


# Hard Real-Time Stationary GANG-Scheduling

Niklas Ueter ✉ 

Department of Computer Science, TU Dortmund University, Germany

Mario Günzel ✉ 

Department of Computer Science, TU Dortmund University, Germany

Georg von der Brüggen ✉ 

Max Planck Institute for Software Systems, Kaiserslautern, Germany

Jian-Jia Chen ✉ 

Department of Computer Science, TU Dortmund University, Germany

---

## Abstract

The scheduling of parallel real-time tasks enables the efficient utilization of modern multiprocessor platforms for systems with real-time constraints. In this situation, the gang task model, in which each parallel sub-job has to be executed simultaneously, has shown significant performance benefits due to reduced context switches and more efficient intra-task synchronization.

In this paper, we provide the first schedulability analysis for sporadic constrained-deadline gang task systems and propose a novel stationary gang scheduling algorithm. We show that the schedulability problem of gang task sets can be reduced to the uniprocessor self-suspension schedulability problem. Furthermore, we provide a class of partitioning algorithms to find a stationary gang assignment and show that it bounds the worst-case interference of each task. To demonstrate the effectiveness of our proposed approach, we evaluate it for implicit-deadline systems using randomized task sets under different settings, showing that our approach outperforms the state-of-the-art.

**2012 ACM Subject Classification** Computing methodologies → Concurrent algorithms; Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Real-time operating systems

**Keywords and phrases** Real-Time Systems, Gang Scheduling, Parallel Computing, Scheduling Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ECRTS.2021.10

**Funding** This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware (Project no. 398602212). This result is part of two projects (PropRT and TOROS) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements No. 865170 and No. 803111).



## 1 Introduction

In hard real-time systems, it is mandatory to verify the temporal behavior of the application, e.g., the compliance to deadline constraints, by means of timing analysis. Due to the high computational demands of modern real-time systems, multiprocessor platforms are increasingly utilized since they potentially allow parallel tasks to be executed efficiently. In parallel task scheduling, inter- and intra-task parallelism has to be considered in the timing analysis, where inter-task parallelism refers to the co-scheduling of different tasks and intra-task parallelism refers to parallel execution of a single task. In the context of task models for parallel computing, fork/join models [26], synchronous parallel task models, and DAG (directed-acyclic graph) based task models [4, 5, 10, 11, 18, 19, 28] have been proposed and analyzed with respect to real-time constraints.



© Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen;

licensed under Creative Commons License CC-BY 4.0

33rd Euromicro Conference on Real-Time Systems (ECRTS 2021).

Editor: Björn B. Brandenburg; Article No. 10; pp. 10:1–10:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 10:2 Hard Real-Time Stationary Gang-Scheduling

The scheduling algorithms for parallel tasks can be classified into three models: *rigid*, *moldable*, and *malleable* tasks. A parallel task is called *rigid* if the number of processors assigned to it is specified externally to the scheduler a priori and does not change throughout its execution; *moldable* if the number of processors assigned to it is determined by the scheduler and does not change throughout its execution; and *malleable* if the number of processors assigned to it can be changed by the scheduler during its execution. Such classifications can be found in the literature of multiprocessor scheduling and real-time systems such as [20].

In the gang task model, a set of threads is grouped together into a so called *gang* with the additional constraint that all threads of a gang must be co-scheduled at the same time on available processors. It has been demonstrated that gang-based parallel computing can improve the performance in many cases [17, 23]. Even more, Wasly et al. [32] provided experimental evidence of negative effects of non-gang scheduling with respect to the number of context-switches and increased thread execution time due to blocking when threads are not executed together. Moreover, the authors argue that by scheduling all threads of a task in-parallel, the communication time can be easily accounted for, given that the inter-processor interconnect provides real-time bounds. Due to its practicability, the gang model is supported by many parallel computing standards, e.g., MPI, OpenMP, Open ACC, or GPU computing.

One advantage of the rigid gang model is that the interference caused by shared resource and intra-task parallelism can potentially be quantified better, thus reducing the worst-case execution time of the gang. Within a gang, co-scheduling of memory accesses and computation is possible, which can also potentially reduce the worst-case execution time of the gang. Specifically, one strict view of this is the RT-Gang model by Ali and Yun [1], in which all processors are allocated to a gang at the same time.

The computational complexity of the rigid gang scheduling problem was studied back in 1980s. Specifically, it has been shown that finding the optimal schedule for the rigid gang scheduling problem is *NP*-hard in the strong sense even when all the tasks have the same period and the same deadline [25]. Even simpler cases, like three machines [9] or unit execution time per task [22] are also shown to be *NP*-hard in the strong sense.

To schedule a set of *ordinary* periodic [27] or sporadic [29] real-time tasks on a multiprocessor platform, three paradigms have been widely adopted: partitioned, global, and semi-partitioned multiprocessor scheduling. A comprehensive survey can be found in [15]. For the rigid gang scheduling problem, the three scheduling paradigms are slightly modified and called *stationary*, *global*, and *semi-stationary* (rigid) gang scheduling. The *stationary gang* scheduling paradigm statically assigns a gang task to a set of processors, in which the cardinality of the set is equal to the gang size of the task. After this assignment is done, a gang task is only eligible to be executed on stationary processors assigned to it. The *semi-stationary* scheduling paradigm allows a gang task to execute on any subset of processors within a given set of processors that is larger than the gang size itself. That is, it allows a job of the gang task to migrate from one subset of processors to another subset of the given processors at any time. The *global* rigid gang scheduler allows a gang task to migrate to any available set of processors as long as the gang size constraints are met.

Note that when the gang size is 1 for each task (i.e., tasks are not executed in parallel and are ordinary periodic or sporadic tasks), the stationary, global, and semi-stationary gang scheduling paradigms correspond to the partitioned, global, and semi-partitioned multiprocessor scheduling paradigms, respectively.

In real-time systems, rigid gang scheduling has been mostly studied under global earliest-deadline-first (EDF) scheduling, in which the set of processors used by a gang task is not fixed and can be dynamically relocated at runtime, e.g., [16, 24, 30]. Specifically, in [24], the authors

extended Baruah's [3] multiprocessor global EDF analysis for ordinary sporadic real-time tasks to deal with global EDF gang scheduling, which has been disproved by Richard et al. [30]. The only valid analysis for global EDF gang scheduling is from Dong and Liu [16] and restricted to implicit-deadline sporadic real-time rigid gang task systems. They provide two utilization-based analyses, one optimized and one approximated.

Goossens and Richard [20] studied fixed-priority scheduling for the rigid gang scheduling problem for implicit-deadline periodic real-time task systems. They presented two algorithms, one based on linear programming and another based on a heuristic algorithm, providing exact and sufficient schedulability tests. Moreover algorithms based on deadline partitioning (DP-Fair) for periodic gang systems have been proposed. However the many preemptions of DP-Fair make this algorithm impractical and the complexity of the proposed algorithms is high especially for a large number of processors. The authors themselves discuss the problems to extend their algorithms to sporadic job arrival sequences due to its non-determinism.

For classical multiprocessor scheduling, it has been recently shown that global static-priority scheduling [31] and global EDF as well as global FIFO scheduling [8] are dominated by partitioned scheduling under state-of-the-art efficient sufficient schedulability tests, e.g., [6, 21]. The main reason is due to the inherited pessimism in those tests, which all stem from the work by Baker [2]. Hence, they all use carry-in interference to compensate the lack of a critical instant theorem and divide the higher-priority interference by the number of processors, i.e., they have a multiplicative factor of  $1/M$  in the corresponding analyses. We note that the factor  $1/M$  also appears in the schedulability tests in [16].

**Contributions:** In this paper we explore *stationary gang scheduling* for a set of sporadic real-time tasks with constrained deadlines (i.e., the relative deadline of a task is no more than its minimum inter-arrival time) on a homogeneous symmetric multiprocessor system consisting of  $M$  processors. We develop the corresponding schedulability analyses for fixed-priority scheduling and a heuristic algorithm for stationary gang assignments.

