} are fixed given values, the deadlines (or the length of time required to complete the jobs) given by Theorem 1 and Theorem 2 are optimal.

Proof: It is obvious that for any job j_{sK}, its C_{sK} (EX) can only execute after its C_{sK} (LO)'s finish time t_{sK}, so that the

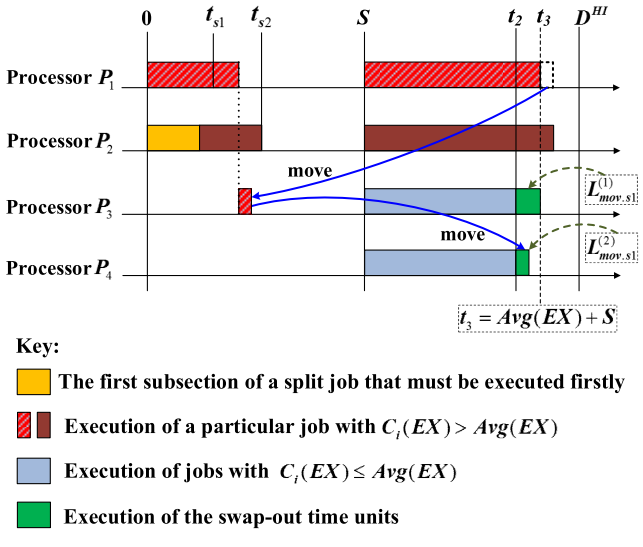

FIGURE 7. Illustration of step (3) of Algorithm 4.

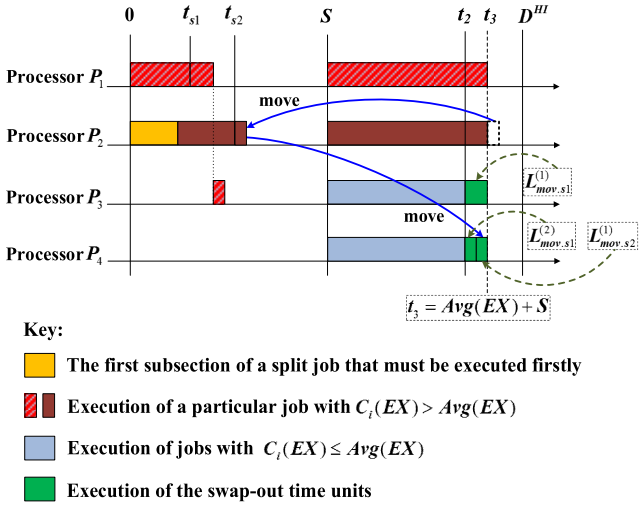

FIGURE 8. Illustration of step (4) of Algorithm 4.

length of the interval between t_{sK} and S is the upper bound for the amount of time units in the case of which C_{sK} (EX) could be swapped in before S. By Algorithm 4, we have already proved that, unless there is insufficient space in $[t_{sK}, S]$, every step is possible until the deadline of j_{sK} has been reduced to t_3. Therefore, the results of Section 3.2 are optimal.

D. COMPLEXITY OF THE ALGORITHMS

Algorithm 1 and Algorithm 2 are ordinary semi-partitioned strategies to allocate the load of jobs to each processor. The complexity of each algorithm depends on the number of jobs (denoted by n) required to be allocated and on the number of processors (denoted by m) in a system. An upper bound for the number of such allocations is $n \times m$.

For Algorithm 4, since the processing of swap-in/out depends on two factors:

1) the number of jobs in Γ_{sub} (with $C_i\,(EX) \ge Avg\,(EX)$);

2) how many time units are available on a chosen processor for swapping according to the algorithm.

Therefore, an upper bound for the number of steps (calculations) of the swap-in and swap-out is $\alpha \times m$, where α is the number of jobs in Γ_{sub}.

