Mixed-Criticality Scheduling for Parallel Real-Time Tasks with Resource Reclamation

Qingqiang He[1], Nan Guan[2], Xu Jiang[3]

[1]The Hong Kong Polytechnic University, Hong Kong SAR
[2]City University of Hong Kong, Hong Kong SAR
[3]Northeastern University, China

***Abstract*—This paper considers the mixed-criticality scheduling of parallel real-time tasks. With the purpose of guaranteeing the deadline for hard real-time tasks and reclaiming computing resources for soft real-time tasks, we propose an approach by online monitoring the execution of hard real-time tasks and adjusting the allocated number of cores dynamically. To achieve this, we present a concept called *allocation vector*, which can serve as the interface between hard real-time tasks and soft real-time tasks: for hard real-time tasks, we derive a schedulability test under the interface; for soft real-time tasks, we discuss the design principle of how to determine the interface to reclaim computing resources as much as possible. We demonstrate the usefulness of the interface and the effectiveness of the proposed approach through examples.**

I. Introduction

This paper considers the mixed-criticality scheduling for parallel real-time tasks under the federated scheduling paradigm. The parallel real-time task is characterized by the volume and the length. The volume is the total workload in this task, and the length is the workload of the longest path in this task. In federated scheduling [1], each heavy task (tasks with the volume larger than its deadline) is assigned and executed exclusively on a set of cores. Therefore, we can restrict our attention to the scheduling of one parallel real-time task on a set of cores.

The federated scheduling suffers from the resource-wasting problem [2], [3] due to the pessimism within its analysis techniques and the overly conservative characterization of parallel real-time tasks. To address the resource-wasting within the scheduling of one parallel real-time task, we propose a mixed-criticality approach by online monitoring the execution of the hard parallel real-time task, and dynamically adjusting the allocated number of cores. Our approach can guarantee the deadline of hard real-time tasks and reclaim computing resources for soft real-time tasks at the same time. To achieve this, we present a concept called *allocation vector*, which can serve as the interface between hard real-time tasks and soft real-time tasks: for hard real-time tasks, we derive a schedulability test under the interface; for soft real-time tasks, we discuss the design principle of how to determine the interface to reclaim computing resources as much as possible.

Our approach only relies on the volume and the length of parallel real-time tasks, not requiring the detailed structure of the parallel task. The usefulness of the introduced interface and the effectiveness of the proposed approach are demonstrated through examples.

II. Related Work

Two closely related works to this paper are [4], [5].

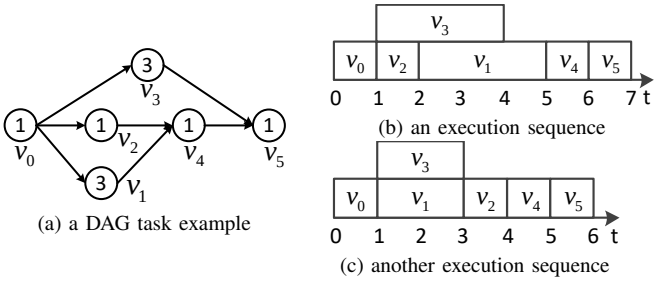
In [4], Agrawal et al. proposed a task model to represent parallel real-time tasks using two pairs of volume and length with different levels of assurance. One pair of volume and length is very conservative and therefore trusted to a very high level of assurance; the other is more representative of the typical execution behavior. The task model enables the scheduling algorithm to dynamically adjust the number of cores assigned to an individual task during the execution of the task. The adjustment of the number of cores is only based on the task model which is obtained by offline profiling the parallel real-time task, so [4] does not utilize the runtime information to reclaim computing resources for soft real-time tasks.

In [5], Baruah extended the method in [4] by combining the worst-case characterizations (i.e., the volume and length) and experimental profiling of execution behavior of parallel real-time tasks. Baruah motivated the work using conditional parallel real-time tasks, since the existence of conditional constructs makes the execution behavior of parallel real-time tasks more complex and the worst-case characterizations more pessimistic.