The contributions of this paper are as follows:

- We present schedulability tests for stationary gang assignments for constrained-deadline sporadic real-time tasks in Section 3. To the best of our knowledge, this is the first schedulability analysis that is capable of verifying the schedulability of sporadic constrained-deadline gang task systems, whilst the analysis in [16] is limited to implicit-deadline sporadic real-time rigid gang task systems and the algorithm in [20] is limited to implicit-deadline periodic tasks. Our success is due to the observation of self-suspension behavior in Section 3.2 and the recent improvement of optimizations and analyses for dynamic self-suspension task behavior [12, 13].
- We propose a class of partitioning algorithms based on the concept of consecutive stationary gang assignment in Section 4. Furthermore, we show that consecutive stationary gang assignments yield beneficial theoretical properties that can be used to upper-bound the worst-case interference suffered by any task according to the ratio of gang sizes of two tasks.
- In Section 5, we compare our algorithm to the state-of-the-art schedulability analysis for global EDF by Dong and Liu [16] by evaluation synthetically generated sporadic real-time task systems with implicit deadlines. The evaluation results show that our algorithm outperforms the algorithm by Dong and Liu [16]. Furthermore, we conducted evaluations for constrained-deadline task systems and observe reasonable schedulability.

## 2 System Model and Stationary GANG Scheduling

In this paper we consider a symmetric multiprocessor (SMP) system composed of  $M$  identical processors and analyze the response-times of a gang task set with constrained deadlines using our proposed stationary gang scheduler.

We consider a set  $\mathbb{T} = \{\tau_0, \tau_1, \dots, \tau_{n-1}\}$  of  $n$  constrained-deadline gang tasks to be scheduled on a set  $\mathbb{P} = \{P_0, P_1, \dots, P_{M-1}\}$  of  $M$  identical processors using fixed-priority rigid gang schedulers under the additional constraint of stationary gang assignments. Each task has a fixed-priority that is inherited by each instantiated job. We use  $\pi_i$  to denote the priority of task  $\tau_i$  and say  $\tau_j$  has higher priority than  $\tau_i$  if and only if  $\pi_j > \pi_i$ . We assume that no two tasks have the same priority, i.e., there are sufficient priority levels. Moreover, each task is assigned and restricted to a subset of processor, namely its stationary gang assignment, to execute on. This subset does not change in time, i.e., it is rigid. Throughout this section, we will assume that a stationary gang assignment is given for each task and revisit the problem to generate provably good stationary assignments in Section 4.

► **Definition 1.** *A sporadic constrained-deadline gang task  $\tau_i$  is defined by  $(C_i, E_i, D_i, T_i)$  and releases an infinite number of task instances, called jobs. Each job of a task releases a gang of  $E_i$  sub-jobs with worst-case execution time  $C_i$ , that have to be executed in parallel. That is, either all  $E_i$  sub-jobs are scheduled simultaneously or none is. Hence, a total workload of  $E_i \cdot C_i$  has to be executed in the time interval between job release and deadline. The period  $T_i$  denotes the minimal inter-arrival time of two jobs of  $\tau_i$  and each task has a relative deadline  $D_i \leq T_i$ . Moreover, the utilization of a gang task is given by  $U_i = E_i \cdot C_i / T_i$ .*

This means that when a job of  $\tau_i$  is released at time  $t$ , the subsequent job of  $\tau_i$  must be released not earlier than at time  $t + T_i$ . Furthermore, to fulfill its timing constraints, this job must be able to finish its execution not later than its absolute deadline at time  $t + D_i$ . The response time of a job of  $\tau_i$  is its finishing time minus its release time, and the worst-case response time  $R_i$  of task  $\tau_i$  under a given scheduling policy is the maximum response time of any job of  $\tau_i$  for any job arrival sequence possible according to the parameters of tasks in  $\mathbb{T}$ .

We now define stationary gang assignment and the related schedules.

► **Definition 2.** *A stationary gang assignment  $A_i \subseteq \{P_0, P_1, \dots, P_{M-1}\}$  of a gang task  $\tau_i$  is a subset of processors of size  $|A_i| = E_i \leq M$ , that are assigned to execute jobs of task  $\tau_i$ .*

In order to formalize the properties of a fixed-priority stationary gang scheduler, we first formalize the definition of an arbitrary schedule.

► **Definition 3.** *A schedule  $\sigma_{P_q} : \mathbb{R} \mapsto \mathbb{T} \cup \{\perp\}$  for a processor  $P_q$  with  $q \in \{0, \dots, M-1\}$  is a mapping from the continuous time domain to the task that is executed at time  $t$  or to  $\perp$  if the processor idles, i.e.,*

$$\sigma_{P_q} : \mathbb{R} \mapsto \mathbb{T} \cup \{\perp\}, \quad \sigma_{P_q}(t) = \begin{cases} \tau_i & \text{if task } \tau_i \text{ is executed on } P_q \text{ at time } t \\ \perp & \text{if } P_q \text{ is idle at time } t \end{cases} \quad (1)$$

Despite that a realistic schedule does not perform context switch arbitrarily, e.g., due to granularity determined by the system tick duration, our analysis can in general be applied in the continuous time domain. A stationary gang schedule is described as follows.

► **Definition 4.** *A schedule for a multiprocessor system satisfies the stationary gang property if for each task  $\tau_i$  and its stationary gang assignment  $A_i$ , the following property holds:*

$$\bigwedge_{P_q \in A_i} (\sigma_{P_q}(t) = \tau_i) \quad \text{if and only if } \tau_i \text{ is scheduled at time } t \quad (2)$$

Whenever we argue about schedules that satisfy the stationary gang property, we for example write  $\sigma_{A_i}(t) = \tau_i$ , if task  $\tau_i$  is scheduled on all the processors in  $P_q \in A_i$  at time  $t$ . Throughout this paper, we say that a gang task  $\tau_i$  is *active* at time  $t$  if a job of  $\tau_i$  is released and not yet finished. The stationary gang scheduler then schedules all active tasks  $\tau_i$  that are the highest-priority tasks with respect to all other tasks that use processors in  $A_i$ .

► **Example 5.** Consider the stationary gang schedule illustrated in Figure 1 with two tasks  $\tau_k$  and  $\tau_i$  with the stationary gang assignments  $A_k = \{P_2, P_3\}$  and  $A_i = \{P_1, P_2, P_3\}$ . Moreover, let  $\pi_k < \pi_i$ . Therefore  $\tau_i$  that releases a job at time 1 preempts task  $\tau_k$ . Whenever  $\tau_i$  is preempted on  $A_i$ ,  $\tau_k$  is the highest-priority task amongst all tasks that compete for processors  $P_2$  and  $P_3$  and is thus scheduled. ◀

### 3 Schedulability Test for Stationary Gang Scheduling

This section presents the schedulability test for stationary gang scheduling, provided that each task  $\tau_i$  has a predefined stationary gang assignment  $A_i$ . How to achieve good stationary gang assignments is discussed in Section 4. Throughout this section, our analysis focuses on the analysis to validate whether task  $\tau_k$  can meet its deadline constraint, provided that the tasks with higher priorities than  $\tau_k$  are validated beforehand. Hence, the validation of schedulability iterates from the highest-priority task to the lowest-priority task in  $\mathbb{T}$ .

Towards this, we present methods to analyze the contention between a higher-priority task  $\tau_i$  and the task  $\tau_k$  under analysis in Section 3.1. Specifically, our result shows that  $\tau_i$  can be considered as a self-suspending task under certain circumstances. Due to this observation that some higher-priority tasks can be transformed into self-suspending tasks, we employ existing suspension-aware schedulability analysis and present our schedulability test for stationary gang scheduling in Section 3.2.

#### 3.1 Contention Analysis

The preemptive fixed-priority stationary gang scheduler always schedules the active task  $\tau_k$  that has the highest priority with respect to all other tasks that use processors in  $A_k$ .

► **Definition 6.** *The contention domain  $\delta(A_k)$  of a set of processors  $A_k$  is defined as*

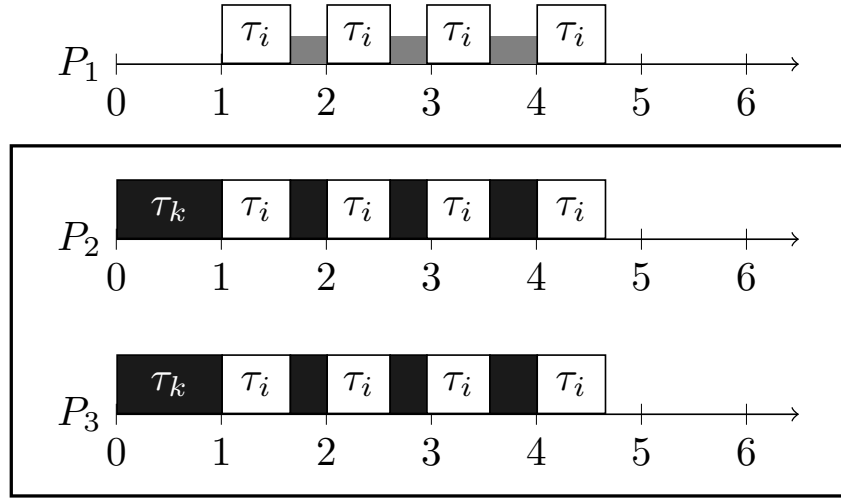
$$\delta(A_k) := \{\tau_\ell \in \mathbb{T} \mid A_k \cap A_\ell \neq \emptyset\} \quad (3)$$

Based on this behavior, we can formalize the condition for a higher-priority task  $\tau_i$  to be able to interfere with a task  $\tau_k$  ( $\pi_i > \pi_k$ ) as follows

$$\tau_i \text{ interferes with } \tau_k \iff \tau_i \in \delta(A_k) \quad (4)$$

This is simply due to the fact that a task  $\tau_i$  is able to preempt another task  $\tau_k$  if and only if it starts to be executed on a processor  $P_q$  on which  $\tau_k$  is assigned. In such a case,  $P_q \in A_k$  and  $P_q \in A_i$  and in conclusion  $P_q \in A_k \cap A_i$ , i.e.,  $\tau_i \in \delta(A_k)$ .