IV. EXPERIMENTS AND EVALUATIONS

This section evaluates the extent to which improvements can be achieved by the scheduling algorithms proposed in Section 3 on the basis of an extensive number of experiments based on a large number of job sets which are randomly generated. As described in Section 2, since the values of S and Δ^{LO} are already optimal due to the system definition, this paper reduces the length of each frame by reducing D^{HI} for the HI-criticality jobs in the event of a criticality change occurring.

The following main issues are addressed in this section.

(1) The average length of the deadline D^{HI} that can be reduced.
(2) The extent of the improvement of the total processor utilization to be obtained, and the average total processor utilization of the job sets after rescheduling.
(3) Since the proposed algorithms provide an opportunity to obtain an optimal/minimum deadline given by equation (3), we will investigate what percentage of the randomly generated job sets can obtain the optimal values.
(4) The percentage of the job sets which cannot be improved due to failures of step (4) or step (6) of Algorithm 2 in relation to allocating each HI-criticality job's C_i (LO) to processors.

The WCETs of the jobs in each set are randomly generated between 1 and 1000, representing the computation times ranging from 1ms to 1s which is very common in real systems. In order to obtain uniformly distributed WCETs in the range 1-1000, we let the job parameters be generated by the same approach provided in [30] according to an exponential distribution. For example, if the number of jobs for each system is 17, we first let $C_{17}(LO) = 1000$, and then let 16 computation times be generated between 1 and 1000. Since $\ln 1000 \cong 6.9$, the range of $C_i(LO)$ is divided into 7 intervals $e^o \sim e^1$, $e^1 \sim e^2$, …, $e^6 \sim 1000$, two computation times are generated randomly in each interval from $C_1(LO)$ to $C_{14}(LO)$, and for the other two computation times $C_{15}(LO) \sim C_{16}(LO)$, one is generated in each of the two minimum intervals.

The parameters of $C_i(LO)$ and $C_i\,(EX)$ for the HI-criticality jobs are generated separately. In order to avoid always having $C_n(LO) = C_n\,(EX) = 1000$, where n is the number of HI-criticality jobs, we let $C_i\,(EX)$ be generated in the reverse order of $C_i(LO)$, that is to say they are generated from $C_n(EX)$ to $C_2(EX)$ in the same intervals.

Each point on the diagrams of this section is the average of 30,000 sets of jobs that are randomly generated by the above described policies.

A. REDUCTION OF FRAME (LENGTH OF COMPLETION TIME)

In this section, we evaluate to what extent the deadline D^{HI} (or the length of time required to complete all jobs) of each frame can be reduced by our algorithms. On the graphs, 'D-original' and 'D-improved' represent the average required D^{HI} before and after the proposed algorithms are applied.

In the first experiment, we let the number of jobs in each frame be varied from 4 to 24 when there are four processors in the system. The results are shown in Figure 9.

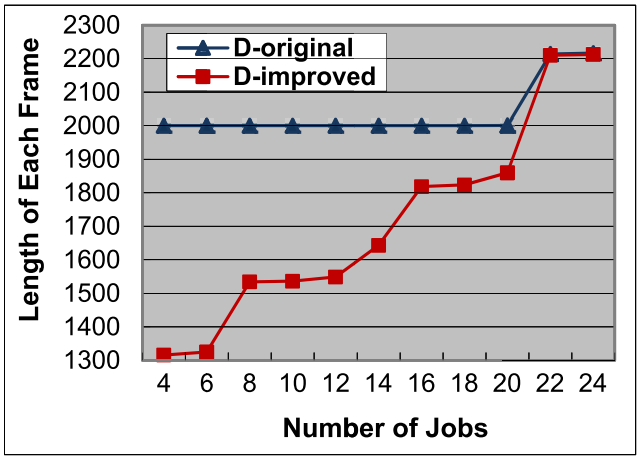

FIGURE 9. Average length of time required to complete all jobs in each frame as a function of the number of jobs on a four-processor platform.

