Resource-Aware Scheduling for Dependable Multicore Real-Time Systems: Utilization Bound and Partitioning Algorithm

Jian-Jun Han, Zhenjiang Wang, Sunlu Gong, Tianpeng Miao, and Laurence T. Yang

Abstract—As the computing devices and software executions are susceptible to manifold faults, fault tolerance has been an important research topic in safety-critical real-time systems. Moreover, multicore processors have recently emerged as prevailing computing engines for modern embedded systems. However, there exists rather rare work on the fault-tolerant scheduling of real-time tasks executing on multicores with shared resources, where the task synchronization originated from resource access contention may significantly degrade the schedulability of task system. With the focus on the partitioned-EDF scheduler with the MSRP (Multiprocessor Stack Resource Policy) protocol and primary/backup recovery mechanism, we first investigate a utilization bound and then identify its *anomaly* where the bound may decrease when more cores are deployed. Next, following the insights gained by the analysis of the bound, we propose a reliability and synchronization aware task partitioning algorithm (RSA-TPA) together with an efficient version to implement the *joint management* of task synchronization and system reliability, where several *resource-oriented* heuristics are developed to improve both the schedulability performance and workload balancing. The extensive simulation results show that the RSA-TPA schemes can obtain higher acceptance ratio (e.g., 60 percent more) and generate more balanced partitions, when compared to the existing schemes that consider either reliability management or task synchronization. Finally, with the different fault arrival rates being considered, the actual implementation in Linux kernel further demonstrates the *applicability* of RSA-TPA that has lower run-time overhead (e.g., 20 percent less) in comparison with other mapping algorithms.

Index Terms—Multicore processor, real-time systems, shared resources, reliability, primary/backup, partitioned scheduling

1 Introduction

Real-time systems have been recently exploited in a broad spectrum of application areas with timeliness requirements, such as industrial automation, embedded control and avionics domains. As the computing systems are susceptible to different kinds of faults caused by run-time errors, the safety-critical activities executing on embedded real-time systems usually demand fault-tolerant techniques to guarantee the system reliability. Faults can be categorized into two types: *permanent* and *transient* faults [1], [2]. In general, permanent faults stem from the persistent hardware component failure, while the transient faults are generated by the electromagnetic interference (e.g., single event upsets) or software errors that can result in incorrect computing results [3]. Moreover, with the goal of satisfying the performance of applications while enabling power efficiency, multicore processors have become *de facto* computing engines for embedded real-time systems. Aiming at the reliability management, task replication has become one of the commonly adopted mechanisms against permanent faults [4] and transient faults [4], [5], [6] in the real-time systems, where two or more copies (replicas) of the same task simultaneously run on different processors and at least one copy must finish executing by its deadline once the faults happen.

On the other hand, for tasks executing on shared resource multicores, a task may need to exclusively access shared resources (such as shared data objects). Such resource access competition can bring about the problem of *task synchronization* and can lead to *priority inversion* for real-time applications [7]. That is, when a low-priority task is holding and accessing a resource exclusively, a high-priority task that needs access to the same resource is blocked and has to wait until the low-priority one releases the resource. Therefore, the priority inversion arisen from task synchronization can affect the executions of those high-priority tasks and thus can give rise to decreased schedulability of task system. To address the problem of task synchronization, several resource access protocols have been explored for uniprocessors [7], [8], where unbounded priority inversions caused by medium-priority tasks are avoided. Following the same criteria, the protocols designed towards single-processor systems have been extended and developed for multiprocessors/multicores [9], [10].

• *The authors are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: {jasonhan, wangzhenjiang, gsl, mtp027}@hust.edu.cn, ltyang@stfx.ca.*

Manuscript received 10 May 2018; revised 4 June 2019; accepted 29 June 2019. Date of publication 2 July 2019; date of current version 8 Nov. 2019. (Corresponding authors: Sunlu Gong and Tianpeng Miao.)
Recommended for acceptance by P. Sadayappan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2019.2926455

An empirical study pointed out that the race for data structures would increase the execution time of a task up to 30 percent on a 16-core system due to accessing shared resources [11]. More importantly, with increasing scaling levels in CMOS technologies, more and more transistors are deployed per unit area. Thus, the susceptibility of computing systems to different faults correspondingly increases and can result in the reliability degradation [12]. This is considered as one of the great challenges for the next generation energy-efficient system designs [13]. Therefore, the *joint management* of system dependability and task synchronization becomes imperative for the safety-critical real-time activities running on multicores with shared resources. Although the real-time scheduling under the requirements of task synchronization [14], [15], [16] or dependability maintenance [4], [6] has been extensively and respectively studied over the past two decades, there exists rather limited work on the scheduling of real-time tasks that access shared resources on multicores under the ever-increasing demands for reliability management.

Generally, there are two approaches to the problem of multiprocessor/multicore scheduling: *partitioned* and *global* scheduling. The previous measurement results for the scheduling of real-time tasks running on multicores showed that when compared to global scheduling, the partitioned based scheduling can usually have higher practical viability due to much lower online overhead [17]. Notice that inappropriate partitioning of tasks (that access shared resources) upon cores can cause larger synchronization overheads of tasks (and inflated system utilization) and thus can affect the schedulability of tasks. Furthermore, the redundancy of tasks (e.g., replicas) needed for the reliability maintenance inevitably results in more blocking interferences among tasks, and thus may significantly decrease the system schedulability.

To address this problem, for periodic real-time tasks running on multicores with shared resources, by incorporating the primary/backup mechanism to tolerate a single permanent fault or one transient fault for each task, we first develop a system utilization bound and verify its characteristics. Then, we propose a Reliability and Synchronization Aware Task Partitioning Algorithm (*RSA-TPA*), which explicitly takes the varying blocking overheads of tasks during task assignments into account. In view of its efficient per-core based feasibility test, this work focuses on the partitioned EDF (Earliest-Deadline-First) scheduling. In addition, we adopt the MSRP (Multiprocessor Stack Resource Policy) resource access protocol [9] that can effectively tackle the task synchronization in real-life applications, such as power-train controller for automotive activities [9] and generic avionics platform task set for avionics applications [18]. In particular, the contributions of this work are summarized as follows.

- Based on the MSRP protocol and primary/backup model, we derive the first reliability and resource aware *system utilization bound* and identify its *non-monotonicity*, where the bound may decrease with more deployed cores due to increased blocking interferences among tasks;
- By exploiting the *varying limits* on the synchronization overheads of primary and backup tasks during task-to-core allocations, we develop a task mapping scheme RSA-TPA, where several resource-guided heuristics are explored for better schedulability and balanced partitions;
- We further investigate a low-complexity task assignment heuristic with the objective of *dual minimization*, which can achieve the schedulability performance approximate to the original probe based RSA-TPA scheme;
- The measurement results from the *actual implementation* in Linux kernel reveal that the RSA-TPA scheme is quite practical due to low online overhead that is benefited from its resource-aware workload balancing policy.

The remainder of this paper is organized as follows. The review of the related studies is given in Section 2. Section 3 presents the system, task and reliability models and some preliminaries about the protocol MSRP. The system utilization bound is verified and analyzed in Section 4. The mapping scheme RSA-TPA with several resource-aware heuristics is proposed in Section 5. The evaluation results from simulations and measurements are given and discussed in Section 6. Section 7 concludes the paper.

2 Closely Related Work

With the goal to tolerate a single permanent fault, Ejlali et al. studied a Standby-Sparing (SS) scheme for aperiodic real-time tasks on a dual-processor system [19]. Following the same principle of separating tasks on different processors, Haque et al. extended SS scheme to a periodic task system, where primary and backup tasks are scheduled by EDF and EDL (Earliest Deadline as Late as possible) respectively on their dedicated processors [20]. A recent work on the hybrid framework for resource allocation that considers permanent and transient faults was reported in [21].

In general, there are two approaches to tolerating transient faults in a real-time application: checkpoints and primary/backup mechanism. Melhem et al. derived the optimal number of checkpoints to save energy for a duplex system [22]. The other approach is primary/backup (PB) model, where each task has primary and backup replicas (copies) [23]. Based on the PB technique, Al-Omari et al. proposed a PB overloading technique, where primary tasks can overlap with backups of other tasks [24]. Recently, Guo et al. developed several general standby-sparing and POED (Preference-Oriented Earliest Deadline) based schemes to improve the energy efficiency [4]. Moreover, several recent studies on the dependability and energy aware real-time scheduling were investigated in [6], [25]. However, these fault-tolerant scheduling algorithms *did not* consider the resource access competition in the context of multicore real-time systems.

To address the problem of task synchronization, several resource access protocols have been explored. For single-processor systems, Sha et al. proposed a Priority Ceiling Protocol (PCP) [7] for fixed priority scheduling (e.g., Rate-Monotonic (RM)) and Baker presented a Stack Resource Policy (SRP) [8] for dynamic priority scheduling (e.g., EDF). The corresponding MPCP (Multiprocessor PCP) [10] and MSRP (Multiprocessor SRP) [9] have been extended and developed for multiprocessor/multicore systems. The Flexible

Multiprocessor Locking Protocol (FMLP) [11] and an optimal locking protocol (OMLP) [26] were also studied. Later, the Multiprocessor resource sharing Protocol (MrsP) was developed for partitioned fixed-priority scheduling [27].