III. System Model

A. Task Model

A sporadic parallel real-time task is specified as a tuple (G, D, T), where G is the DAG task model, D is the relative deadline and T is the period. We consider constrained deadline, i.e., $D \leq T$. The DAG task model is a directed acyclic graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. Each vertex $v \in V$ represents a piece of sequential workload with worst-case execution time (WCET) $c(v)$. An edge $(v_i, v_j) \in E$ represents the precedence relation between v_i and v_j, i.e., v_j can only start its execution after v_i finishes its execution. A vertex with no incoming (outgoing) edges is called a *source vertex* (*sink vertex*). Without loss of generality, we assume that G has exactly one source (denoted as v_{src}) and one sink (denoted as v_{snk}). In case G has multiple



(a) a DAG task example

(b) an execution sequence

(c) another execution sequence

Fig. 1. An illustrative example.

source/sink vertices, a dummy source/sink vertex with zero WCET can be added to comply with our assumption.

A *path* λ is a set of vertices $(\pi_0, \cdots, \pi_k)$ such that $\forall i \in [0, k-1]$, $(\pi_i, \pi_{i+1}) \in E$. The length of a path λ is defined as $len(\lambda) \coloneqq \sum_{\pi_i \in \lambda} c(\pi_i)$. A *complete path* is a path $(\pi_0, \cdots, \pi_k)$ such that $\pi_0 = v_{src}$ and $\pi_k = v_{snk}$, i.e., a complete path is a path starting from the single source vertex and ending at the single sink vertex. The *longest path* is a complete path with largest $len(\lambda)$ in G. If there is an edge $(u, v) \in E$, u is a *predecessor* of v, and v is a *successor* of u. If there is a path in G from u to v, u is an *ancestor* of v and v is a *descendant* of u. We use $pred(v)$, $succ(v)$, $ance(v)$ and $desc(v)$ to denote the set of predecessors, successors, ancestors and descendants of v, respectively.

Example 1. *Fig. 1a shows a parallel real-time task G where the number inside vertices is the WCET. The deadline and the period $D = T = 7$. v_0, v_5 are the source vertex and the sink vertex, respectively. The longest path is $\lambda = (v_0, v_1, v_4, v_5)$ and $len(\lambda) = 6$. For vertex v_4, $pred(v_4) = \{v_1, v_2\}$, $succ(v_4) = \{v_5\}$, $ance(v_4) = \{v_0, v_1, v_2\}$, $desc(v_4) = \{v_5\}$.*

B. Runtime Behavior

The parallel task G executes on a multi-core platform with identical cores. A vertex v is *eligible* if all of its predecessors have finished, thus v can be immediately executed if there are available cores. The parallel task G is scheduled by any algorithm that satisfies the *work-conserving* property, i.e., an eligible vertex must be executed if there are available cores.

At runtime, vertices of G execute at certain time points on certain cores under the decision of the scheduling algorithm. An *execution sequence* ε of G describes which vertex executes on which core at every time point. For example, two execution sequences of the task in Fig. 1a are shown in Fig. 1b and Fig. 1c. For a vertex v, the *start time* $s(v)$ and *finish time* $f(v)$ are the time point when v first starts its execution and completes its execution, respectively. Note that $s(v)$ and $f(v)$ are specific to a certain execution sequence ε, but we do not include ε in their notations for simplicity. Without loss of generality, we assume the source vertex of G starts execution at time 0, so the *response time* R of G in an execution sequence equals $f(v_{snk})$.

IV. Motivation

This section discusses the scheduling algorithm of a parallel real-time task in federated scheduling, which motivates this work.

For a DAG task $G = (V, E)$, two important characterizations of G are the *volume* and the *length*. The volume (denoted as $vol(G)$) is the total workload in this task and is defined as $vol(G) \coloneqq \sum_{v \in V} c(v)$. The length (denoted as $len(G)$) is the length of the longest path in this task. For example, for the task G in Fig. 1a, $vol(G) = 10$ and $len(G) = 6$. The volume can be measured by executing the task in a platform with one core, and the length can be measured by executing the task in a platform with a sufficiently large number (bounded by the number of vertices in this task) of cores [4].

In [6], Graham proposed a well-known response time bound using the volume and the length of a DAG as follows. The response time R of DAG task G scheduled by work-conserving scheduling on m cores is bounded by (1).

$$R \leq len(G) + \frac{vol(G) - len(G)}{m} \tag{1}$$

Therefore, in federated scheduling, the number of cores m allocated to a DAG G can be computed by (2).

$$m = \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \tag{2}$$

Equation (2) computes the minimum number of cores m such that the response time bound in (1) is no larger than the deadline D.