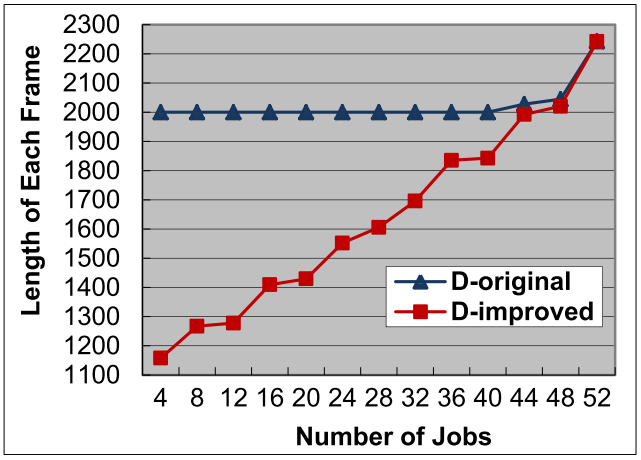

FIGURE 10. Average length of time required to complete all jobs in each frame as a function of the number of jobs on an eight-processor platform.

In the next experiment described by Figure 10, we change the number of processors to 8, and let the number of jobs vary between 4 and 52, so the average deadline is still a function of the number of jobs within each frame.

In the following experiment, we let the number of processors in each system vary between 2 and 16, and let the number of jobs in each frame be 12, such that the length of the deadline is expressed as a function of the processors' number. The results are presented in Figure 11.

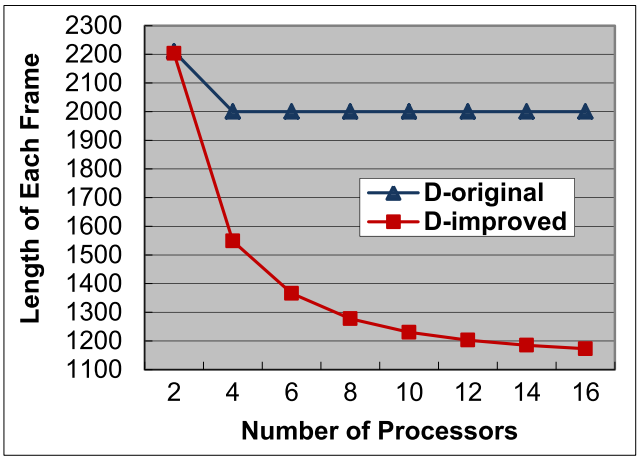

FIGURE 11. Average length of time required to complete all jobs in each frame as a function of the number of processors when there are 12 jobs in each frame.

B. POSSIBILITY OF FURTHER IMPROVEMENT

This section evaluates the possibility of further improvement by investigating the processor utilization in each given interval. In this section, we denote the new completion time of each frame to be D^{HI}_{New} after the proposed algorithms are applied, and denote D^{HI}_{Orig} to be the original completion time before the improvement. We investigate the total processor utilization of jobs in the intervals $[0, D^{HI}_{New}]$ and $[0, D^{HI}_{Orig}]$. This is because when the total processor utilization in a time interval reaches its maximum possible value (e.g. the processor utilization in $[0, D^{HI}_{New}]$ reaches m for a system with m processors), no algorithm can further reduce the length of deadline in this time interval.

In the first experiment, we let the parameters of each frame and the number of processors be exactly the same as the experiment described by Figure 9, and the number of jobs in each frame still varies between 4 and 28, thus the processor utilization is determined by the number of jobs. On the graph, 'U-original' represents the average processor utilization in the interval $[0, D^{HI}_{Orig}]$ before the improvement, and 'U-improved' represents the average processor utilizations in the interval $[0, D^{HI}_{New}]$ after the improvement is made. The results are shown in Figure 12.

We can see from the results of Figure 12, that the processor utilization in each interval almost reaches its maximum capacity 4 when the number of jobs is equal to or above 22. This means that when the number of jobs in each frame reaches 22 or more, no algorithm can further reduce the required length of such a frame.