For the partitioned scheduling of real-time tasks that access shared resource, based on EDF and MSRP, Han et al. proposed a Synchronization-Aware Worst-Fit Decreasing (SA-WFD) task mapping algorithm that tries to allocate tasks needing to access the same resources to a single core for better schedulability. Recently, based on the MSRP protocol, Tsai et al. presented a triple speed algorithm to enable energy awareness under the constraints of task synchronization, while neglecting the mapping heuristics for better schedulability [15]. Wieder et al. [16] presented a greedy-slacker algorithm for MSRP (GS-MSRP) to assign a task onto a processor that the minimum slack of all its tasks is maximized by using Audsley's optimal priority assignment [28]. Recently, Huang et al. studied the resource aware partitioned scheduling and derived a speedup factor for multiprocessors, where the access to shared resources must execute on dedicated processor [29]. Here, the resource access pattern bears some resemblance to the inter-process communication protocol, which is different from that exploited in this work (i.e., MSRP). Georg et al. [30] further improved the resource-oriented partitioned scheduling as reported in [29]. A survey of task synchronization for real-time scheduling can be found in [31]. Nonetheless, these mapping schemes *did not* take the demands for system reliability into consideration.

3 System Models and Preliminaries

We first introduce the system, task and fault recovery models, followed by the brief review of MSRP protocol and schedulability condition for partitioned-EDF and MSRP with the PB model.

3.1 System and Task Models

We consider a homogeneous multicore system that consists of M identical processor cores ($\{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$) having the same function and capability. A set of N implicit-deadline real-time (primary) tasks $\Psi^P = \{\tau_1, \ldots, \tau_N\}$ execute on the system with their initial arrivals at time 0. With the focus on the implicit deadlines, each task τ_i has a period (relative deadline) p_i. That is, the jth instance (or job) of task τ_i arrives at time $(j-1) \cdot p_i$ and must complete executing by its absolute deadline $j \cdot p_i$. The worst-case execution time (WCET) of task τ_i is denoted as c_i.

This work considers *logical* shared resources (e.g., shared data objects or global variables in the program codes) [9], [10], [16], [29], [32], [33], instead of those *hardware* resources (e.g., bus and network) that are beyond the scope of this paper and can be tackled by their corresponding schemes [34], [35]. The extended discussions for real-time scheduling under the requirements of hardware resource contention are available in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TPDS.2019.2926455.

The system has a set of R resources $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$, which can be shared by all tasks and are protected within their critical sections for data integrity. The access to any resource by a task is exclusive and non-preemptable. That is, a resource can be held and accessed by only *one* task at any time, and the low-priority task that is accessing a resource cannot be preempted by any high-priority task. We focus on *non-nested critical sections* as the nested ones occur infrequently in practice and can be dealt with by group locks [11]. Therefore, at any time, a task is not permitted to request for a resource while holding another one. The number of critical sections in task τ_i is denoted as n_i and the xth critical section is denoted as $z_{i,x}$ ($x = 1, \ldots, n_i$), where task τ_i spends the maximum time $c_{i,x}$ accessing one of its resource $r_{i,x}$ ($\in \mathcal{R}$).

3.2 Fault and Recovery Models

When operating a real-time system, the applications are usually vulnerable to various run-time faults, such as hardware failure and electromagnetic interference. Although the permanent fault happens rather rarely in real-life applications, it is imperative for a safety-critical system to make provisions for the tolerance of permanent fault to satisfy the reliability demands.

To guarantee the deadlines of tasks and meet different reliability targets, the *overloading* technique has been utilized in the fault-tolerant real-time scheduling [4], where multiple task copies execute in parallel and can overlap with their time intervals during executions. Notice that at least one copy of every task must complete before deadline even in the presence of faults [6]. This is quite different form those fault recovery schemes designed for non real-time applications [36], where the backup tasks are normally invoked once the primary task failed.

With the objective of tolerating permanent and transient faults for the reliability management, the *Primary/Backup* (*PB*) based task replication mechanism is employed in this work. That is, for any task τ_i, there exists one backup task τ_i^B with the same timing and synchronization parameters as its primary (e.g., c_i, p_i and $c_{i,x}$ ($1 \leq x \leq n_i$)). Therefore, in addition to the primary task set Ψ^P, there is a backup task set Ψ^B ($\{\tau_1^B, \ldots, \tau_N^B\}$) concurrently running on the system and the task system under consideration is denoted as $\Psi = \Psi^P \cup \Psi^B$.

Instead of deploying dedicated cores against permanent faults, for any task, its primary and backup only need to be mapped onto *different* cores in this work. For any task, its any primary/backup instance can be safely cancelled as soon as the other one finishes successfully. Therefore, in addition to *a single permanent fault*, here *one transient fault* for every task instance (in the absence of permanent fault) can also be tolerated based on the PB mechanism [4]. With the focus on the partitioned scheduling, the preemptive EDF scheduler is deployed on each core $\mathcal{P}_m$ ($1 \leq m \leq M$) and it schedules all its primary and backup tasks in the subset Ψ_m. Thus, there is $\Psi = \cup_{m=1}^{M} \Psi_m$. Based on the PB model, for any task on any core, there only exists a single primary/backup copy. Then, for any primary/backup task τ_i on a specific core, we simply use τ_i to represent it for convenience. Moreover, as there exists at most one active job for any primary/backup task at any time, we use *task* and *job* exchangeably hereafter unless specified otherwise.

For permanent faults, based on failure-stop model, we assume that a faulty processor core can be detected by other

working cores at the earliest finish time of a task [2]. In addition, the software errors caused by transient faults can be detected at the end of execution of a task via consistency checks (e.g., parity or signature checks) [2]. Another solution to the detection of transient faults, which can affect the core hardware logic, is using error-correcting codes (ECCs) and memory interleaving method to provide strong protection against such faults [12]. Normally, it is assumed that the fault detection components are free of faults (or they may adopt internal modular redundancy to instantaneously mask faults) [1], [2]. As in most of existing studies concerning the PB technique based fault tolerance, we assume that the detection mechanisms for faults are available in the system and the detection overheads have been incorporated into the WCETs of tasks [4], [6].

3.3 Resource Access Protocol MSRP

We first briefly review the principles of MSRP protocol based on spin-lock scheme [9] and its basic rules are summarized below.

- When a task τ_i on core $\mathcal{P}_m$ issues a request for a resource $\mathcal{R}_a$, if resource $\mathcal{R}_a$ is free, task τ_i locks and then accesses it non-preemptively; otherwise, if resource $\mathcal{R}_a$ is currently occupied and accessed by another task from a different core, task τ_i is appended to the FIFO queue of resource $\mathcal{R}_a$ and the core $\mathcal{P}_m$ enters the busy-waiting state;
- Once task τ_i completes accessing resource $\mathcal{R}_a$, it unlocks resource $\mathcal{R}_a$ and becomes preemptable again. If the FIFO queue of resource $\mathcal{R}_a$ is not empty (i.e., there exist tasks on other cores waiting for accessing resource $\mathcal{R}_a$), the header task is de-queued and then locks the resource.

Notice that for the partitioned scheduling of tasks that access shared resources, comparing the priority of two tasks on different cores is meaningless [11], [26]. Based on above rules for MSRP, the execution of a task τ_i on core $\mathcal{P}_m$ may be blocked due to the resource access race at two different scenarios as follows.

- When task τ_i issues access to resource $\mathcal{R}_a$ that is currently occupied by a task from a different core, it has to wait in the FIFO queue of resource $\mathcal{R}_a$ and such duration caused by inter-core synchronization interference is denoted as *global waiting time* (or *busy-waiting time*);
- When a low-priority task τ_j on the same core (i.e., $\tau_j \in \Psi_m$) is holding and accessing a resource or busy-waiting for the release of a resource, task τ_i is locally blocked and such duration is called *local blocking time*.

For periodic tasks executing on multicores that access shared resources in their non-nested critical sections and are scheduled by EDF and MSRP, we have the following properties [8], [9].

Property 1. *For any core at any time, there exists* at most one *task that is either holding a resource or busy-waiting for a resource that is currently held by a task on another core.*

Property 2. *A task can be* locally *blocked by a low-priority task on the same core* at most once.

Property 3. *The local blocking time of a task is bounded by the* longest duration *that any low-priority task on the same core busy-waits for (if applicable) and accesses one of its resources* once.

Evidently, the above properties are still applicable to the task system Ψ considered here. Note that for any task, its primary and backup may cause inter-core blocking interferences on each other as they are mapped onto different cores based on the PB model.

3.4 Feasibility Condition for Partitioned EDF Scheduling with MSRP and PB

For the ease of discussions and presentations, some important notations used throughout this paper are first defined as follows.

- $BW_{i,x}$: the maximum amount of *global waiting time* that the primary/backup task τ_i on core $\mathcal{P}_m$ spends busy-waiting for resource $\mathcal{R}_a$ in its critical section $z_{i,x}$ (where $r_{i,x} = \mathcal{R}_a$). Based on Property 1, it can be expressed as [9], [14]

$$\begin{aligned} BW_{i,x} &= \sum_{k=1,\cdots,M}^{k\neq m} tp_k^{max}(\mathcal{R}_a), \\ tp_k^{max}(\mathcal{R}_a) &= \max\{tt_l^{max}(\mathcal{R}_a) | \forall \tau_l \in \Psi_k\}, \\ tt_l^{max}(\mathcal{R}_a) &= \max\{c_{l,y} | \forall z_{l,y} : r_{l,y} = \mathcal{R}_a\}, \end{aligned} \tag{1}$$

  where $tp_k^{max}(\mathcal{R}_a)$ denotes the maximum amount of time for any task on another core $\mathcal{P}_k$ to access resource $\mathcal{R}_a$ *once*, and $tt_l^{max}(\mathcal{R}_a)$ refers to the longest time for task τ_l to access resource $\mathcal{R}_a$ *once*;
- BW_i: the maximum amount of *total global waiting time* taken by the primary/backup task τ_i to access resources in all its critical sections, i.e., $BW_i = \sum_{x=1}^{n_i} BW_{i,x}$;
- B_i: the maximum amount of *local blocking time* that can be experienced by primary/backup task τ_i on core $\mathcal{P}_m$. Based on Properties 2 and 3 and the results from [8], [9], there is

$$B_i = \max\{BW_{j,y} + c_{j,y} | \forall z_{j,y} : \tau_j \in \Psi_m \wedge p_j > p_i\}. \tag{2}$$

Following the results as given in [9], [14], a set Ψ of implicit-deadline primary and backup tasks executing on a homogeneous shared resource multicore system is schedulable under partitioned-EDF and MSRP if, for each core $\mathcal{P}_m$ ($m = 1, \ldots, M$), there is

$$\forall \tau_i \in \Psi_m, \frac{B_i}{p_i} + \sum_{\forall \tau_j \in \Psi_m}^{p_j \leq p_i} \frac{c_j + BW_j}{p_j} \leq 1. \tag{3}$$

For a task-to-core partition Π ($\{\Psi_1, \cdots, \Psi_M\}$), the synchronization aware *core utilization* of core $\mathcal{P}_m$ is defined as

$$U(\Psi_m) = \max\left\{ \frac{B_i}{p_i} + \sum_{\forall \tau_j \in \Psi_m}^{p_j \leq p_i} \frac{c_j + BW_j}{p_j} \middle| \forall \tau_i \in \Psi_m \right\}. \tag{4}$$

Then the system utilization is defined as $U(\Pi) = \max_{\forall \mathcal{P}_m}\{U(\Psi_m)\}$. We can directly have the following proposition from Equation (3).