As a consequence, the schedulability of gang task  $\tau_k$  can be reduced to the schedulability of a single job with worst-case execution time  $C_k$  that is subjected to the maximum interference by jobs of tasks in  $\delta(A_k)$ . In the remainder of this subsection, we show that the interfering behavior of task  $\tau_i$  in  $\delta(A_k)$  can be over approximated by the interference behaviour of a corresponding sequential task with dynamic self-suspension behavior, where the suspension-time depends on the stationary gang assignments of the interfering tasks.



■ **Figure 1** An illustration of the suspension behavior of task  $\tau_k$  from the point of view of the task  $\tau_i$ . The gray boxes denote interference due to other higher-priority tasks on processor  $P_1$ .

► **Definition 7** (Dynamic Self-Suspension [13, 14]). *A task is said to have dynamic self-suspension behavior if an active task can transition from a ready state into a suspended state, in which the task is exempted from the scheduling decisions, and resume into a ready state at any time. The cumulative amount of time that an active task  $\tau_i$  can spend in a suspended state is upper-bounded by a parameter  $S_i$ .* ◀

The link between stationary gang schedules and dynamic self-suspension behavior can be illustrated in the following example.

► **Example 8.** Assume an arbitrary fixed-priority gang schedule for two tasks  $\tau_k, \tau_i$  with  $\pi_k < \pi_i$  and a stationary gang assignments  $A_k = \{P_2, P_3\}$  and  $A_i = \{P_1, P_2, P_3\}$  as shown in Figure 1. We analyze the execution of task  $\tau_k$  solely from the perspective of the processors specified in  $A_k$ , i.e.,  $P_2$  and  $P_3$ . Due to the arrival of the higher-priority task  $\tau_i$  at time  $t = 1$ ,  $\tau_k$  is preempted. However, execution on a processor not in  $A_k$  interferes with the execution of  $\tau_i$ . Whenever  $\tau_i$  is preempted by some interfering tasks on  $P_1$  (denoted by the gray boxes),  $\tau_k$  is scheduled on its assigned processors as described in the definition of the stationary gang scheduling paradigm. Hence, if we only analyze the execution of  $\tau_k$  with respect to its assigned processors, then transparent preemption of  $\tau_i$  equates to self-suspending behavior that needs to be accounted for in the response-time analysis of  $\tau_k$ . ◀

In the following, we formalize and explain how these task model substitutions can be safely obtained. Before moving into the formal proof, we present the conditions that hold for our scheduling policy.

► **Definition 9.** *A task  $\tau_j$  is executed at time  $t$  if and only if*

1. *Task  $\tau_j$  is active at time  $t$ .*
2. *There exists no task  $\tau_\ell \in \delta(A_j)$  with higher priority, i.e.,  $\pi_\ell > \pi_j$ , such that  $\tau_\ell$  is executed at time  $t$ .*

For further clarification, assume that we are interested in the response-time of task  $\tau_k$  and thus analyze the interference caused by higher-priority tasks that use some processors in  $A_k$ . Assume that  $\tau_i$  is active and has the highest priority among all active jobs that use

some processors in  $A_k$  at time  $t$ , but is interfered by a higher-priority task  $\tau_\ell \in \delta(A_i)$  (e.g., the grey boxes in Figure 1). From the perspective of task  $\tau_k$  this job is *self-suspended* and is resumed when the interfering task  $\tau_\ell$  releases the processor. More specifically, we provide the following definition.

► **Definition 10.** A task  $\tau_i \in \delta(A_k)$  is in a suspended state at time  $t$  with respect to a task  $\tau_k$  under analysis if and only if

1. Task  $\tau_i$  is active at time  $t$ .
2. Task  $\tau_i$  has the highest priority among all active tasks on the processors in  $A_k$ , i.e.,  $\pi_i \geq \max \{\pi_j \mid \tau_j \in \delta(A_k) \text{ active at } t\}$
3. Task  $\tau_i$  is not executed.

We use the following definition to collect the set of tasks that may interfere with a higher-priority task  $\tau_i \in \delta(A_k)$  but not interfere with the task  $\tau_k$  under analysis.

► **Definition 11 (Self-Suspension Inducing Tasks).** The set of tasks that can induce self-suspending behavior of  $\tau_i$  when analyzing task  $\tau_k$  is denoted by

$$V_{i,k} = \{\tau_\ell \in \delta(A_i) \mid \tau_\ell \notin \delta(A_k) \text{ and } \pi_\ell > \pi_i\} \quad (5)$$

We now validate that only these tasks induce self-suspending behavior for  $\tau_i$ .

► **Lemma 12.** Suppose that task  $\tau_i$  is in a suspended state at time  $t$  with respect to a task  $\tau_k$  under analysis, then at least one task in  $V_{i,k}$  is executed at time  $t$ .

**Proof.** By Definition 10, (i)  $\tau_i$  is active at time  $t$ , (ii)  $\pi_i \geq \max \{\pi_j \mid \tau_j \in \delta(A_k) \text{ active at } t\}$ , and (iii)  $\tau_i$  is not executed. Due to Definition 9 and since (i) and (iii) hold, there exists some task  $\tau_\ell \in \delta(A_i)$  with  $\pi_\ell > \pi_i$  that is executed at time  $t$ . It remains to show that  $\tau_\ell \notin \delta(A_k)$ . Assume that  $\tau_\ell \in \delta(A_k)$ , then from (ii) follows that  $\pi_i > \pi_\ell$  which contradicts  $\pi_\ell > \pi_i$ . ◀

Now, we can provide a safe upper bound of the self-suspension time if  $V_{i,k}$  is not empty.

► **Theorem 13.** Suppose that  $\pi_i > \pi_k$  and  $R_i \leq D_i \leq T_i$ , where  $R_i$  is an upper bound on the worst-case response time of task  $\tau_i$ , which was already verified beforehand. The amount of time  $S_{i,k}$  that a job of an active task  $\tau_i$  self-suspends with respect to  $\tau_k$  is at most

$$S_{i,k} \leq \min \left\{ R_i - C_i, \sum_{\tau_j \in V_{i,k}} \left( 1 + \left\lceil \frac{R_i}{T_j} \right\rceil \right) \cdot C_j \right\} \quad (6)$$

**Proof.** Suppose that a job of  $\tau_i$  is released at time  $t_i$  and finished at time at  $t_i + \Delta$ . By the assumption,  $\Delta \leq R_i$ . Let  $f(t)$  be 1 if tasks  $\tau_k$  and  $\tau_i$  are both active at time  $t$  and  $\pi_i > \pi_k$  but task  $\tau_k$  is executed at time  $t$ ; otherwise  $f(t)$  is 0. Therefore, the amount of time that  $f(t)$  is set to 1 is the amount of time  $S_{i,k}$  that the job of task  $\tau_i$  self-suspends instead of preempting  $\tau_k$ . Therefore,  $S_{i,k}$  can be calculated by integrating the function  $f(t)$  from  $t_i$  to  $t_i + \Delta$ , i.e.,  $S_{i,k} = \int_{t_i}^{t_i + \Delta} f(t) dt$ .

Suppose the amount of time that the job of  $\tau_i$  suspends during  $t_i$  and  $t_i + \Delta$  is  $> R_i - C_i$  for contradiction. This implies that the job of  $\tau_i$  has only completed  $\Delta - R_i + C_i$  amount of computation. This violates the assumption that  $R_i$  is the worst-case response time of  $\tau_i$ .