For a DAG task, the computing resources allocated according to (1) and (2) exhibit several types of pessimism and cause a large amount of resources being wasted, as summarized in the following.

- **Analysis Pessimism.** The bound in (1) is derived by constructing an artificial scenario where vertices not in the longest path do not execute in parallel with the execution of the longest path. However, in real execution, many vertices not in the longest path actually can execute in parallel with the longest path. As observed in [3], this type of pessimism may cause the portion of wasted computing resources arbitrarily close to 100%.
- **Execution Pessimism.** Parameters used in (2), such as $vol(G)$, $len(G)$, are based on the worst case execution time. To comply with the hard real-time requirements, these worst case execution times can be overly pessimistic [7], [8], and the actual execution time can be far less than the WCET, leading to severe resource-wasting during execution.

The analysis pessimism can be partially addressed through improved offline analysis. For example, the technique of intra-task priority assignment can be employed to improve system schedulability [9]–[11]. Under this technique, priorities are assigned to the vertices of a DAG task to control the execution order of vertices and runtime behavior of the task. However, the execution pessimism cannot be mitigated through offline

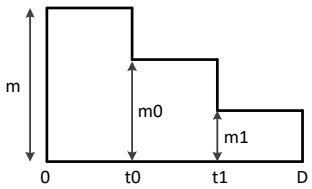


Fig. 2. An illustration of the scheduling for our approach.

analysis, since the required information (such as the actual execution time of vertices) is only available during runtime.

In this paper, we propose an approach to address both two types of pessimism by online monitoring the execution of parallel tasks and adjusting the allocated number of cores dynamically with the target of both satisfying the hard real-time deadline and reclaiming computing resources for soft real-time tasks.

V. Our Approach with Resource Reclamation

In this approach, during the execution of the parallel real-time task, we collect information regarding the execution of the hard real-time task and gradually reduce the allocated number of cores to reclaim computing resources for soft real-time tasks.

A. The Scheduling Algorithm

Definition 1 (Allocation Vector)**.** *For a parallel real-time task (G, D, T), the allocation vector Φ is a set of time points $\{t_0, \cdots, t_k\}$ $(k \geq 0)$ satisfying all of the following conditions.*

1) $\forall i \in [0, k]$, $0 \leq t_i < D$.
2) $\forall i, j \in [0, k]$ *and* $i < j$, $t_i < t_j$.

Given a parallel real-time task (G, D, T) and the allocation vector $\Phi = \{t_0, \cdots, t_k\}$, the scheduling starts at time 0 with the number of cores m computed by (2). During the scheduling, two types of information are monitored.

1) $w(t)$: the volume of the workload executed from time point 0 to time point t.
2) $l(t)$: the cumulative length of time intervals during which some core is idle from time point 0 to time point t.

At each time point t_i, if G does not complete its execution, we adjust the allocated number of cores to m_i. The conditions for m_i will be derived in Section V-B. The scheduling is illustrated in Fig. 2.

B. Schedulability Test for Hard Real-Time Tasks

Definition 2 (Critical Path [9])**.** *The critical path $\lambda^* = (\pi_0, \cdots, \pi_k)$ of an execution sequence is a complete path satisfying the following property.*

$$\forall \pi_i \in \lambda^* \setminus \{\pi_0\} : f(\pi_{i-1}) = \max_{u \in pred(\pi_i)} \{f(u)\} \tag{3}$$

The critical path is specific to an execution sequence of the DAG task G. The critical path of G in an execution sequence is not necessarily the longest path of G.

Example 2. *For the execution sequence in Fig. 1b, the critical path of G is (v_0, v_1, v_4, v_5). In Fig. 1c, the critical path of G is (v_0, v_2, v_4, v_5), which is not the longest path of G. In Fig. 3b, the critical path of G is also (v_0, v_1, v_4, v_5).*

Lemma 1. *In an execution sequence under work-conserving scheduling, when the critical path is not executing, all cores are busy.*

Proof. Suppose that the critical path of this execution sequence is $\lambda^* = (\pi_0, \cdots, \pi_k)$. $\forall i \in (0, k]$, by Definition 2, π_{i-1} is with the maximum finish time among all the predecessors of π_i. This means that when π_{i-1} completes its execution, all predecessors of π_i have completed execution. Therefore, π_i is eligible at $f(\pi_{i-1})$. If some core is idle in $[f(\pi_{i-1}), s(\pi_i)]$, it contradicts the fact that the scheduling is work-conserving. □