Proposition 1. *For a task set Ψ of periodic primary and backup tasks running on homogeneous multicores with shared resources under the partitioned-EDF scheduler with MSRP and PB, a given partition Π is feasible if there is $U(\Pi) \leq 1$.*

Problem Description. Based on above assumptions and definitions, the problem to be studied in this work is: *for a set Ψ of periodic primary and backup tasks that access shared resources on homogeneous multicores, find a*feasible and balanced*mapping Π based on Proposition 1.*

4 System Utilization Bound with Non-Monotonicity

For real-time tasks executing on multiprocessors without shared resources and reliability demands, López et al. [37] explored the utilization bounds for partitioned-EDF with different heuristics, e.g., WFD (Worst-Fit Decreasing). If the total utilization of a task set does not exceed such bounds, the partitions generated by these heuristics are guaranteed to be schedulable. Note that such bounds increase *monotonically* with more available processors. In this work, we study a system utilization bound for the task set Ψ under the partitioned EDF scheduling with MSRP, PB and WFD, and then identify its *non-monotonicity*. This phenomena typically exhibits the inherent characteristics of task synchronization in the context of multicore real-time systems as the similar scheduling anomaly of the bound for multicore mixed-criticality systems can also be observed in [32].

4.1 System Utilization Bound

From the discussions in Section 3.4, we see that the synchronization overheads of tasks are rather dependent on specific task-to-core mappings. To obtain the utilization bound for *any* feasible partitioning of tasks upon cores, we need to find the upper-bounds for such overheads and some notations are first defined below.

- p^{min}: the minimum period of primary and backup tasks in the task set Ψ, i.e., $\min\{p_i | \forall \tau_i \in \Psi\}$;
- $n^{max,cs}$: the maximum number of critical sections in a primary/backup task, i.e., $\max\{n_i | \forall \tau_i \in \Psi\}$;
- $c^{max,cs}$: the largest size of critical sections in primary and backup tasks, i.e., $\max_{\forall \tau_i \in \Psi}\{c_{i,x} | 1 \leq x \leq n_i\}$;
- BW^{ub}: the upper-bound on the total busy-waiting time taken by any primary/backup task to access all its resources. From Equation (1), there is $BW^{ub} = n^{max,cs} \cdot (M-1) \cdot c^{max,cs}$;
- B^{ub}: the upper-bound on the local blocking time of a primary/backup task. Based on Properties 2 and 3, we have $B^{ub} = c^{max,cs} + (M-1) \cdot c^{max,cs} = M \cdot c^{max,cs}$;
- α: the maximum resource-aware utilization of primary and backup tasks that is defined as $\alpha = \max\{\frac{c_i + BW^{ub}}{p_i} | \forall \tau_i \in \Psi\}$;
- γ: the upper-bound for the utilization penalty on any core due to local blocking, which is defined as $\gamma = \frac{B^{ub}}{p^{min}}$;
- σ: the synchronization overhead factor, which is defined as $\max\{\frac{B^{ub}}{p^{min}}, \frac{BW^{ub}}{p^{min}}\} = \max\{\gamma, \frac{BW^{ub}}{p^{min}}\}$;
- β: the minimum number of primary and backup tasks that can feasibly fit into one core under the EDF scheduler with the MSRP protocol and PB model.

Based on above notations and Proposition 1, with BW^{up} and B^{up} being incorporated into Equation (3), we can directly obtain the following lemma with respect to the feasibility condition for the task set Ψ under partitioned-EDF with MSRP and PB.

Lemma 1. *For a set Ψ of primary and backup tasks that access shared resources in a multicore system with M cores, a task-to-core mapping is feasible under partitioned-EDF with MSRP and PB if, for every core $\mathcal{P}_m$ ($m = 1, \ldots, M$), there is*

$$\forall \tau_i \in \Psi_m, \frac{B^{ub}}{p^{min}} + \sum_{\forall \tau_j \in \Psi_m}^{p_j \leq p_i} \frac{c_j + BW^{ub}}{p_j} \leq 1. \tag{5}$$

Based on the feasibility test for EDF with MSRP and PB as represented in Equation (5) and the definitions of α and γ, we can directly obtain the following lemma related to β.

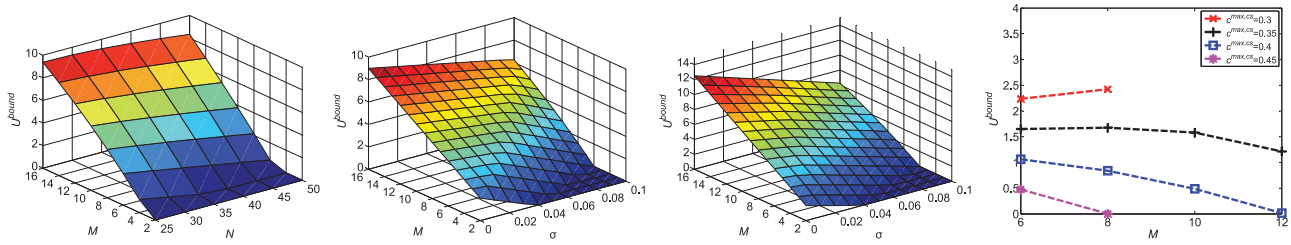
Lemma 2. *For the system and task models and reliability management considered, there is $\beta = \lfloor \frac{1-\gamma}{\alpha} \rfloor$ under partitioned-EDF with the MSRP protocol and PB recovery scheme.*

Since the current multicore processor normally has an even number of processing cores, we assume that the number of cores M is a multiple of two in the following analysis. Then, the cores can be equally divided into two groups (i.e., each group consists of $\frac{M}{2}$ cores). We know that N primary tasks can fit on the cores in the first group under partitioned EDF if $N \leq \frac{\beta \cdot M}{2}$ [37]. For this case, N backup tasks also fit on the cores in the second group as the primary and backup of a task have the same timing and blocking parameters. Therefore, we can have the following lemma associated with the number of tasks and system schedulability.

Lemma 3. *For a set Ψ of $2 \cdot N$ primary and backup tasks running on an M-core system, the task set is guaranteed to be schedulable under partitioned-EDF with MSRP and PB if $N \leq \frac{\beta \cdot M}{2}$.*

Next, based on above definitions and analysis, we can obtain the following theorem related to the reliability and resource aware utilization bound (U^{bound}) for tasks that access shared resources on multicores under the partitioned EDF with MSRP, PB and the WFD mapping as follows. Here, U^{b1} refers to the upper-bound for the minimum allowable number of tasks (i.e., $\frac{\beta \cdot M}{2} + 1$), while U^{b2} accounts for the bound for total number of primary/backup tasks (i.e., N). The proof and the detailed discussions for the related parameters (i.e., β and σ) are available in supplementary material.

Theorem 1. *For a set Ψ of $2 \cdot N$ periodic primary and backup tasks that access shared resources in an M-core system, where M is an even number, the number of primary and backup tasks $2 \cdot N > \beta \cdot M$, $\beta \geq 1$ and $\sigma < \frac{1}{2+\beta}$, the system utilization*



(a) $\beta = 3$ and $\sigma = 0.05$ (b) $\beta = 1$ and $N = 25$ (c) $\beta = 3$ and $N = 25$ (d) Non-monotonicity of the bound

Fig. 1. Utilization bound and its non-monotonicity for partitioned-EDF and MSRP with the PB mechanism and WFD mapping.

bound U^{bound} for partitioned-EDF with MSRP, PB and WFD can be determined as

$$U^{bound} = \min\{U^{b1}, U^{b2}\}, \tag{6}$$

$$U^{b1} = (\beta \cdot M + 2) \cdot \left(\frac{1-\sigma}{1+\beta} - \sigma\right), \tag{7}$$

$$U^{b2} = \frac{2 \cdot M \cdot N}{M + 2 \cdot (N-1)} \cdot (1-\sigma) - 2 \cdot N \cdot \sigma. \tag{8}$$

4.2 Non-Monotonicity of the Bound

The interplay of the system and task parameters on the bound U^{bound} can be more explicitly illustrated in Fig. 1. As shown in Fig. 1a, when there are more available cores (i.e., larger M) and fewer tasks (i.e., smaller N) in the system, U^{bound} usually increases quickly. From Figs. 1b and 1c, we can see that U^{bound} decreases drastically when the synchronization overhead factor σ increases (e.g., when there are more and/or larger sized critical sections in tasks). For task systems with high resource access competition (e.g., $\sigma > 0.05$), the bound U^{bound} can be extremely low as shown in Figs. 1b and 1c, which rather constrains its usability. Moreover, with other parameters being fixed, larger bound can usually be obtained as β increases.