In addition, the suspension behavior of  $\tau_i$  is in fact induced by the tasks in  $V_{i,k}$  when analyzing task  $\tau_k$ . By Lemma 12, we know that such interference can only come from tasks in  $V_{i,k}$ . Since  $R_j \leq T_j$  for every task  $\tau_j$  with  $\pi_j > \pi_k$ , we know that the amount of time that a task  $\tau_j$  is executed from  $t_i$  to  $t_i + \Delta$  is at most  $\left( 1 + \left\lceil \frac{\Delta}{T_j} \right\rceil C_j \right)$ . This can be proved by

showing that the jobs of  $\tau_j$  that are executed in the interval  $[t_i, t_i + \Delta)$  are (i) at most only one job released prior to  $t_i$ , and (ii) the amount of jobs that we get by releasing jobs after  $t_1$  as soon as possible.<sup>1</sup>

Summing all tasks in  $V_{i,k}$  together, we have

$$S_{i,k} = \sum_{\tau_j \in V_{i,k}} \left( 1 + \left\lceil \frac{\Delta}{T_j} \right\rceil C_j \right) \leq \sum_{\tau_j \in V_{i,k}} \left( 1 + \left\lceil \frac{R_i}{T_j} \right\rceil C_j \right)$$

where the inequality is due to the assumption that  $\Delta \leq R_i$ .

Putting these two safe conditions together, we reach the conclusion.  $\blacktriangleleft$

Note that the estimation in Theorem 13 may not be precise as it counts the higher-priority interference of  $\tau_j \in \delta(A_i)$  and  $\tau_j \in \delta(A_k)$  as the suspension time of  $\tau_i$  as well. This is in fact standard higher-priority interference as in uniprocessor systems.

The following corollary is a direct implication from Theorem 13.

**► Corollary 14.** *If  $V_{i,k}$  is empty, then task  $\tau_i$  does not have any self-suspension behavior, i.e.,  $S_{i,k} = 0$  when analyzing task  $\tau_k$ .*

**Proof.** This is because the right-hand side of Equation (6) is 0 under this condition.  $\blacktriangleleft$

### 3.2 Schedulability Analysis

After analyzing the link between the stationary gang scheduling problem and the dynamic self-suspension problem, we now construct a worst-case response time analysis and schedulability analysis for task  $\tau_k$ . We provide such a bound based on suspension-aware analyses on uniprocessor systems.

On the basis of Theorem 13 and Corollary 14, we can safely upper-bound the interference of task  $\tau_k$ . We first collect the higher-priority tasks that interfere with  $\tau_k$  in  $\Psi_k$ , i.e.,  $\Psi_k = \{\tau_i \mid \tau_i \in \delta(A_k) \wedge \pi_i > \pi_k\}$ . For every task in  $\Psi_k$ , we transform it to an equivalent dynamic self-suspension task as follows:

**► Definition 15.** *Let a sporadic gang task  $\tau_i \in \Psi_k$  be transformed to the corresponding self-suspending task  $\tau_i^{sus} = (C_i, D_i, T_i, S_{i,k})$  with the same  $C_i$ ,  $D_i$ , and  $T_i$  as for  $\tau_i$ , where*

$$\begin{cases} S_{i,k} = \min \left\{ R_i - C_i, \sum_{\tau_j \in V_{i,k}} \left( 1 + \left\lceil \frac{R_i}{T_j} \right\rceil \right) \cdot C_j \right\} & \text{if } V_{i,k} \neq \emptyset \\ S_{i,k} = 0 & \text{otherwise} \end{cases} \quad (7)$$

and  $V_{i,k}$  is defined as in Definition 11.

The set  $\Psi_k^{sus}$  is the set of all transformed tasks, i.e.,  $\Psi_k^{sus} = \cup_{\tau_i \in \Psi_k} \tau_i^{sus}$ .

**► Theorem 16.** *Suppose that all higher-priority tasks  $\tau_0, \tau_1, \dots, \tau_{k-1}$  with given stationary gang assignments  $A_0, A_1, \dots, A_{k-1}$  are already verified to be schedulable. A sporadic constrained-deadline gang task  $\tau_k$  with stationary gang assignment  $A_k$  is schedulable by the fixed-priority stationary gang scheduling algorithm if the worst-case response time of executing  $C_k$  time units (without suspending task  $\tau_k$ ) is at most  $D_k \leq T_k$  under the interference of  $\Psi_k^{sus}$  on one processor under the same priority assignment.*

<sup>1</sup> This is typically done with the concept of *carry-in* jobs. Since  $R_i \leq T_i$ , there is at most one carry-in job of  $\tau_j$  released before  $t_i$ .



**Proof.** Suppose that a job of task  $\tau_k$  is released at time  $t_k$  and there is no other job of  $\tau_k$  active at time  $t_k$ . From  $t_k$ , the schedule  $\sigma$  either executes  $\tau_k$  or higher-priority jobs on the processors in  $A_k$ . Therefore, the job of  $\tau_k$  is either executed or interfered by higher-priority tasks in  $\delta(A_k)$ . Hence, only tasks in  $\Psi_k$  have interference with  $\tau_k$ . The equivalence of the self-suspension behavior is due to Theorem 13. Therefore, the proof is complete.  $\blacktriangleleft$

We adopt the current sound state-of-the-art self-suspension aware uniprocessor schedulability analyses by Chen et al. [12] for gang-scheduling, hence the correctness follows directly from the related proofs in [12] and Theorem 16.

► **Corollary 17.** *By the statement in Theorem 16, a gang sporadic task  $\tau_k$  is schedulable by the stationary gang scheduling algorithm if*

$$\exists 0 < t \leq D_k, C_k + \sum_{\tau_i^{sus} \in \Psi_k^{sus}} \min\{C_i, S_{i,k}\} + \sum_{\tau_i^{sus} \in \Psi_k^{sus}} \left\lceil \frac{t}{T_i} \right\rceil \cdot C_i \leq t \quad (8)$$

► **Corollary 18.** *By the statement in Theorem 16, a gang sporadic task  $\tau_k$  is schedulable by the stationary gang scheduling algorithm if*

$$\exists 0 < t \leq D_k, C_k + \sum_{\tau_i^{sus} \in \Psi_k^{sus}} \left\lceil \frac{t + R_i - C_i}{T_i} \right\rceil \cdot C_i \leq t \quad (9)$$

► **Corollary 19.** *Suppose that there are  $k$  tasks in  $\Psi_k^{sus}$ , indexed from the highest priority to the lowest priority, i.e.,  $\tau_0^{sus}$  is the highest-priority task in  $\Psi_k^{sus}$ . By the statement in Theorem 16, a gang sporadic task  $\tau_k$  is schedulable by the stationary gang scheduling if there is a vector  $\vec{x} = (x_0, x_1, \dots, x_{k-1})$  with  $x_i \in \{0, 1\}$  such that*

$$\exists 0 < t \leq D_k, C_k + \sum_{\tau_i^{sus} \in \Psi_k^{sus}} \left\lceil \frac{t + Q_i(\vec{x}) + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil \leq t \quad (10)$$

where  $Q_i(\vec{x}) = \sum_{j=i}^{k-1} S_{j,k} \cdot x_j$ .

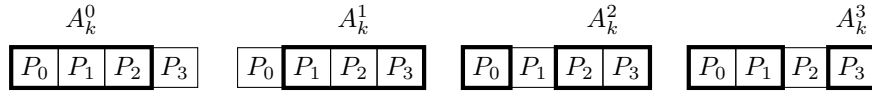
The provided schedulability analyses in Corollary 17, Corollary 18, and Corollary 19 can be evaluated using fixed-point iteration techniques. More precisely, let  $W_k(t)$  denote the left-hand sides of the inequalities in the above corollaries and  $\epsilon > 0$ , then we verify all test-points  $t_0 = W_k(\epsilon)$ ,  $t_1 = W_k(t_0)$ ,  $\dots$ ,  $t_n = W_k(t_{n-1})$  until convergence is reached or  $t_n > D_k$ . Due to the fact that the above equations are step-functions and can thus only change at discontinuity points of  $W_k(t)$ , the amount of test-points is at most  $k \cdot D_k / \min_{i < k} \{T_i\}$  resulting in pseudo-polynomial time-complexity. In the remainder of this paper, we only use  $\mathcal{O}(kD_k)$  for time-complexity, since the scaling of the deadline does not change the asymptotic complexity.