Comparing the experimental results of Figure 9 with Figure 12, a comparative analysis can be found that before the processor utilization in each interval reaches its maximum possible capacity, our proposed algorithms significantly reduce the length of time required to complete the jobs, and the lower the processor utilization in an interval, the greater the improvement that can be made.

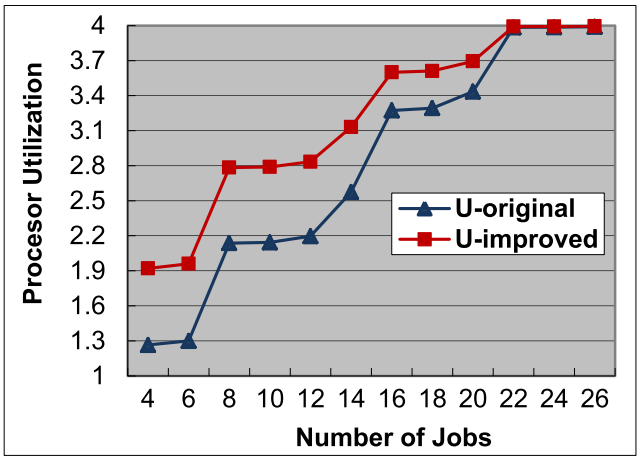

FIGURE 12. Total Processor utilization in the intervals $[0, D^{HI}_{New}]$ (U-improved) and $[0, D^{HI}_{Orig}]$ (U-original) as a function of the number of jobs on a four-processor platform.

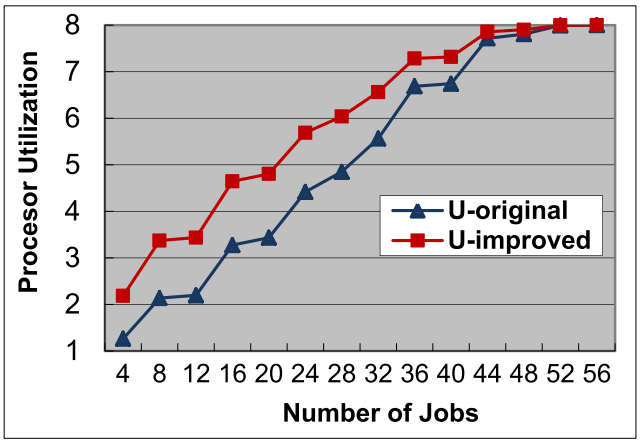

FIGURE 13. Total Processor utilization in the intervals $[0, D^{HI}_{New}]$ (U-improved) and $[0, D^{HI}_{Orig}]$ (U-original) as a function of the number of jobs on an eight-processor platform.

The second experiment described by Figure 13 has the same parameters and number of processors with the experiment of Figure 10. We can see from Figure 13 that when the number of jobs is above 44, the utilization in each interval almost reaches the maximum possible capacity. Comparing the experimental results of Figure 10 with Figure 9, we can get the same conclusion that before the utilization of each interval reaches its maximum value, our algorithms always greatly improve the completion of each frame.

Summary of Results From the experimental results of Section 4.A and Section 4.B, we can conclude that the proposed algorithms always significantly reduce the required length of each frame under nearly all situations as long as the total processor utilization in this time interval does not reach its maximum possible value. The proposed algorithms also significantly increase the total processor utilization in each frame before the processor utilization reaches its maximum possible value in this time interval. In each frame, when there are a reduced number of jobs, or a lower processor utilization, or there are more processors in the system, more improvements can be made by the algorithms.

C. PERCENTAGE OF OPTIMAL/MINIMUM LENGTH OF FRAMES OBTAINED

In this section, we first evaluate what percentage of the job sets can obtain the optimal length of frames (given by equation (3) which is a minimum possible value) before and after the proposed algorithms are applied; in this experiment, only the sets of jobs that can be successfully allocated to the processors by Algorithm 2 are considered. The results of the four-processor and eight-processsor platforms are shown in Figure 14 and Figure 15 respectively. Each point on the diagrams of this section is based on at least 40,000 randomly generated sets of jobs.