Generally, when there are more available cores, U^{bound} becomes larger. However, for given configurations of task set, the increased σ (due to increased M) and thus increased blocking overheads of tasks can in turn give rise to reduced U^{bound}. Such *anomaly* can be more clearly depicted for task sets with $N = 48$, $p^{min} = 130$, $n^{max,cs} = 3$ and the maximum raw task utilization being 0.03 as shown in Fig. 1d. Here, for a given largest size of critical sections in tasks (e.g., $c^{max,cs} = 0.4$ or $c^{max,cs} = 0.45$), U^{bound} drops as M increases. In addition, U^{bound} can also decrease as M increases when $c^{max,cs} = 0.35$ and $M \geq 8$. For the case of $c^{max,cs} = 0.3$, it turns out that there is $N \leq \frac{\beta \cdot M}{2}$ when $M > 8$. Therefore, based on Lemma 3, any primary and backup task set satisfying above configurations is guaranteed to be feasible under partitioned-EDF with MSRP and PB. In addition, the meaningless bounds (i.e., $U^{bound} < 0$ based on Equation (6)) can be obtained when $c^{max,cs} = 0.45$ and $M > 8$.

Next, we formally verify the non-monotonicity of the bound U^{bound}. First, we can have following lemmas associated with β and the proofs are available in the supplementary file, available online.

Lemma 4. *The minimum number of cores needed to feasibly allocate the task set Ψ under partitioned-EDF is $M^{min} = \left\lceil 2 \cdot \sum_{i=1}^{N} \frac{c_i}{p_i} \right\rceil$.*

Lemma 5. β cannot *decrease when the number of deployed cores decreases and other parameters are fixed.*

For a given number of available cores M, taking the floor function of β (see Lemma 2) into consideration, we know that there exists an integer ρ and an interval $[M, M-1, \ldots, M-\rho]$, where β remains invariant when M decreases. Define such interval as the *constant interval* of β and ρ as its maximum range. Based on the invariance of β in its constant interval, we can obtain the following sufficient condition for the non-monotonicity of U^{bound}, which constrains the range of the synchronization overhead factor σ related to β (the proof is available in supplementary material).

Theorem 2. *For a set Ψ of primary and backup tasks running on an M-core system with shared resources, the bound U^{bound} under partitioned-EDF with MSRP, PB and WFD as given in Equation (6) can* decrease *if the number of deployed cores increases when β is a constant and there is $\frac{1}{3+\beta} \leq \sigma < \frac{1}{2+\beta}$.*

5 Reliability and Resource Aware Mapping Algorithm

Although the bound U^{bound} can offer an efficient feasibility test for partitioned-EDF with MSRP and PB, it only focuses on the worst case scenarios, where the pessimistic blocking overheads of tasks are considered, such as α, β, γ and σ. Hence, the applicability of the bound is rather constrained, especially when σ becomes large as shown in Fig. 1. Moreover, based on Equations (1) and (2), we see that the synchronization costs of tasks can vary a lot for different partitionings of tasks upon cores. Thus, without the consideration of blocking interferences when ordering tasks to be assigned, the workload of partitions generated by the original WFD, e.g., the G-SS (Generalized Standby-Sparing) scheme [4], may be rather imbalanced, which can lead to high online overheads as validated from the empirical results in Section 6.3.

However, the intrinsic features of the bound essentially bring some important insights into the designs towards the resource and reliability aware partitioned scheduling algorithm as follows.

- The non-monotonicity of the bound U^{bound} indicates that the schedulability of task system may be further promoted through deploying fewer processor cores;

- To improve the feasibility test as shown in Proposition 1, the upper bound on the blocking overheads of primary and backup tasks needs to be further reduced.
- Since the primary and backup of a task have the same synchronization configurations, the mapping scheme needs to consider them on an equal footing when prioritizing tasks to be allocated. Moreover, the synchronization overheads of unmapped tasks need to be taken into account during task assignments for better schedulability;
- Different from conventional mapping heuristics, the partitioning algorithm needs to focus on the resource oriented workload balancing strategy to effectively reduce the run-time overhead for high practical viability.

5.1 Overview of RSA-TPA

To tackle the partition problem under consideration, we study effective heuristics for improved schedulability and balanced workload under the constraints of task synchronization and the requirements of system reliability. In general, for partitioned scheduling algorithm, there are two necessary phases when performing task-to-core allocations: a) prioritize tasks to be allocated; and b) find appropriate cores for tasks. In this paper, we propose a reliability and synchronization aware task partitioning algorithm (RSA-TPA) for multicore systems with shared resources. Specifically, the basic ideas of RSA-TPA are highlighted below.

- With the objective of reducing the negative effects of resource access contention on the schedulability of tasks, for given task set, RSA-TPA tries to find feasible partition by gradually *reducing* the number of deployed cores;
- Taking into account the limits for the inter-core synchronization interferences among tasks, we obtain a *tightened bound* on the global waiting times (thus synchronization overheads) for allocated tasks to improve the feasibility test based on Proposition 1 (see Section 5.2);
- Intuitively, for any task, when compared to local blocking time, the global waiting time often plays a more important role in the core utilization as given in Equation (4). Different from the existing task ordering schemes based on fixed utilizations [4], [6] or the policy of allocating primary tasks prior to backups [4], with the global waiting times being considered, un-allocated tasks are prioritized solely based on their *resource-aware utilizations* to improve the schedulability performance (see Section 5.3);
- Furthermore, by incorporating the changes in the blocking overheads of tasks for different mappings, RSA-TPA endeavors to generate feasible partitions with more *balanced workload*, where each task assignment results in minimum system utilization increase (see Section 5.4).

The outline of RSA-TPA is summarized in Algorithm 1. The partition Π is first initialized (line 1). From Lemma 4, the available number of cores M' is in the range $[M^{min}, M]$. To obtain a feasible mapping, RSA-TPA tries to reduce the number of deployed cores M' (lines 2 to 17). For all unmapped primary and backup tasks in the task set Γ, we choose the highest-priority task one at a time, where the priorities of primary and backup tasks are iteratively updated based on current mapping decisions and their estimated utilizations (line 5; Section 5.3). Then, for at most M' possible task allocations, the target core is found based on the system utilization increment (line 6; Section 5.4). If the current partition is infeasible (line 7), RSA-TPA conducts task assignments with fewer deployed cores; otherwise, the utilizations of all cores are updated based on the tightened blocking overheads of allocated primary and backup tasks (line 11; Section 5.2). Once a feasible mapping is obtained (line 14), RSA-TPA quits searching another feasible partition and returns (line 15). This policy is to be consistent with the *objective* of this work, i.e., generating feasible partition with balanced workload distribution across cores for potentially reduced online overhead. Its effectiveness is discussed and validated from measurement results as given in Section A8.3 (see supplementary file, available online). Finally, the RSA-TPA scheme either fails or returns the feasible task-to-core mapping Π (line 18).

Algorithm 1. Outline of RSA-TPA

```
    Input: Ψ = Ψ^P ∪ Ψ^B (the task set);
    Output: A feasible partition Π or FAIL;
1   Π = ∅;
2   for (M' : M → M^min) do
3     Ψ_k = ∅ (k = 1, ..., M'); Γ = Ψ;
4     while (Γ ≠ ∅) do
5       τ_i = ChooseTask(Γ, Π); //see Section 5.3;
6       m = SearchCore(τ_i, Π); //see Section 5.4;
7       if (m == null) then
8         Π = ∅; exit;
9       else
10        Γ = Γ − {τ_i}; Ψ_m = Ψ_m ∪ {τ_i};
          Π = {Ψ_1, ···, Ψ_M'};
11        Update U(Ψ_k) (k = 1, ..., M'); //see Section 5.2;
12      end
13    end
14    if (Π ≠ ∅) then
15      break;
16    end
17  end
18  Return (Π = ∅ ? FAIL: Π);
```

In what follows, we discuss the detailed resource-guided fault-tolerant mapping heuristics of RSA-TPA. Moreover, for the ease of presentations, we still use M to represent the number of *deployed cores* in the remainder of this section.

5.2 Tightened Global Waiting Times for Allocated Tasks

We see that for a task τ_i, the approach to compute its total global waiting time by simply accumulating $BW_{i,x}$ ($r_{i,x} = \mathcal{R}_a$) as shown in Equation (1) is rather pessimistic, where the largest critical section of tasks on other cores that needs access to resource $\mathcal{R}_a$ is assumed to affect task τ_i when it accesses resource $\mathcal{R}_a$ each time. Here, RSA-TPA tries to tighten the maximum global waiting times and maximum estimated ones for allocated tasks and unmapped tasks respectively in a *resource-oriented* manner.

Recall that the initial arrival time of every task is 0. We first study the limits on the inter-core synchronization interferences for the primary and backup tasks that access the same resources on different cores. Specifically, the inter-core blocking interferences among tasks can be verified as follows [32].

Proposition 2. *For two periodic tasks τ_i and τ_j on different cores scheduled by partitioned-EDF, the maximum number of jobs of τ_j, which may affect the execution of any job of τ_i due to the resource access competition, can be given as*

$$\pi_{i,j} = \begin{cases} 1, & p_i < p_j \wedge mod(p_j, p_i) = 0 \\ \frac{p_i}{p_j}, & p_i \geq p_j \wedge mod(p_i, p_j) = 0 \\ \left\lceil \frac{p_i}{p_j} \right\rceil + 1, & otherwise, \end{cases} \tag{9}$$

where $mod(x, y)$ refers to the remainder of dividing x by y.