As reported in [12], neither of the schedulability analyses in Corollary 17 and Corollary 18 dominate each other analytically and are incomparable. The authors also showed that the test in Corollary 19 dominates those in Corollary 17 (i.e., Lemma 17 in [12]) and Corollary 18 (i.e., Lemma 16 in [12]). To efficiently find a vector  $\vec{x}$  for Corollary 19, they suggest to use three vectors, one is based on a linear approximation, one sets all elements of  $\vec{x}$  to 0, and one sets the  $x_i$  in  $\vec{x}$  to 1 if  $S_{i,k} \leq C_i$ , and 0 otherwise. Specifically, in the case when the entries in  $\vec{x}$  are all 0, Equation (19) is the same as Equation (18). In our evaluations we use Corollary 19 with the above three vectors and choose the best one, i.e., a task is determined schedulable if it is schedulable for at least one of the three vectors.

#### 4 GANG Assignment Algorithm

Since finding optimal schedules for the rigid gang scheduling problem is NP-hard in the strong sense even in the simplest settings, we seek for approximation algorithms to solve the gang assignment problem.

In fixed-priority stationary gang scheduling, next to priority assignments, the gang assignments determine the schedulability of the task set  $\mathbb{T}$ . A key problem in finding stationary gang assignments is the dependency of gang assignments and the resulting interference behaviour of higher-priority tasks. In general, each task  $\tau_k$  under consideration can have  $\binom{M}{E_k}$  many distinct gang assignments in terms of gang to processor mappings. However, for any given gang assignment of all higher-priority tasks, there may exist subsets of these distinct gang assignments, in which the interference of all higher-priority tasks of  $\tau_k$  is equivalent. A trivial example is the gang assignment of the first task, in which all gang assignments are equivalent, since there is no interfering tasks. In that case, all distinct gang assignments belong to the same equivalence class and any representative can be chosen for the gang assignment. However, finding all equivalence classes results in an exhaustive exploration of all possible solutions, which is computationally expensive especially for larger task sets.



■ **Figure 2** Consecutive stationary gang assignments  $A_k^0, A_k^1, A_k^2, A_k^3$  of a gang task  $\tau_k$  with  $E_k = 3$  on a system using 4 processors are generated by a sliding window.

We intend to identify a class of computationally feasible gang assignment algorithms that allow to formulate worst-case performance guarantees with respect to any optimal rigid gang scheduling algorithm. In order to get worst-case performance guarantees, it is mandatory to find (preferably small) upper bounds of interference caused by higher-priority tasks. Thus, instead of arbitrary gang assignments, we restrict ourselves to consecutive stationary gang assignments that allow to bound interference. We note however that other gang assignments can be explored starting from the consecutive assignments. By this, the approximation properties can be kept whilst improving the schedulability using any heuristic.

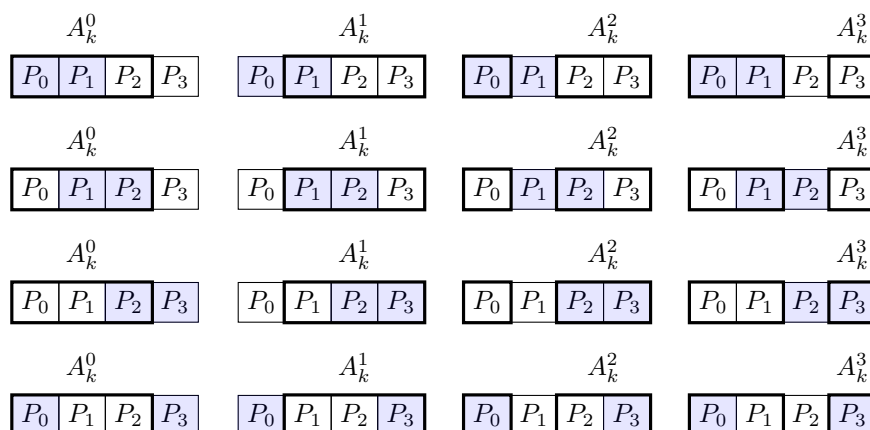
► **Definition 20.** A consecutive stationary gang assignment  $A_k^\ell$ ,  $\ell \in \{0, 1, \dots, M - 1\}$  for a gang task  $\tau_k$  in a system of  $M$  processors is a set of consecutive processor indices

$$\ell \bmod M, (\ell + 1) \bmod M, \dots, (\ell + E_k - 1) \bmod M \quad (11)$$

where  $|A_k^\ell| = E_k \leq M$ .

An example of consecutive stationary gang assignments of a task  $\tau_k$  with  $E_k = 3$  on a platform of 4 processors is illustrated in Figure 2. Intuitively, the consecutive stationary gang assignments are generated by a sliding window of length 3.

Another restriction in our algorithm is to devise gang assignments in priority-order under the premise that all higher-priority tasks are verified to be schedulable. By this restriction, we only have to determine the interference behaviour of each higher-priority task that only depends on the gang assignment  $A_k$  of task  $\tau_k$ .



■ **Figure 3** Enumeration of all consecutive stationary gang assignments of a task  $\tau_k$  (black window) under the condition of a given consecutive stationary gang assignment of a higher-priority task (light blue window).

The above restrictions yield the following two important theoretical properties:

1. There are always  $M$  different consecutive stationary gang assignments for each task.
2. Out of these  $M$  assignments, we are able to find upper-bounds for the number of consecutive stationary gang assignments of  $\tau_k$ , in which higher-priority tasks have self-suspension behaviour and non self-suspension behaviour respectively. That means, we are able to argue that in at most  $x$  out of the  $M$  consecutive stationary gang assignments, a higher-priority task  $\tau_i$  has self-suspension behaviour irrespective of the actual consecutive assignment of  $\tau_i$ .

In Figure 3, each column illustrates the  $M$  consecutive stationary gang assignments of  $\tau_k$ , that is subject to assignment and analysis, given a consecutive stationary gang assignment of a higher-priority task. Each row shows a different assignment of a higher-priority task  $\tau_i$  indicated by the light blue window. According to the discussion in Section 3,  $\tau_i$  interferes with  $\tau_k$  if and only if  $A_k \cap A_i$ , i.e., whenever the windows in Figure 3 intersect. If for a column (consecutive assignment of  $\tau_k$ ) there exists at least one row (consecutive assignment of  $\tau_i$ ) in which both windows intersect then  $\tau_i$  interferes with  $\tau_k$  for the consecutive assignment under consideration. In the provided example, all consecutive assignments suffer interference from  $\tau_i$ . In Lemma 21 we prove that there are at most  $E_k + E_i - 1$  out of the  $M$  consecutive stationary gang assignments of  $\tau_k$ , in which  $\tau_i$  interferes with  $\tau_k$ .

Moreover, guided by the observation that if  $A_k \subseteq A_i$  then  $\tau_i$  can not have self-suspension behaviour with respect to  $\tau_k$  under analysis, we can lower-bound the number of consecutive stationary gang assignments of  $\tau_k$  in which  $\tau_i$  can not exhibit self-suspension behaviour. For better illustration of this observation, assume that  $\tau_i$  has self-suspension behavior with respect to  $\tau_k$  then there exists a task  $\tau_\ell$  with higher priority than  $\tau_i$  (and subsequently higher priority than  $\tau_k$ ) such that  $A_\ell \cap A_i \neq \emptyset$  and  $A_\ell \cap A_k = \emptyset$ . This however implies that  $A_k \not\subseteq A_i$  and contradicts the assumption. This can only happen if  $E_i \geq E_k$  and if so then  $E_i - E_k$  many of the  $M$  consecutive stationary assignments satisfy this property. In the next two lemmas, we formally prove the intuition described above.

► **Lemma 21.** *Given a task  $\tau_k$  under analysis, each higher-priority task  $\tau_i$  causes interference, i.e.,  $\tau_i \in \delta(A_k)$ , in at most  $E_i + E_k - 1$  of the  $M$ -many consecutive stationary gang assignments.*

## 10:12 Hard Real-Time Stationary Gang-Scheduling

**Proof.** Let the consecutive stationary gang assignments for some higher-priority tasks  $i < k$  be given by the following processor indices:

$$j \bmod M, (j + 1) \bmod M, \dots, (j + E_i - 1) \bmod M \quad (12)$$

where  $j \in \{0, 1, \dots, M - 1\}$  is already given (fixed). Furthermore, let

$$\ell + h \bmod M, (\ell + h + 1) \bmod M, \dots, (\ell + h + E_k - 1) \bmod M \quad (13)$$

denote the processor indices of a consecutive stationary gang assignment of task  $\tau_k$  after the  $h$ -iteration for some arbitrary initial  $\ell \in \{0, 1, \dots, M - 1\}$  (we only need this to show that this works for an arbitrary initial position and can be set to 0 for comprehension). Then let  $h'$  denote the first iteration such that  $(\ell + h' + E_k - 1) \bmod M \equiv j - 1 \bmod M$  (we shift the window of  $A_k$  to the border of window of  $A_i$ , i.e., the two consecutive stationary gang assignments intersect in the next iteration for the first time). Therefore,

$$(\ell + h') \bmod M \equiv (j - E_k) \bmod M.$$

We have to iterate further  $z$  allocations until the index of the first processor in the allocation of  $\tau_k$ , i.e.,  $(\ell + h' + z) \bmod M \equiv (j + E_i - 1) \bmod M$  coincides with the index of the last processor in the assignment of task  $\tau_i$ . More formally, we seek to find the smallest  $z > 0$  such that:

$$\begin{aligned} (\ell + h' + z) \bmod M &\equiv (j + E_i - 1) \bmod M \\ ((\ell + h') \bmod M) + (z \bmod M) &\equiv (j + E_i - 1) \bmod M \\ (j - E_k + z) \bmod M &\equiv (j + E_i - 1) \bmod M \end{aligned}$$

which implies that  $z = E_i + E_k - 1$ , i.e.,  $z$  consecutive stationary gang assignments yield an intersection of both tasks.  $\blacktriangleleft$