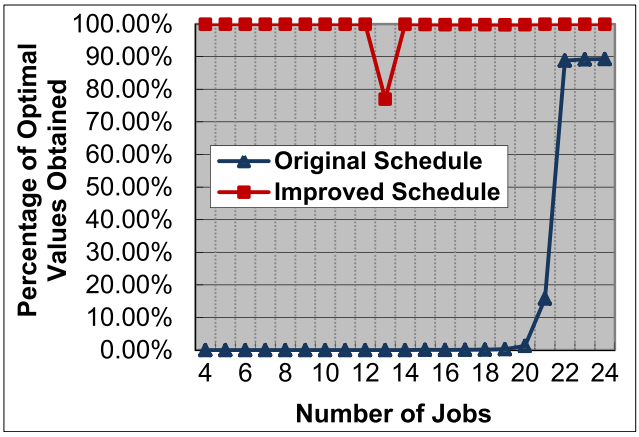

FIGURE 14. Percentage of the job sets which obtained the optimal length of frames before and after the improvement on a four-processor platform.

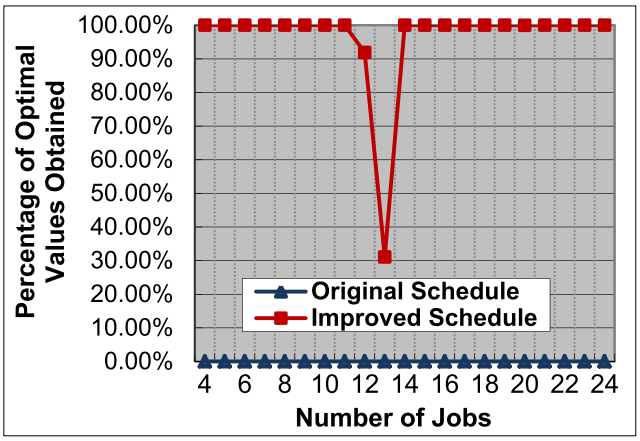

FIGURE 15. Percentage of the job sets which obtained optimal length of frames before and after the improvement on an eight-processor platform.

We can see in Figure 14 and Figure 15 that with the exception of one or two specific points ($n = 13, n = 12$), almost 100% of the job sets get optimal length of frames in all situations after the improvement, while the original schedule always has 0% of the optimal values in every situation

before the number of tasks is increased to a certain value which results in the maximum capacity of the total processor utilization in each time interval.

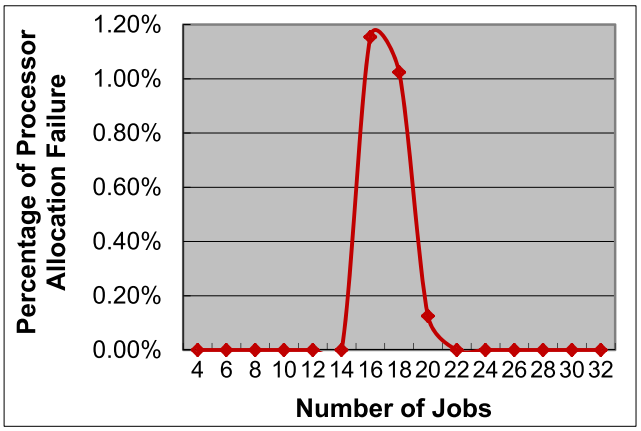

FIGURE 16. Percentage of the job sets that cannot be improved due to processor allocation failure on a four-processor platform.

In the following experiment, we investigate what percentage of the job sets cannot be improved due to the processor allocation failures by Algorithm 2. The numbers are reported in Figure 16 and Figure 17.