It is worth noting that task τ_i and task τ_j may be the same task as the primary and backup of a task need to be partitioned upon different cores subject to the PB recovery mechanism.

Define Φ_i as the subset of resources that task τ_i needs to access. The subset of critical sections in primary/backup task τ_i where it accesses resource $\mathcal{R}_a$ ($\in \Phi_i$) is denoted as

$$\mathcal{S}_{i,a} = \{z_{i,x} | \forall z_{i,x} : r_{i,x} = \mathcal{R}_a; x = 1, \ldots, n_i\}. \tag{10}$$

Then, from Property 1, the number of inter-core interferences that can be experienced by any job of task τ_i ($\in \Psi_m$) when accessing resource $\mathcal{R}_a$ has the following two limits.

- $|\mathcal{S}_{i,a}|$: the number of critical sections in the set $\mathcal{S}_{i,a}$;
- $\pi_{i,j}$: the maximum number of jobs of any task τ_j on another core (i.e., $\tau_j \notin \Psi_m$) that need access to resource $\mathcal{R}_a$ during the execution of any job of task τ_i.

With the focus on the above limits for inter-core interferences among tasks, we study a resource-guided approach to reduce the maximum total busy-waiting time $BW^{max}_{i,\mathcal{R}_a}$ for *allocated* primary/backup task τ_i ($\in \Psi_m$) to access its *every resource* $\mathcal{R}_a$ ($\in \Phi_i$). Again, for any task, its primary/backup may cause inter-core blocking interferences on the other based on the PB model. Recall that the primary and backup of any task have the same blocking parameters. For primary/backup task τ_i, we define $\mathbb{S}_{\bar{i},a}$ as the subset of critical sections of all primary and backup tasks other than itself where resource $\mathcal{R}_a$ can be accessed, i.e., $\mathbb{S}_{\bar{i},a} = \cup_{\forall \tau_j \in (\Psi - \{\tau_i\})} \mathcal{S}_{j,a}$. The critical sections in the set $\mathbb{S}_{\bar{i},a}$ are ordered by their non-increasing sizes and the tie is broken arbitrarily.

To further tighten the upper bound on $BW^{max}_{i,\mathcal{R}_a}$, instead of directly getting $BW^{max}_{i,\mathcal{R}_a}$ based on $\pi_{i,j}$ (defined in Equation (9)) [32], here we restrict attention on the individual critical sections of any task on another core that may affect task τ_i when it accesses resource $\mathcal{R}_a$. We first give the following theorem and the proof can be found in supplementary material.

Theorem 3. *Let $f_{i,j} = \left\lfloor \frac{p_i}{p_j} \right\rfloor$. For two tasks τ_i and τ_j on different cores, if $mod(p_i, p_j) \neq 0$, the worst-case global waiting time $BW^{max}_{i,z_{j,y},\mathcal{R}_a}$ for task τ_i to access its resource $\mathcal{R}_a$, which is caused by task τ_j when accessing resource $\mathcal{R}_a$ in its critical section $z_{j,y}$ ($r_{j,y} = \mathcal{R}_a$), can be obtained as*

$$BW^{max}_{i,z_{j,y},\mathcal{R}_a} = c_{j,y} \cdot f_{i,j} + \min\{2 \cdot c_{j,y}, p_i - f_{i,j} \cdot p_j\}. \tag{11}$$

Next, we compute the busy-waiting time $BW^{max}_{i,z_{j,y},\mathcal{R}_a}$ of task τ_i for any number of jobs num ($\leq \pi_{i,j}$) of task τ_j, which may affect task τ_i when task τ_j accesses resource $\mathcal{R}_a$ in its critical section $z_{j,y}$ ($r_{i,j} = \mathcal{R}_a$). Based on Equation (11) in Theorem 3, the detailed steps to obtain $BW^{max}_{i,z_{j,y},\mathcal{R}_a}$ are shown in Algorithm 2.

Algorithm 2. CalBWMax (τ_i ($\in \Psi_m$), $z_{j,y}$ ($\tau_j \notin \Psi_m \wedge r_{j,y} = \mathcal{R}_a$), num ($\leq \pi_{i,j}$), $\mathcal{R}_a$ ($\in \Phi_i$))

Input: τ_i, $f_{i,j} = \left\lfloor \frac{p_i}{p_j} \right\rfloor$;
Output: $BW^{max}_{i,z_{j,y},\mathcal{R}_a}$;
if $(mod(p_i, p_j) \neq 0 \wedge num == \pi_{i,j})$ **then**
  $BW^{max}_{i,z_{j,y},\mathcal{R}_a} = c_{j,y} \cdot f_{i,j} + \min\{2 \cdot c_{j,y}, p_i - f_{i,j} \cdot p_j\}$;
else
  $BW^{max}_{i,z_{j,y},\mathcal{R}_a} = c_{j,y} \cdot num$;
end

Algorithm 3. CalBW(τ_i ($\in \Psi_m$), $\mathcal{R}_a$ ($\in \Phi_i$))

Input: τ_i, $\mathbb{S}_{\bar{i},a}$, $\Pi = \{\Psi_1, \cdots, \Psi_M\}$;
Output: $BW^{max}_{i,\mathcal{R}_a}$;
1 $BW^{max}_{i,\mathcal{R}_a} = 0$; $limit[k] = |\mathcal{S}_{i,a}|$ $(k = 1, \ldots, M \wedge k \neq m)$;
2 **for** $(z_{j,y} \in \mathbb{S}_{\bar{i},a})$ **do**
3 **if** $(\exists k, \tau_j \in \Psi_k \wedge k \neq m)$ **then**
4 $num = \min\{\pi_{i,j}, limit[k]\}$;
5 **if** $(num == 0)$ **then**
6 continue;
7 **else**
8 $BW^{max}_{i,\mathcal{R}_a} +=$ CalBWMax $(\tau_i, z_{j,y}, num, \mathcal{R}_a)$;
  $limit[k] -= num$; //see Algorithm 2;
9 **end**
10 **end**
11 **end**

Notice that the number of inter-core blocking interferences on task τ_i is also constrained by $|\mathcal{S}_{i,a}|$ when τ_i accesses resource $\mathcal{R}_a$. For a given task-to-core mapping Π, the basic steps to calculate the worst-case $BW^{max}_{i,\mathcal{R}_a}$ are shown in Algorithm 3. The remaining limits on the number of interferences from other cores are first initialized (line 1). Then, by the non-ascending order of sizes, the critical sections in the set $\mathbb{S}_{\bar{i},a}$ are tested one at a time (line 2). For a critical section $z_{j,y}$ ($\in \mathbb{S}_{\bar{i},a}$) where task τ_j accesses resource $\mathcal{R}_a$ on another core $\mathcal{P}_k$ ($k \neq m$) (i.e., τ_j may affect τ_i when accessing resource $\mathcal{R}_a$), the number of interferences on task τ_i follows the limits from task τ_j (i.e., $\pi_{i,j}$) and the remaining limits from core $\mathcal{P}_k$ (i.e., $limit[k]$) (line 4). When there exists no such interference (line 5), the next critical section in the set $\mathbb{S}_{\bar{i},a}$ is to be checked (line 6); otherwise, $BW^{max}_{i,\mathcal{R}_a}$ accumulates based on Algorithm 2 and the remaining interference limits ($limit[k]$) from core $\mathcal{P}_k$ is updated accordingly (line 8). Finally, the resulting $BW^{max}_{i,\mathcal{R}_a}$ is returned once all critical sections in the set $\mathbb{S}_{\bar{i},a}$ have been tested.

Therefore, the tightened maximum overall busy-waiting time BW_i^{max} of task τ_i can be expressed as $BW_i^{max} = \sum_{\forall \mathcal{R}_a \in \Phi_i} BW_{i,\mathcal{R}_a}^{max}$. Hence, with the reduced blocking overheads of tasks, we can have lower core utilizations in Equation (4) for better feasibility test.

5.3 Utilization Estimates for Unmapped Tasks

Based on Equation (4), we find that for every task, in addition to its WCET, its global waiting time and local blocking time can also affect the core utilization. Intuitively, for any task, its busy-waiting time usually plays a more important part in the core utilization compared to local blocking time. More importantly, the additional task copies based on the PB technique can potentially increase the blocking interferences in task system. Therefore, with the focus on the changing blocking overheads during task assignments, the *task ordering priorities* of unmapped primary and backup tasks need to be updated *iteratively* for determining the sequence to be mapped onto cores, subject to such variations for better system schedulability. This is quite different from the traditional partitioned scheduling without the consideration of task synchronization requirements [37], where tasks are usually sorted by their fixed utilizations.

In view of the varying limits for the blocking interferences during task-to-core mappings (see Algorithm 3), the utilization of an *unmapped* primary/backup task τ_i is estimated as

$$u_i^{est} = (c_i + BW_i^{est})/p_i, \tag{12}$$

where BW_i^{est} is the estimated overall global waiting time taken by task τ_i to access all its resources. The detailed steps to obtain the estimated global waiting time $BW_{i,\mathcal{R}_a}^{est}$ for task τ_i to access its **every resource** $\mathcal{R}_a$ ($\in \Phi_i$) are given in Algorithm 4.