We can furthermore bound the interference for self-suspending tasks as follows:

**► Lemma 22.** *For task  $\tau_k$  (under analysis), there are at most  $\min\{2E_k - 1, E_i + E_k - 1\}$  many consecutive stationary gang assignments, in which a higher-priority task  $\tau_i$  has self-suspension behavior with respect to task  $\tau_k$ .*

**Proof.** From Lemma 21, we know that at most  $E_k + E_i - 1$  many consecutive stationary gang assignments cause an intersection of consecutive stationary gang assignments of task  $\tau_i$  and task  $\tau_k$ . We hence subtract  $\max\{E_i - E_k, 0\}$ , namely the number of consecutive stationary gang assignments in which self-suspension behavior of  $\tau_i$  is impossible, from the above. Clearly, in the case that  $E_k \leq E_i$  we have  $(E_k + E_i - 1) - E_i + E_k = 2E_k - 1$ . Since  $2E_k - 1 \leq E_i + E_k - 1$  implies that  $E_k \leq E_i$  we can write it as  $\min\{2E_k - 1, E_i + E_k - 1\}$ .  $\blacktriangleleft$

For the rest of this paper, we used deadline-monotonic priority assignment and index the tasks such that  $D_1 \leq D_2 \leq \dots \leq D_n$ , in which  $\tau_i$  has a higher priority than  $\tau_k$  if  $i < k$ . Due to the additional restrictions described above, it is possible to prove interference bounds and in consequence approximation guarantees in terms of schedulability for any stationary gang assignment algorithm that uses the following algorithm as a basis.

We first sort the tasks according to the relative deadlines. Starting from the highest-priority task, we consider each of the possible stationary gang assignment candidates  $A_k^0, A_k^1, \dots, A_k^{M-1}$  and check whether it is feasible to assign task  $\tau_k$  to the consecutive

gang assignment. It starts from  $\ell = 0, 1, \dots, M - 1$ . If the consecutive gang assignment candidate  $A_k^\ell$  is feasible, we assign the gang task to the consecutive gang assignment; otherwise, we move to the next candidate. If none of the  $M$  possible consecutive gang assignments is feasible, this assignment step fails and the algorithm returns failure.

In Theorem 16, we assume that all stationary gang assignments  $A_i$  are given for all tasks with higher priority than  $\tau_k$ . Based on the information, we need to calculate  $V_{i,k}$ ,  $\delta(A_k)$ , and  $\delta(A_i)$  before using Theorem 16.

To facilitate efficient implementation, we use a matrix representation to indicate whether task  $\tau_i$  is assigned on processor  $P_j$ . Let  $\rho$  be a  $n \times M$  matrix in which

$$\rho(i, j) := \begin{cases} True & P_j \in A_i \\ False & P_j \notin A_i \end{cases}. \quad (14)$$

Given the stationary gang assignment matrix  $\rho$ , the algorithm constructs the interference matrix

$$\Gamma(i, j) := \begin{cases} True & A_i \cap A_j \neq \emptyset \\ False & \text{otherwise} \end{cases} \quad (15)$$

by the boolean matrix multiplication  $\rho \cdot \rho^T$ , where  $\rho^T$  is the transpose matrix of  $\rho$ . That is, the multiplication operation of two elements is replaced with the *logical and* operation and the addition operation of two elements is replaced with a *logical or* operation. More precisely, each entry in the interference matrix is computed as follows:

$$\Gamma(i, j) = \bigvee_{m=0}^{M-1} \rho(i, m) \wedge \rho(j, m)$$

which is true only if task  $\tau_i$  and task  $\tau_j$  share at least one processor in their stationary gang assignments. The asymptotic time-complexity for the matrix multiplication is given by  $\mathcal{O}(n^2M)$ . The space complexity is given by  $\mathcal{O}(nM)$ .

The transformation of the higher-priority tasks in  $\mathbb{T}$  into  $\Psi_k$ , which is later needed to construct  $\Psi_k^{sus}$ , can be done by the following operation:

$$\begin{cases} \tau_i \in \Psi_k & \text{if } \bigvee_{\ell=1}^{i-1} \Gamma(\ell, i) \wedge \overline{\Gamma(\ell, k)} \\ \tau_i \notin \Psi_k & \text{otherwise} \end{cases} \quad (16)$$

We now analyze the time-complexity of Algorithm 1. Line 4 requires  $\mathcal{O}(i)$  for each task  $\tau_i$  and therefore  $\mathcal{O}(k^2)$  for one iteration. Line 5 requires to calculate the right-hand side of Equation (6), which can be done in  $\mathcal{O}(1)$  if we only take  $R_i - C_i$  or  $\mathcal{O}(i)$  if both terms are evaluated in Equation (6) for a task  $\tau_i \in \Psi_k$ . Therefore, Line 5 in one iteration requires  $\mathcal{O}(k^2)$ . The schedulability test in Line 6 from Corollaries 17, 18 and 19 is  $\mathcal{O}(kD_k)$ . Line 7 is  $\mathcal{O}(M)$ . Since the loop can run up to  $\mathcal{O}(nM)$  iterations, the time complexity is  $\mathcal{O}(nM^2 + n^3MD_n)$ .

## 5 Evaluation

In this section, we present evaluations with synthetically generated gang task sets to evaluate our proposed algorithm (denoted as *OUR-DM* here) against the current state-of-the-art by Dong and Liu [16] for sporadic implicit-deadline gang task systems under global EDF. Specifically, we compare to the optimized schedulability test in [16], denoted as *DONG-OPT*, based on the acceptance ratio, i.e., the number of schedulable task sets compared to the number of tested task sets.

■ **Algorithm 1** Deadline-Monotonic Stationary GANG Schedulability Analysis and Assignment.

---

```

1: Sort task set  $\mathbb{T}$  such that  $D_i \leq D_j$  for  $i < j$  (ties are broken arbitrarily);
2: for  $k$  in  $\{1, 2, \dots, n\}$  do {Loop tasks.}
3:   for  $\ell \in \{0, 1, \dots, M - 1\}$  do {Loop candidates.}
4:     Generate  $\Psi_k$  given the candidate  $A_k^\ell$  from Def. 20;
5:     Transform  $\Psi_k$  to  $\Psi_k^{sus}$  using Def. 15;
6:     if  $(C_k, D_k, T_k) \cup \Psi_k^{sus}$  is schedulable according to any self-suspension aware uni-
       processor schedulability test (from Cor. 17, 18 and 19) then
7:       Assign  $A_k \leftarrow A_k^\ell$ ;
8:       break;
9:   return No feasible stationary gang assignments found;
10: return Feasible stationary gang assignment  $A_i$  for each task  $\tau_i$ ;

```

---

We also evaluate our algorithm for sporadic constrained-deadline gang task systems under different settings of gang sizes, but without comparison due to the absence of research results for constrained-deadline gang tasks. In these experiments, we seek to explore how much the imposed constraints in terms of stationary gang assignments and fixed-priority scheduling algorithms impact the schedulability of the tested task sets.

## 5.1 Experimental Setup

We generate synthetic task sets of sporadic gang tasks with implicit- and constrained-deadlines in the following way. To generate the task sets, we use the UUniFast algorithm [7] to draw  $n$  samples of  $x_i = E_i \cdot C_i / MT_i$  uniform at random where  $x_i \in (0, 1]$  such that  $\sum_{i=1}^n x_i = x$  for  $x \in \{0.05, 0.1, 0.15, \dots, 1\}$ . Moreover, the periods  $T_i$  are drawn from a log-uniform distribution in the range of [10, 100] ms.