Lemma 2. *In an execution sequence under work-conserving scheduling, when some core is idle, the critical path is executing.*

Proof. Lemma 2 is the contrapositive of Lemma 1. □

Theorem 1. *At each time point t_i ($i \in [0, k]$), if the allocated number of cores m_i is computed by (4) and (5), then the parallel real-time task G is schedulable under work-conserving scheduling with the allocation vector $\Phi = \{t_0, \cdots, t_k\}$.*

if $vol(G) - w(t_i) \leq len(G) - l(t_i)$, then

$$m_i = 1 \tag{4}$$

else

$$m_i = \left\lceil \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \right\rceil \tag{5}$$

Proof. Let ε be the execution sequence under analysis of G. At time point t_i, we focus on the remaining graph G_i of G (i.e, the part of G that has not been executed until t_i). We denote the critical path of ε as λ^*. Since l_i is the cumulative length of time intervals before t_i where some core is idle, by Lemma 2, λ^* is executing in these time intervals. Therefore, the length of λ^* in G_i (i.e., the length of λ^* executing after t_i) is at least $len(\lambda^*) - l(t_i)$, which is bounded by

$$len(G) - l(t_i)$$

And the volume of G_i is bounded by

$$vol(G) - w(t_i)$$

By (1), the response time of G_i is bounded by

$$len(G) - l(t_i) + \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{m_i}$$

The new deadline of G_i is $D - t_i$. Let

$$len(G) - l(t_i) + \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{m_i} \leq D - t_i$$

which means (5). □

Note that the volume $vol(G)$ and length $len(G)$ in Theorem 1 are profiled and determined offline. We do not need to monitor these two parameters online.

Corollary 1. *The schedulability test in Theorem 1 dominates the test in [1] (shown in (1) and (2)) in the sense that the computing resource allocated by Theorem 1 is no larger than that of [1].*

Proof. It is sufficient to show that $\forall i \in [0, k]$, the m_i in (5) is no larger than the m in (2), i.e.,

$$\left\lceil \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \right\rceil \leq \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \quad (6)$$

Suppose $t_0 = 0$, we have $w(0) = 0$, $l(0) = 0$, so (6) holds trivially. Therefore, to prove (6), it suffices to show that $\forall i \in [0, k)$, $m_{i+1} \leq m_i$, i.e.,

$$\frac{vol(G) - w(t_{i+1}) - len(G) + l(t_{i+1})}{D - t_{i+1} - len(G) + l(t_{i+1})} \leq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (7)$$

We define

$$\Delta(t) \coloneqq t_{i+1} - t_i$$

$$\Delta(w) \coloneqq w(t_{i+1}) - w(t_i)$$

$$\Delta(l) \coloneqq l(t_{i+1}) - l(t_i)$$

Equation (7) can be rewritten as (8).

$$\frac{vol(G) - w(t_i) - len(G) + l(t_i) - (\Delta(w) - \Delta(l))}{D - t_i - len(G) + l(t_i) - (\Delta(t) - \Delta(l))} \leq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (8)$$

Be aware of the following statement: for $0 < x_1 < x_2$, $0 < y_1 < y_2$,

$$\frac{x_1}{y_1} \geq \frac{x_2}{y_2} \Rightarrow \frac{x_2 - x_1}{y_2 - y_1} \leq \frac{x_2}{y_2}$$

Therefore, to prove (8), it suffices to show that

$$\frac{\Delta(w) - \Delta(l)}{\Delta(t) - \Delta(l)} \geq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (9)$$

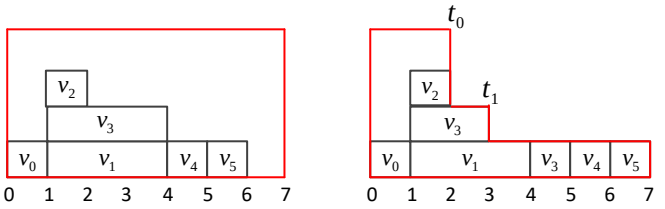
The length of time interval $[t_i, t_{i+1}]$ is $\Delta(t)$. During time interval $[t_i, t_{i+1}]$, the allocated number of cores is m_i. The volume of the workload executed in $[t_i, t_{i+1}]$ is $\Delta(w)$. By the definitions of $l(t)$ and $\Delta(l)$, $\Delta(l)$ is the cumulative length of time intervals during which some core is idle in $[t_i, t_{i+1}]$. Therefore, we have