Algorithm 4. GetBWEst(τ_i, $\mathcal{R}_a$ ($\in \Phi_i$))

Input: τ_i, $\mathbb{S}_{\bar{i},a}$, $\Pi = \{\Psi_1, \ldots, \Psi_M\}$;
Output: $BW_{i,\mathcal{R}_a}^{est}$;
1 $BW_{i,\mathcal{R}_a}^{est} = 0$; $limit[k] = |\mathcal{S}_{i,a}|$ $(k = 1, \ldots, M)$; $sum = (M-1) \cdot |\mathcal{S}_{i,a}|$;
2 **for** $(z_{j,y} \in \mathbb{S}_{\bar{i},a})$ **do**
3 **if** $(\exists k, \tau_j \in \Psi_k)$ **then**
4 $num = \min\{sum, \pi_{i,j}, limit[k]\}$; $limit[k] -= num$;
5 **else**
6 $num = \min\{sum, \pi_{i,j}, |\mathcal{S}_{i,a}|\}$;
7 **end**
8 $BW_{i,\mathcal{R}_a}^{est} += \text{CalBWMax}\ (\tau_i, z_{j,y}, num, \mathcal{R}_a)$;
9 **if** $((sum -= num) == 0)$ **then**
10 exit;
11 **end**
12 **end**

Similar to Algorithm 3, Algorithm 4 also relies on the partial mapping Π and inter-core interference limits. In particular, a total remaining limit (i.e., sum) is adopted as the number of inter-core interferences on task τ_i when it accesses resource $\mathcal{R}_a$ is at most $(M-1) \cdot |\mathcal{S}_{i,a}|$ based on Property 1. Recall that the critical sections in the set $\mathbb{S}_{\bar{i},a}$ are sorted by their non-increasing sizes. The remaining limits on the number of interferences from cores are first initialized (line 1). The inter-core interference limits for the allocated tasks (line 4) and unmapped tasks (line 6) are tackled differently. Here, for the critical section in an unmapped task τ_j whose exact assignment is unknown till now, the number of its interferences on task τ_i conforms to the limits from $\pi_{i,j}$, $|\mathcal{S}_{i,a}|$ and sum. If there exist remaining interferences, $BW_{i,\mathcal{R}_a}^{est}$ accumulates correspondingly based on Algorithm 2 (line 8); otherwise, the resulting $BW_{i,\mathcal{R}_a}^{est}$ is returned (lines 9 to 11).

Thus we have $BW_i^{est} = \sum_{\forall \mathcal{R}_a \in \Phi_i} BW_{i,\mathcal{R}_a}^{est}$. The estimated utilizations of unmapped primary and backup tasks are updated from Equation (12) and they are prioritized by their non-ascending *new* estimated utilizations, where the tie is broken favoring the primary task with smaller index. Finally, the primary/backup task having the highest task ordering priority (i.e., the maximum estimated utilization) is chosen to be allocated next.

5.4 Selection of Target Processor Core

The second stage of mapping scheme is finding proper cores to accommodate tasks. Here we propose an effective heuristic where each task assignment minimizes the system utilization increase for balanced workload across cores.

The rationale behind this heuristic is two-fold: a) when most tasks are gathered into several over-loaded (heavy) cores, the feasibility condition as given in Proposition 1 for these cores may be more likely violated since the new tasks on other cores possibly cause increased busy-waiting times and thus local blocking times for those tasks on heavy cores; and b) over-exploiting several cores may lead to severe workload imbalance, which may result in more context switchings on cores under EDF (thus higher online overhead as validated in Section A8.3 in supplementary material). Therefore, this strategy can improve not only the schedulability performance but workload balancing (thus *practical viability*) under the resource access race and reliability requirements.

We consider the set $\mathcal{V}_i$ of all *valid* cores for new primary/backup task τ_i, where the other copy of task τ_i (if exists) is not allocated. Define the *new* utilization of any valid core $\mathcal{P}_k$ ($\in \mathcal{V}_i$) with a new task τ_i as $U(\Psi_k \cup \{\tau_i\})$ based on Equation (4). Based on the feasibility condition as presented in Proposition 1, when there is $U(\Psi_k \cup \{\tau_i\}) > 1$, task τ_i does not fit on core $\mathcal{P}_k$ and we set $U(\Psi_k \cup \{\tau_i\})$ as ∞. Note that after task τ_i is partitioned upon a core $\mathcal{P}_k$, the synchronization costs of tasks on other cores may be altered by this new task due to the resource access contention in the system. Thus, for another core $\mathcal{P}_m$, the blocking overheads of all its tasks need to be adjusted correspondingly and its *new* core utilization is denoted as $U_{\tau_i}(\Psi_m)$, where $U_{\tau_i}(\Psi_m)$ is also set as ∞ once there is $U_{\tau_i}(\Psi_m) > 1$.

By assuming that task τ_i is designated to core $\mathcal{P}_k$, the *new* system utilization can be defined as

$$U_{\tau_i,\mathcal{P}_k}(\Pi) = \max\{U(\Psi_k \cup \{\tau_i\}), \max\{U_{\tau_i}(\Psi_m) | \forall m : m \neq k\}\}. \tag{13}$$

Then, the system utilization increase for this task allocation is

$$\Delta_{\tau_i,\mathcal{P}_k}^{sys} = U_{\tau_i,\mathcal{P}_k}(\Pi) - U(\Pi). \tag{14}$$

TABLE 1
System and Task Parameters in Experiments

Parameters	Values/ranges
Number of processor cores (M)	2, 4, 8, 16
Normalized system raw utilization ($NSRU$)	[0.4, 0.9]
Number of resources (R)	[1, 10]
Periods of tasks	[50, 2000]
Number of critical sections per task	[1, 10]
Critical section ratio (CSR)	[0.005, 0.05]

By probing every possible mapping of task τ_i for all its valid cores and obtaining at most M new system utilizations, the most appropriate core $\mathcal{P}_m$ that can lead to *valid* minimum system utilization increment is chosen, i.e., $\min\{\Delta^{sys}_{\tau_i,\mathcal{P}_k} | \forall \mathcal{P}_k \in \mathcal{V}_i : \Delta^{sys}_{\tau_i,\mathcal{P}_k} \leq 1\}$ based on Equations (13) and (14). If there is no feasible core for task τ_i, RSA-TPA continues finding feasible partition with fewer deployed cores as shown in Algorithm 1. Once there exist more than one such core, the core with smaller index is preferable.

To reduce the complexity of RSA-TPA, an efficient algorithm (RSA-TPA-efficient) is available in supplementary material.

5.5 Complexity and an Example for RSA-TPA

We discuss the time complexity of RSA-TPA here while a concrete example for RSA-TPA can be found in supplementary material.

Assume that the number of critical sections per task and the number of resources R are constants. For any mapped task, getting its busy-waiting time needs $O(N)$ time based on Algorithm 3 and obtaining the local blocking time also takes $O(N)$ time based on Equation (2). Based on Algorithm 4, the maximum estimated utilization for each un-allocated task can be calculated in $O(N)$ time and thus choosing the highest-priority task can be done in $O(N^2)$ time. For each core probe, $O(N^2)$ time is needed to update the blocking costs of tasks and utilizations of cores. Moreover, computing the minimum system utilization increase takes $O(M)$ time. Since M is usually smaller than N, each task assignment can be done in $O(M \cdot N^2)$ time and thus finding target cores for all tasks needs $O(M \cdot N^3)$ time. Hence, with the decreased number of deployed cores being considered, the overall time complexity of the RSA-TPA scheme can be determined as $O(M^2 \cdot N^3)$.

6 Evaluations and Discussions

Now we report on the performance evaluations of the RSA-TPA schemes through extensive simulations and measurements. For comparison, we extended Synchronization-Aware WFD (*SA-WFD*) [14] and Greedy Slacker for MSRP (*GS-MSRP*) [16] by incorporating the PB model. Here, several mapping schemes (such as [29] and [30] as discussed in Section 2) that exploit different resource access paradigms are omitted in the experiments. We also implemented the recent *G-SS* (Generalized Standby-Sparing) algorithm targeted at reliability management in multiprocessors [4].

6.1 Simulation Settings

In simulations, we consider a homogeneous multicore architecture, such as Qualcomm SDM850 mobile processor with eight Kryo™385 CPU cores. These schemes are evaluated based on the following performance metrics: a) *schedulability ratio* defined as the number of task sets having feasible partitions under these schemes over the total number of task sets; and b) *workload imbalance factor (WIF)* defined as $\frac{U(\Pi)-U^{min}(\Pi)}{U(\Pi)}$ to assess the workload balance of *feasible* mappings generated by the schemes, where $U(\Pi)$ is the system utilization (defined in Section 3.4) and $U^{min}(\Pi) = \min\{U(\Psi_m) | \forall \mathcal{P}_m\}$.

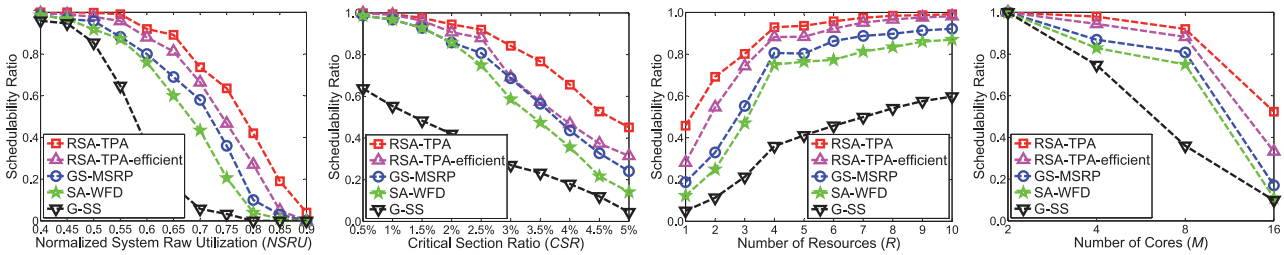
The schedulability performance of a mapping scheme can be affected by many factors. We consider the following parameters as summarized in Table 1. The system parameters include the number of cores (M), the number of shared resources in the system (R) and the *normalized system raw utilization* defined as $NSRU = \frac{2 \cdot \sum_{i=1}^{N} \frac{c_i}{p_i}}{M}$ which excludes the blocking costs of tasks. For the task system, we restrict attention on the periods of tasks, the number of critical sections in a task and the critical section ratio CSR (defined as the ratio of the total size of critical sections in a task to its WCET).