The generated task sets are classified by the range of admissible gang sizes into *light*, *moderate*, and *heavy*. We differentiate two different settings for these gang sizes:

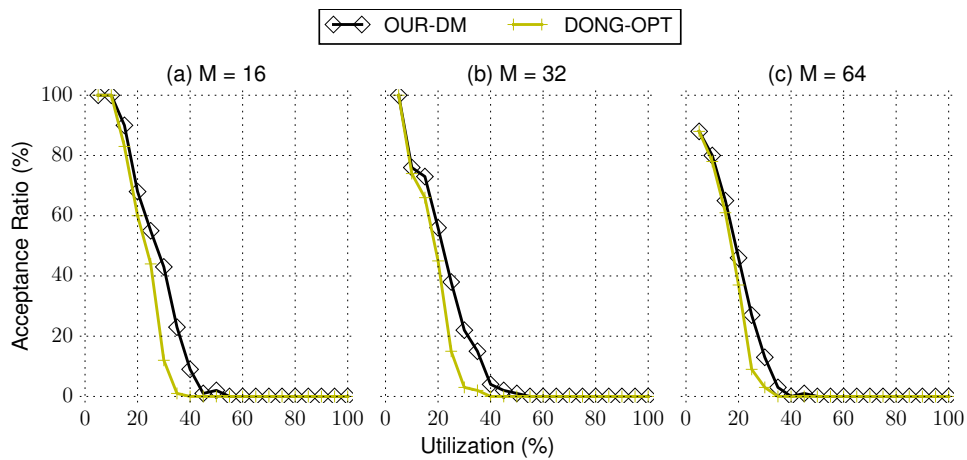
1. **Setting I** - with variable gang sizes: In the first setting, each *light* gang task can have a gang size in  $[1, M/8]$ , a *moderate* task can gang size in  $[1, M/4]$ , and a *heavy* task can have gang size in  $[M/8, M/2]$ .
2. **Setting II** - with fixed gang sizes: In this setting, a fixed gang size number is assigned to each task of a category. Namely, each *light* task has gang size  $M/8$ , each *moderate* task has gang size  $M/4$  and each *heavy* task has a gang size  $3M/8$ .

We avoid the generation of too heavy tasks, since in these cases the scheduling problem is degraded to uniprocessor scheduling.<sup>2</sup> With respect to constrained-deadlines, we only demonstrate our proposed algorithm by a case of variable gang sizes (Setting I) in Figure 7 and a case of fixed gang sizes (Setting II) in Figure 8.

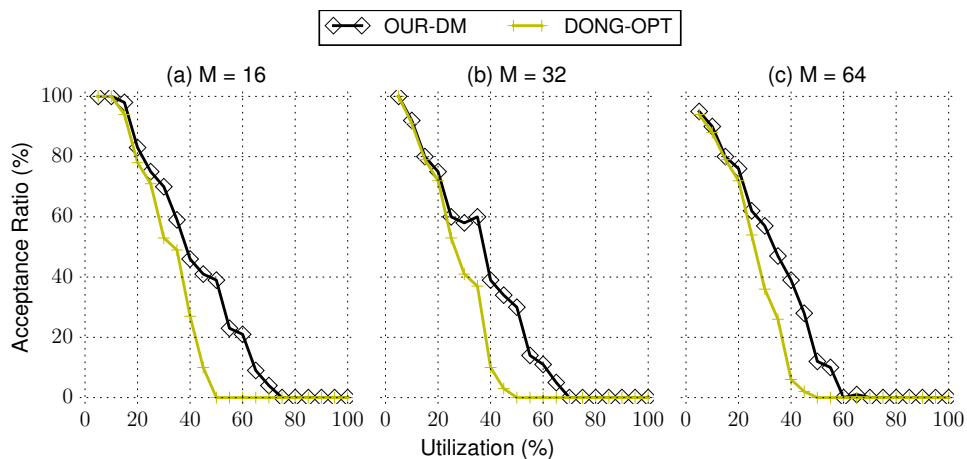
---

<sup>2</sup> Dong and Liu [16] also performed their evaluations for gang size in  $[5M/8, M]$  for all tasks. This configuration is not considered here as this setup implies that there is no possibility to concurrently execute two gang tasks in parallel due to the imposed gang size. The problem becomes equivalent to uniprocessor scheduling by viewing all processors as one virtual group. In this case, preemptive EDF is the optimal solution and the classical timing analysis for uniprocessor EDF scheduling can be applied.

## 5.2 Evaluation Results



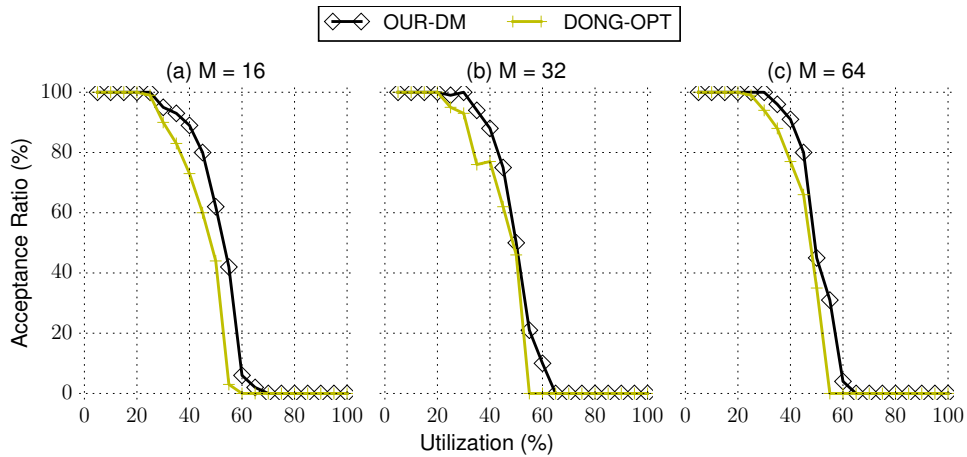
■ **Figure 4** Acceptance ratio for *light* sporadic implicit-deadline gang task sets where the gang size of each task is chosen according *Setting II*.



■ **Figure 5** Acceptance ratio for *moderate* sporadic implicit-deadline gang task sets where the gang size of each task is chosen according to *Setting I*.

### 5.2.1 Evaluation results for implicit-deadline task sets

For sporadic implicit-deadline gang task systems, we compare our algorithm (*OUR-DM*) with the approach by Dong and Liu [16] (*DONG-OPT*) under the setting with variable gang sizes, in which each configuration is evaluated with 100 task sets and 20 tasks for each task set. In all conducted experiments shown in Figures 4, 5, and 6, our algorithm *OUR-DM* outperforms *DONG-OPT* for all evaluated scenarios under the setting with variable gang sizes. The most significant improvement of *OUR-DM* compared with *DONG-OPT* is demonstrated for the *moderate* task set in Figure 5 where up to 40% can be achieved for 50% normalized utilization. The smallest improvement can be observed for *heavy* gang task sets, where *OUR-DM* slightly outperforms *DONG-OPT*. This is due to the fact that the heavier the task sets are, the more similar the schedulability is to the uniprocessor schedulability problem. This also implies



■ **Figure 6** Acceptance ratio for *heavy* sporadic implicit-deadline gang task sets where the gang size of each task is chosen according to *Setting I*.

that the stationary gang scheduling has less choices for gang assignments. Since EDF is an optimal uniprocessor schedulability, the trouble to deal with the heavy gang task sets comes from the adopted schedulability tests. For *OUR-DM*, we have to consider more tasks in  $\Psi_k$  and for *DONG-OPT* their analysis becomes less pessimistic as the multiplicative of  $1/M$  in their analysis decreases.

### 5.2.2 Evaluation results for constrained-deadline task sets

For constrained-deadlines, we show our schedulability test for light, moderate, and heavy task sets for gang sizes compliant to Setting I in Figure 7 and gang sizes compliant to the Setting II described in Figure 8, in which each configuration is tested with 100 task sets and 20 tasks per task set. The behavior of Setting I is almost similar to the results in Figures 4, 5, and 6 but with lower acceptance ratios.

For constrained-deadlines with fixed numbers of gang sizes as explained in Setting II, a similar trend can be observed. However, moderate as well as heavy task sets almost show the same acceptance ratio and the acceptance ratio of light tasks also increases. This further supports the assumption, that the increased number of tasks with self-suspension behaviour decreases the overall schedulability. This is explained by the fact that it is less likely to have self-suspension behaviour of interfering tasks if all tasks have the same gang size.

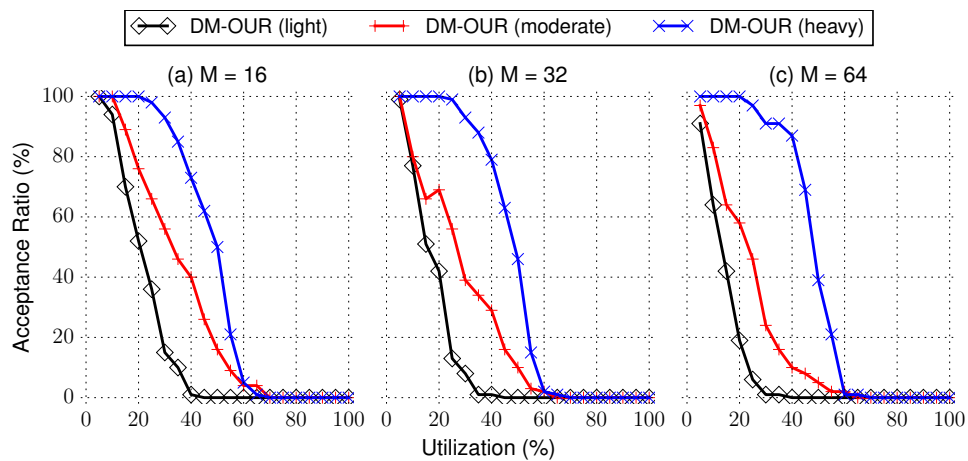
## 5.3 Summary of Evaluation Results

In summary, the evaluations demonstrate, that the restriction of fixed-priority stationary gang scheduling does not significantly sacrifice the schedulability of sporadic implicit-deadline rigid gang task systems, in comparison to the state-of-the-art. In contrast, the schedulability could be improved slightly without even considering performance benefits of implementations in real systems, e.g., reduced context switches and migrations.

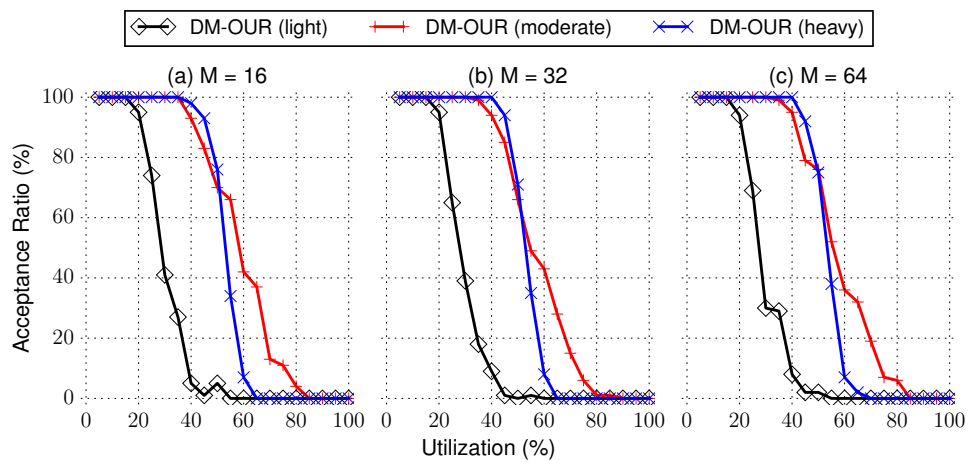
## 6 Conclusion and Future Work

In this paper we propose a specialization of the rigid gang scheduling problem for hard real-time systems. We present how this problem can be analyzed and reduced to the uniprocessor self-suspension problem and the schedulability analyses thereof. We show how to derive





■ **Figure 7** Acceptance ratio for *light* sporadic constrained-deadline gang task sets according to *Setting I*. The deadline is chosen randomly between 70% – 100% of the minimum inter-arrival time.



■ **Figure 8** Acceptance ratio for *light, moderate, heavy* sporadic constrained-deadline gang task sets according to *Setting I*. The deadline is chosen randomly between 70% – 100% of the minimum inter-arrival time.

stationary gang assignments for deadline-monotonic gang scheduling that yields worst-case interference bounds proportional to parameters defined by the ratios of the gang sizes of tasks in the task set.

This paper is limited to constrained-deadline task systems, as there is no result known for schedulability analyses for arbitrary-deadline dynamic self-suspending task systems. The concept in this paper can be extended to EDF by adopting the proper schedulability tests and suspension analysis. As future work, we plan to implement a fixed-priority stationary gang scheduler in real-time operating systems and evaluate if there are significant benefits in terms of scheduling overheads and investigate potential benefits with respect to improved worst-case execution time.

## References

- 1 W. Ali and H. Yun. RT-Gang: Real-time gang scheduling framework for safety-critical systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 143–155, 2019. doi:10.1109/RTAS.2019.00020.
- 2 Theodore P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 120–129, 2003. doi:10.1109/REAL.2003.1253260.
- 3 Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 119–128, 2007.
- 4 Sanjoy Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1323–1328, 2015. URL: <http://dl.acm.org/citation.cfm?id=2757121>.
- 5 Sanjoy Baruah. Federated scheduling of sporadic DAG task systems. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS*, pages 179–186, 2015. doi:10.1109/IPDPS.2015.33.
- 6 Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium (RTSS)*, pages 149–160, 2007. doi:10.1109/RTSS.2007.31.
- 7 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005. doi:10.1007/s11241-005-0507-9.
- 8 Alessandro Biondi and Youcheng Sun. On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling. *Real Time Syst.*, 54(3):515–536, 2018. doi:10.1007/s11241-018-9303-1.
- 9 J. Błażewicz, P. Dell’ Olmo, M. Drozdowski, and M.G. Speranza. Corrigendum to: Scheduling multiprocessor tasks on three dedicated processors. *Inf. Process. Lett.*, 49(5):269–270, 1994.
- 10 Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. Feasibility analysis in the sporadic dag task model. In *ECRTS*, pages 225–233, 2013.
- 11 Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio C. Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *RTSS*, pages 421–433. IEEE Computer Society, 2018.
- 12 Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 61–71, 2016.
- 13 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019. doi:10.1007/s11241-018-9316-9.
- 14 Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*, pages 1–10. IEEE Computer Society, 2017. doi:10.1109/RTCSA.2017.8046321.
- 15 Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011. doi:10.1145/1978802.1978814.
- 16 Zheng Dong and Cong Liu. Analysis techniques for supporting hard real-time sporadic gang task systems. In *IEEE Real-Time Systems Symposium, RTSS*, pages 128–138, 2017. doi:10.1109/RTSS.2017.00019.
- 17 Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.
- 18 José Fonseca, Geoffrey Nelissen, and Vincent Nélis. Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017. doi:10.1145/3139258.3139288.

- 19 José Carlos Fonseca, Geoffrey Nelissen, Vincent Nélis, and Luís Miguel Pinho. Response time analysis of sporadic DAG tasks under partitioned scheduling. In *SIES*, pages 290–299. IEEE, 2016.
- 20 Joël Goossens and Pascal Richard. Optimal scheduling of periodic gang tasks. *LITES*, 3(1):04:1–04:18, 2016. doi:10.4230/LITES-v003-i001-a004.
- 21 Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *IEEE Real-Time Systems Symposium*, pages 387–397, 2009.
- 22 J.A. Hoogeveen, S.L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math.*, 55(3):259–272, 1994.
- 23 Morris A. Jette. Performance characteristics of gang scheduling in multiprogrammed environments. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing, SC '97*, 1997.
- 24 Shinpei Kato and Yutaka Ishikawa. Gang EDF scheduling of parallel task systems. In *IEEE Real-Time Systems Symposium, RTSS*, pages 459–468, 2009. doi:10.1109/RTSS.2009.42.
- 25 M. Kubale. The complexity of scheduling independent two-processor tasks on dedicated processors. *Inf. Process. Lett.*, 24(3):141–147, 1987.
- 26 Karthik Lakshmanan, Shinpei Kato, and Ragnathan (Raj) Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium, RTSS '10*, pages 259–268, 2010.
- 27 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- 28 Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems. In *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems*, 2015.
- 29 Aloysius K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.
- 30 Pascal Richard, Joël Goossens, and Shinpei Kato. Comments on "gang EDF schedulability analysis". *CoRR*, <http://arxiv.org/abs/1705.05798>, 2017. URL: <http://arxiv.org/abs/1705.05798>.
- 31 Youcheng Sun and Marco Di Natale. Assessing the pessimism of current multicore global fixed-priority schedulability analysis. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC*, pages 575–583. ACM, 2018.
- 32 Saud Wasly and Rodolfo Pellizzoni. Bundled scheduling of parallel real-time tasks. In *RTAS*, pages 130–142. IEEE, 2019.