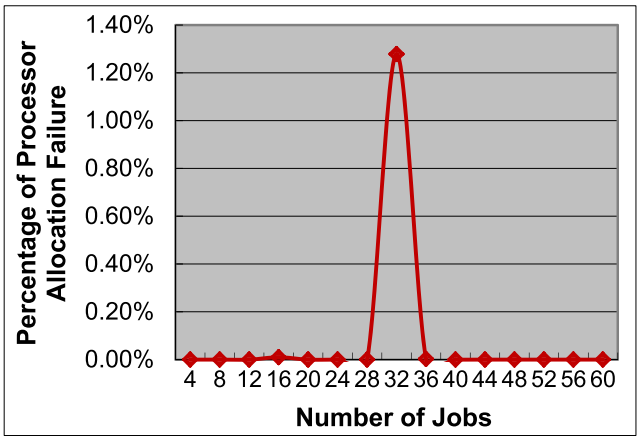

FIGURE 17. Percentage of the job sets that cannot be improved due to processor allocation failure on an eight-processor platform.

We can see in Figure 16 and Figure 17 that with the exception of a few numbers on the x-axis, in most situations, almost 100% of the job sets can be successfully allocated to the processors in order to improve their schedule. When the number of jobs in each system falls into some specific interval, less than 1.4% of the job sets cannot be improved due to failures in step (4) or step (6) of Algorithm 2.

V. CONCLUSION

In this paper, we presented algorithms for improving the schedulability of cyclic executive mixed criticality systems. These algorithms provide solutions to the following issues: (1) how to allocate the workloads of jobs among processors before and after time S; (2) how to implement the executions of jobs in each frame when the schedule is being remapped; (3) how to rearrange the schedule in order to reduce each frame's length (or the length of time required to complete all jobs) to a minimum possible value.

We reduce the length of the frame by "swapping-in" the time units of HI-criticality jobs' $C_i(EX)$ before time S, whist "swapping-out" the same amount of time units of $C_i(LO)$ after S. We let the time units of HI-criticality jobs' $C_i(EX)$ share the same time intervals with $C_i(LO)$ before S, so that the value of S is not affected and all the timing restrictions from the system definition of Section 2 are guaranteed.

The presented algorithms can reduce the length of each frame toward an optimal (minimum) value which is a necessary condition for schedulability under any schedule. The length of the frame that can be reduced only depends on the finish times of $C_i(LO)$ from a few specific jobs in Γ_{sub}. We provided algorithms in order to enable such jobs' $C_i(LO)$ to complete as soon as possible so as to maximize the improvement. We also proved that, when these finish times are fixed given values, the proposed algorithms and the results given by Theorem 1 and Theorem 2 are optimal.

We evaluated the performance of the proposed algorithms by conducting experiments on a very large number of randomly generated job sets. We showed that the proposed algorithms always significantly improve the completion time of each frame in nearly all situations as long as the processor utilization of this time interval does not reach its maximum possible value. We also showed that if our algorithms cannot shorten the frame, no algorithm can further reduce the length of such a frame.

The original length of the frames is reduced by approximately 25% across all job sets in our experiments before the processor utilization of each time interval is equal to the number of processors in the system, in which case no algorithm can further improve the frames.

With the exception of a very limited number of specific situations, we found that almost 100% of the job sets in our experiments received the optimal length of completion time after the improvements.

ACKNOWLEDGMENT

The authors would like to thank Prof. S. Baruah and Prof. A. Burns for their helpful suggestions and discussions.

REFERENCES

[1] *DO-254: Design Assurance Guidance for Airborne Electronic Hardware*, Eur. Org. Civil Aviation Equip., Radio Tech. Commission Aeronaut., Washington, DC, USA, 2000.
[2] *Road Vehicles—Functional Safety*, Standard ISO 46:2011, International Organization for Standardization, 2011.
[3] *Software Considerations in Airborne Systems and Equipment Certification*, EUROCAE, RTCA, Washington, DC, USA, 2011, vol. 41.
[4] K. Agrawal, S. Baruah, and A. Burns, "Semi-clairvoyance in mixed-criticality scheduling," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 458–468.
[5] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessors," in *Proc. Real-Time Syst. Symp.*, Nov. 2008, pp. 385–394.