For given M, R, $NSRU$ and CSR, the task sets are generated as follows. The raw utilization of task τ_i (i.e., $\frac{c_i}{p_i}$) without considering blocking costs are generated randomly according to the UUniFast scheme [38]. The average raw utilization u^{ave} of a task is set as 0.05, 0.1 and 0.2 respectively, where any task with raw utilization being more than 100 percent is discarded. We generate enough number of tasks such that $NSRU$ reaches a given value. Then, the average number of primary/backup tasks in a task set is $\frac{NSRU \cdot M}{2 \cdot u^{ave}}$. The periods of tasks are uniformly distributed within [50, 2000] and thus the WCET of task τ_i (i.e., c_i) can be obtained based on its raw utilization and period. Next, the number of critical sections n_i in task τ_i is generated in the range $[1, 10]$, where each resource that task τ_i accesses is randomly selected. Finally, the execution time of a critical section in task τ_i is generated within $[\frac{x \cdot c_i \cdot CSR}{n_i}, \frac{(2-x) \cdot c_i \cdot CSR}{n_i}]$, where x represents the degree of variations of the execution time.

6.2 Simulation Results

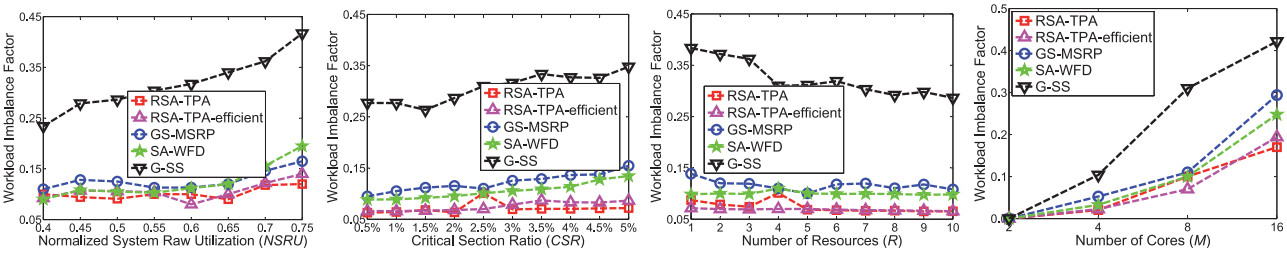
In simulations, the default parameter values are $M = 8$, $NSRU = 0.6$, $R = 4$, $CSR = 0.025$ and $x = 0.2$ unless otherwise noted. For the results reported below, each data point represents the average of 30,000 synthetic task sets. Figs. 2 and 3 respectively show the schedulability ratios and workload imbalance factors (WIFs) generated by the mapping schemes considered. Based on the resource access costs measured from real-life applications [33] (where the minimum task period is $3ms$, the maximum execution time in a critical section is $15.5us$ and the maximum number of resource requests by a task is 9), CSR is chosen in the range [0.5%, 5%] in the experiments.

Fig. 2a shows the effects of varying $NSRU$ on the schedulability performance of the mapping schemes. In general, a larger $NSRU$ refers to higher system workload when other parameters are fixed. Compared to other schemes, RSA-TPA tries to decrease the number of deployed cores, tightens the bound on the blocking overheads of tasks and uses



(a) Schedulability with varying $NSRU$ (b) Schedulability with varying CSR (c) Schedulability with varying R (d) Schedulability with varying M

Fig. 2. Schedulability performance of the mapping schemes.



(a) WIF with varying $NSRU$ (b) WIF with varying CSR (c) WIF with varying R (d) WIF with varying M

Fig. 3. Performance of workload balancing for the mapping schemes.

a dynamic task ordering approach based on the varying limits for the blocking interferences during task mappings. Therefore, RSA-TPA can reduce the impacts of blocking overheads of tasks on the system schedulability, and thus can have greater advantages in the acceptance ratio when the system workload becomes heavier (i.e., larger $NSRU$). Based on the principle of RSA-TPA, RSA-TPA-efficient has the acceptance ratio close to RSA-TPA but higher than other schemes that focus on either system reliability or task synchronization. We see that the G-SS scheme usually has the worst performance as it ignores the impacts of blocking interferences on the schedulability. Moreover, when compared to SA-WFD, GS-MSRP has better performance due to its more subtle feasibility test (i.e., response time analysis) and the policy of mapping a task onto the core that the minimum slack of all its tasks is maximal among all possible cores.

Usually, a greater CSR stands for larger blocking overheads of tasks with other parameters being fixed. As shown in Fig. 2b , when CSR becomes larger, the acceptance ratios of the GS-MSRP, SA-WFD and G-SS schemes drop quickly; in contrast, RSA-TPA exhibits relatively steady performance due to the resource-oriented and fault-tolerant mapping heuristics as discussed in Section 5.

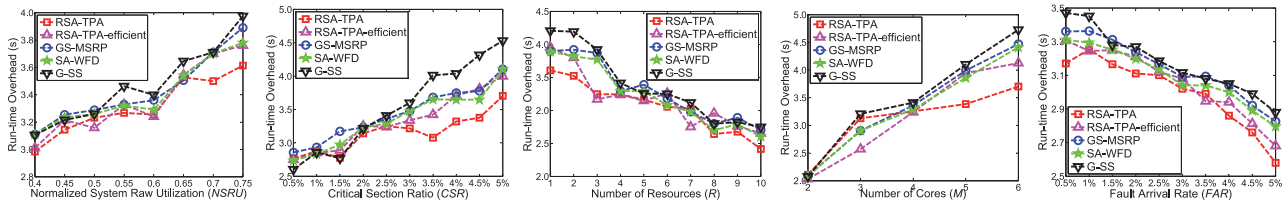
Fig. 2c depicts the performance evaluations of the mapping schemes as the number of resources R can vary. Intuitively, with other parameters (e.g., $NSRU$, M and CSR) being fixed, a smaller R accounts for more blocking requirements in the system. Similar to the trends as those shown in Fig. 2b , when R becomes smaller, RSA-TPA can yield better schedulability ratio compared to other schemes based on above-mentioned reasons.

Fig. 2d further illustrates the performance comparison among all algorithms with different number of available cores M. When other parameters are fixed, more available cores normally indicate more inter-core blocking interferences (i.e., longer busy-waiting times) among tasks for given R, CSR and $NSRU$. For this case (e.g., $M = 16$), RSA-TPA and RSA-TPA-efficient can obtain much better acceptance ratio in comparison with other schemes. Moreover, the schedulability performance for the algorithms without the consideration of the PB recovery scheme is evaluated and discussed in Section A7 (in supplementary material).

Recall that RSA-TPA tries to minimize the system utilization increase for each task allocation and RSA-TPA-efficient exploits a dual minimization strategy for each mapping. Not surprisingly, as show in Fig. 3, both RSA-TPA and RSA-TPA-efficient can usually generate feasible partitions with more balanced workload distribution (i.e., smaller WIF) compared to other schemes, thanks to their inherent resource-oriented workload balancing policies. Moreover, our mapping algorithms usually perform better for the scenarios with more blocking interferences in the system, such as larger $NSRU$ in Fig. 3a , greater CSR in Fig. 3b , smaller R in Fig. 3c as well as larger M in Fig. 3d . Note that WIF is meaningful only for the task sets *schedulable by all schemes*. As the acceptance ratio generated by G-SS is extremely low when $NSRU$ reaches 0.75, $NSRU$ is chosen within [0.4, 0.75] here.

6.3 Measurement Evaluations

To evaluate the practicability of the RSA-TPA schemes, we implemented the mapping algorithms under consideration in Linux kernel with version 2.6.38.8, which run on a server with 32 nm AMD FX-8320 processor (8 cores, 3.5 Ghz clock speed, 8M L2 and L3 cache) and 8G RAM. To implement real-time scheduling, we followed the basic design paradigms of LITMUS^{RT} [33] which is built on Linux kernel, including the construction of the highest-priority scheduler, the parameter delivery from user space to kernel space, etc. Moreover, to assess the impacts of resource access race and PB recovery mechanism on the run-time overhead, we supplemented additional schemes in kernel as described below, such as MSRP protocol, online job cancellation and global synchronization mechanisms. We also utilized the available



(a) Overhead with varying $NSRU$ (b) Overhead with varying CSR (c) Overhead with varying R (d) Overhead with varying M (e) Overhead with varying FAR

Fig. 4. Measurement performance of the task partitioning algorithms with respect to run-time overhead.

Linux infrastructure to establish partitioned EDF scheduler: a new scheduling class with the highest priority is added to the traditional Linux scheduler and it always executes the highest-priority jobs prior to the regular Linux ones. The new scheduler replaces the original Linux scheduler to invoke the initialization functions, scheduling as well as tick handler at run-time.

Similar to the design patterns of LITMUS^{RT}, we provided a user space library to create primary and backup tasks through multi-threading. The tasks are initially created as non real-time where each task executes the same function codes by increasing a local variable in while-loop to stress the operating system [17], [32], [33]. Moreover, we constructed individual FIFO job queue for each resource (i.e., global shared variable) and implemented the MSRP protocol (including the operations for the queues of resources) in kernel for measurements [32]. For each resource access, its location in a primary/backup task and the amount of access time conform to the parameter settings of tasks (see Section 6.1 and parameter settings below). Furthermore, the system call is invoked to notify the kernel when a task acquires a resource or leaves a critical section [32], [33]. Specifically, following the rules of MSRP as presented in Section 3.3, when a task issues a request for a resource and becomes the header task in the queue of this resource, it locks the corresponding global variable through spin lock until it receives the notification of exit. Meanwhile, other tasks from different cores (if they exist), which need to visit the resource and stay in the FIFO queue, stop executing and wait for the unlocking of this global variable.