$$\Delta(w) \geq m_i(\Delta(t) - \Delta(l)) + \Delta(l) \quad (10)$$

which means

$$\frac{\Delta(w) - \Delta(l)}{\Delta(t) - \Delta(l)} \geq m_i$$

Therefore, (9) holds, which means that (6) holds. □



(a) federated scheduling in [1] (b) the proposed approach

Fig. 3. Compare the allocated computing resources between the original federated scheduling and our approach.

C. Design Principle for Soft Real-Time Tasks

This subsection discusses that for a parallel task, how to determine the allocation vector. The objective is that at each time point t_{i+1}, we want to reduce the number of cores m_{i+1} compared to m_i. By Corollary 1, we know that this reduction of cores lies in (10). Therefore, during time interval $[t_i, t_{i+1}]$, the monitoring procedure should observe the type of execution satisfying both of the following conditions.

1) During the execution, at least one core is idle.
2) During the execution, more than one core are busy.

The more this type of execution, the more the number of cores can be reduced. With this guideline, the allocation vector can be determined by offline profiling or dynamically determined during execution.

D. An Example

This subsection provides an example to illustrate the usefulness of the allocation vector interface and the effectiveness of the proposed approach. For the parallel real-time task G in Fig. 1a, suppose that the deadline $D = 7$. The volume $vol(G) = 10$ and the length $len(G) = 6$. Fig. 3 shows the possible execution sequences and the allocated computing resources (circled by red rectangles) under the original federated scheduling in [1] and our approach.

In Fig. 3a, by (2), the allocated number of cores $m = (10-6)/(7-6) = 4$. So the total allocated computing resources (the area of the red rectangle) are $m \times D = 4 \times 7 = 28$. In Fig. 3b, suppose that the allocation vector is $\Phi = \{t_0 = 2, t_1 = 3\}$. At time point 0, same as the original federated scheduling, the allocated number of cores $m = 4$. At time point $t_0 = 2$, the monitored information are $w(t_0) = 4$, $l(t_0) = 2$. $vol(G) - w(t_0) = 6$, $len(G) - l(t_0) = 4$ and $D - t_0 = 5$. By (5), the adjusted number of cores $m_0 = (6-4)/(5-4) = 2$. At time point $t_1 = 3$, the monitored information are $w(t_1) = 6$, $l(t_1) = 2$. $vol(G) - w(t_1) = 4$, $len(G) - l(t_1) = 4$ and $D - t_1 = 4$. Since $vol(G) - w(t_1) \leq len(G) - l(t_1)$, by (4), the adjusted number of cores $m_1 = 1$. So the total allocated computing resources are $4 \times 2 + 2 \times 1 + 1 \times 4 = 14$. Therefore, in this example, our approach reclaims $(28 - 14)/28 = 50\%$ computing resources for soft real-time tasks and guarantees that the hard real-time task meets its deadline.

VI. Conclusion

In this paper, to address the analysis and execution pessimism that lead to the resource-wasting problem in federated scheduling, we propose a mixed-criticality approach by online monitoring the execution of hard parallel real-time tasks, and dynamically adjusting the allocated number of cores to reclaim computing resources for soft real-time tasks. We give an example that illustrates the effectiveness of the proposed approach.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive and insightful comments.

References

[1] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014, pp. 85–96.

[2] N. Ueter, G. Von Der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 482–494.

[3] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and W. Yi, "Virtually-federated scheduling of parallel real-time tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 482–494.

[4] K. Agrawal and S. Baruah, "A measurement-based model for parallel real-time tasks," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[5] S. Baruah, "Resource-efficient execution of conditional parallel real-time tasks," in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

[6] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[7] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE, 2001, pp. 215–224.

[8] G. Bernat, A. Colin, and S. M. Petters, "Wcet analysis of probabilistic hard real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE, 2002, pp. 279–288.

[9] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2283–2295, 2019.

[10] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *IEEE Real-Time Systems Symposium*. IEEE, 2020.

[11] Q. He, M. Lv, and N. Guan, "Response time bounds for dag tasks with arbitrary intra-task priority assignment," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.