[6] T. P. Baker and A. Shaw, "The cyclic executive model and ada," in *Proc. Real-Time Syst. Symp.*, 1988, pp. 120–129.
[7] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Syst.*, vol. 50, no. 1, pp. 142–177, Jan. 2014.
[8] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Syst.*, vol. 25, no. 1, pp. 5–37, 2003.
[9] A. A. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 469–480.
[10] K. Bletsas and B. Andersson, "Notional processors: An approach for multiprocessor scheduling," in *Proc. 15th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2009, pp. 3–12.
[11] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages*, 4th ed. Reading, MA, USA: Addison-Wesley, 2009.
[12] A. Burns, R. I. Davis, P. Wang, and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme," *Real-Time Syst.*, vol. 48, no. 1, pp. 3–33, Jan. 2012.
[13] A. Burns and S. Baruah, "Semi-partitioned cyclic executives for mixed criticality systems," in *Proc. 3rd Int. Workshop Mixed Criticality Syst. (RTSS)*, 2015, pp. 40–45.
[14] A. Burns, T. Fleming, and S. Baruah, "Cyclic executives, multi-core platforms and mixed criticality applications," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 3–12.
[15] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, 2018, Art. no. 82.
[16] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1478–1491, Oct. 2018.
[17] J. Caplan, Z. Al-bayati, H. Zeng, and B. H. Meyer, "Mapping and scheduling mixed-criticality systems with on-demand redundancy," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 582–588, Apr. 2018.
[18] G. Chen, N. Guan, D. Liu, Q. He, K. Huang, T. Stefanov, and W. Yi, "Utilization-based scheduling of flexible mixed-criticality real-time tasks," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 543–558, Apr. 2018.
[19] H. S. Chwa, H. Baek, and J. Lee, "Necessary feasibility analysis for mixed-criticality task systems on uniprocessor," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 446–457.
[20] R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?" *IEEE Des. Test.*, vol. 33, no. 5, pp. 65–74, Oct. 2016.
[21] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Sep. 2013, pp. 17:1–17:15.
[22] N. Guan, M. Stigge, W. Yi, and G. Yu, "Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound," in *Proc. IEEE Real-Time Technol. Appl. Symp.*, Apr. 2010, pp. 165–174.
[23] Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong, "Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2018, pp. 373–383.
[24] S. Kato and N. Yamasaki, "Real-time scheduling with task splitting on multiprocessors," in *Proc. 13th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2007, pp. 441–450.
[25] K. Lakshmanan, R. Rajkumar, and J. Lehoczky, "Partitioned fixed-priority preemptive scheduling for multi-core processors," in *Proc. 21st Euromicro Conf. Real-Time Syst.*, Jul. 2009, pp. 239–248.
[26] P. A. Laplante and S. J. Ovaska, *Real-Time Systems Design and Analysis: Tools for the Practitioner*, 4th ed. Hoboken, NJ, USA: Wiley, 2011.
[27] D. Liu, N. Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Scheduling analysis of imprecise mixed-criticality real-time tasks," *IEEE Trans. Comput.*, vol. 67, no. 7, pp. 975–991, Jul. 2018.
[28] R. McNaughton, "Scheduling with deadlines and loss functions," *Manage. Sci.*, vol. 6, no. 1, pp. 1–12, Oct. 1959.
[29] R. Medina, E. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality DAGs on multi-core architectures," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2018, pp. 254–264.
[30] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.

FENGXIANG ZHANG (Member, IEEE) received the Ph.D. degree in computer science from the University of York, U.K., in 2009. He is currently an Associate Professor with the School of Computer and Information Science, Southwest University, China. His research interests include scheduling analysis of real-time and embedded systems.