The timing and blocking parameters of tasks are delivered from user space to kernel space using a system call, and then per-task data structures and job queues of resources are constructed [17], [32], [33]. In addition, we implemented two additional *global synchronization mechanisms* in the experiments. The first one is exploited to synchronize the tick counters of cores; while the second is used for the reliability management, where the primary/backup of a job can be safely abandoned upon exiting all its critical sections as soon as the other one on another core finishes executing successfully.

The *metric* of measurement performance is run-time overhead, including context switching overhead, scheduling overhead caused by the operations for individual job queues of cores and resources, synchronization overheads invoked by synchronization-related system calls and the additional overheads for two global synchronization mechanisms as mentioned above.

Parameter Settings. There is an additional parameter necessary for the measurements: *fault arrival rate* (FAR) that accounts for the online execution variations of tasks. As the permanent fault occurs rather rarely in practice, we evaluate the effects of transient fault rates on the overhead performance here. The value of FAR is ranged from 0.5 to 5 percent by randomly selecting affected jobs. In addition, the period range for a task is set as $[5ms, 1s]$ and each task set executes for 20 seconds. Unless specified otherwise, the default parameter values are $M = 4$, $NSRU = 0.6$, $R = 4$, $CSR = 0.025$, $FAR = 0.01$ and $x = 0.2$. For the results given below, each data point corresponds to the average result of 1000 task sets *schedulable* by all mapping schemes in simulations.

Empirical Results. The empirical results for the online overheads (in s) of the mapping algorithms are shown in Fig. 4. The results exhibit the usability of RSA-TPA: its overhead normally consumes about 3 to 5 percent of the total execution time on cores (i.e., 80 seconds by default where $M = 4$), and it is usually lower than those of other schemes (e.g., 20 percent less) due to the resource-aware workload balancing policy as stated in Section 5.4, which can potentially reduce the number of context switchings (due to task preemptions under EDF) on cores. The detailed discussions for measurement results are available in the supplementary file, available online.

7 Conclusions

For period real-time tasks that access shared resources on multicores and are scheduled by partitioned-EDF with the MSRP protocol and PB mechanism, we develop the *first* utilization bound and then verify its anomaly. The non-monotonicity of the bound indicates that the schedulability of tasks may be promoted by decreasing the number of deployed cores. Based on the insights obtained from the bound, we study the *first* task partitioning algorithm to implement the joint management of task synchronization and reliability maintenance. Several resource-guided and fault-tolerant policies are investigated for better schedulability of task system and workload balancing on deployed cores.

In comparison with the existing mapping schemes that focus on either resource access race or system reliability, the experimental results show that the policies of our partitioning algorithms, such as the reduced number of deployed cores, tightened bound on the synchronization overheads of tasks as well as dynamic task ordering prioritization for unallocated tasks, can effectively improve the schedulability ratio, especially when more blocking requirements in task system are expected. Moreover, the effective heuristic of task-to-core mapping is validated from empirical experiments, which can generate more balanced workload

distribution across cores and thus can turn out lower online overheads.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC) Awards 61872411 and 61472150.

References

[1] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[2] D. K. Pradhan, *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[3] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single event transients in digital CMOS–a review," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 1767–1790, Jun. 2013.
[4] Y. Guo, D. Zhu, H. Aydin, J.-J. Han, and L. T. Yang, "Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems," *J. Syst. Archit.*, vol. 78, pp. 68–80, 2017.
[5] H. Han, W. Bao, X. Zhu, X. Feng, and W. Zhou, "Fault-tolerant scheduling for hybrid real-time tasks based on CPB model in cloud," *IEEE Access*, vol. 6, no. 2, pp. 18616–18629, 2018.
[6] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 813–825, Mar. 2017.
[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.
[8] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, 1991.
[9] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca, "A comparison of MPCP and MSRP when sharing resources in the janus multiple-processor on a chip platform," in *Proc. Real-Time Embedded Technol. Appl. Symp.*, 2003, pp. 189–198.
[10] K. Lakshmanan, D.-D. Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *Proc. IEEE Real-Time Syst. Symp.*, 2009, pp. 469–478.
[11] A. Block, H. Leontyev, B. B. Brandenburg, and J.-H. Anderson, "A flexible real-time locking protocol for multiprocessors," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2007, pp. 47–56.
[12] M. Ebrahimi, A. Evans, M. B. Tahoori, R. Seyyedi, E. Costenaro, and A. Dan, "Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2014, pp. 1–6.
[13] S. Borkar, "Extreme energy efficiency by near threshold voltage operation," in *Near Threshold Computing*. Cham, Switzerland: Springer, 2016.
[14] J.-J. Han, X. Wu, D. Zhu, H. Jin, L. T. Yang, and J.-L. Gaudiot, "Synchronization-aware energy management for VFI-based multicore real-time systems," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1682–1696, Dec. 2012.
[15] T. H. Tsai, L. F. Fan, Y. S. Chen, and T. S. Yao, "Triple speed: Energy-aware real-time task synchronization in homogeneous multi-core systems," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1297–1309, Apr. 2016.
[16] A. Wieder and B. B. Brandenburg, "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks," in *Proc. IEEE Int. Symp. Ind. Embedded Syst.*, 2013, pp. 49–58.
[17] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time os," *Real-Time Syst.*, vol. 49, no. 6, pp. 669–714, 2013.
[18] J. Ras and A. M. Cheng, "An evaluation of the dynamic and static multiprocessor priority ceiling protocol and the multiprocessor stack resource policy in an smp system," in *Proc. Real-Time Embedded Technol. Appl. Symp.*, 2009, pp. 13–22.
[19] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems," in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2009, pp. 193–202.
[20] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware standby-sparing technique for periodic real-time applications," in *Proc. IEEE Int. Conf. Comput. Des.*, 2011, pp. 190–197.
[21] Y. Xiang and S. Pasricha, "Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 1, no. 4, pp. 220–235, Apr. 2015.
[22] R. Melhem, D. Moss, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217–231, Feb. 2004.
[23] S. Ghosh, R. Melhem, and D. Moss, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, Mar. 1997.
[24] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time systems," *J. Parallel Distrib. Comput.*, vol. 64, no. 5, pp. 629–648, 2004.
[25] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, and K. Li, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Sustainable Comput.*, vol. 3, no. 3, pp. 167–181, Jul.-Sep. 2018.
[26] B. B. Brandenburg and J.-H. Anderson, "Optimality results for multiprocessor real-time locking," in *Proc. IEEE Real-Time Syst. Symp.*, 2010, pp. 49–60.
[27] A. Burns and A. J. Wellings, "A schedulability compatible multiprocessor resource sharing protocol–mrsp," in *Proc. Euromicro Conf. Real-Time Syst.*, 2013, pp. 282–291.
[28] N. C. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times*. Citeseer, 1991.
[29] W. H. Huang, M. Yang, and J. J. Chen, "Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share?," in *Proc. Real-Time Syst. Symp.*, 2016, pp. 111–122.
[30] V. D. B. Georg, J. J. Chen, W. H. Huang, and M. Yang, "Release enforcement in resource-oriented partitioned scheduling for multiprocessor systems," in *Proc. Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 287–296.
[31] J. Wu, "A survey of energy-efficient task synchronization for real-time embedded systems," in *Proc. Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2017, pp. 1–6.
[32] J.-J. Han, X. Tao, D. Zhu, and L. Yang, "Resource sharing in multicore mixed-criticality systems: Utilization bound and blocking overhead," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3626–3641, Dec. 2017.
[33] B. B. Brandenburg and J. H. Anderson, "A comparison of the M-PCP, D-PCP, and FMLP on litmusrt," in *Proc. Int. Conf. Principles Distrib. Syst.*, 2008, pp. 105–124.
[34] J. J. Han, M. Lin, D. Zhu, and L. T. Yang, "Contention-aware energy management scheme for NoC-based multicore real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 691–701, Mar. 2015.
[35] C. Zoubek and P. Trommler, "Overview of worst case execution time analysis in single- and multicore environments," in *Proc. Int. Conf. Archit. Comput. Syst.*, 2017, pp. 1–5.
[36] M. F. Minicz and A. Anzaloni, "Fault recovery performance in multicast networks for smart grid," *IEEE Latin America Trans.*, vol. 15, no. 11, pp. 2207–2213, Nov. 2017.
[37] J. M. López, J. L. Díaz, and D. F. García, "Utilization bounds for EDF scheduling on real-time multiprocessor systems," *Real-Time Syst.*, vol. 28, no. 1, pp. 39–68, 2004.
[38] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1/2, pp. 129–154, 2005.

Jian-Jun Han received the PhD degree in computer science and engineering from the Huazhong University of Science and Technology (HUST), in 2005. He is now a professor with the School of Computer Science and Technology in HUST. He worked with the University of California, Irvine as a visiting scholar between 2008 and 2009, and with the Seoul National University between 2009 and 2010. His research interests include real-time systems, parallel processing and green computing.

Zhenjiang Wang is working toward the doctor's degree in the School of Computer Science and Technology at HUST. His current research interests include real-time scheduling algorithm and operating systems.

Sunlu Gong is now working towards the master's degree in the School of Computer Science and Technology at HUST. His current research interests include real-time scheduling algorithm and embedded systems.

Tianpeng Miao is now an engineer in the School of Computer Science and Technology at HUST. His current research interests include embedded systems and natural language processing.

Laurence T. Yang received the PhD degree in computer science from the University of Victoria, BC, Canada. He is currently a professor at School of Computer Science and Technology, Huazhong University of Science and Technology, China and at Department of Computer Science, St. Francis Xavier University, Canada. His current research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing.