Utility-Based Scheduling of (m, k)-firm Real-Time Tasks – New Empirical Results

Florian Kluge[*]

Department of Computer Science, University of Augsburg, Germany
fkuau@gmx.net

Abstract

The concept of a firm real-time task includes the notion of a firm deadline that should not be missed by the jobs of this task. If a deadline miss occurs, the concerned job yields no value to the system. For some applications domains, this restrictive notion can be relaxed. For example, robust control systems can tolerate that single executions of a control loop miss their deadlines, and still yield an acceptable behaviour. Thus, systems can be designed under more optimistic assumptions, e.g. by allowing overloads. However, care must be taken that deadline misses do not accumulate. This restriction can be expressed by the model of (m, k)-firm real-time tasks that require that from any k consecutive jobs at least m are executed successfully. In this article, we extend our prior work on the MKU scheduling heuristic. MKU is based on history-cognisant utility functions as means for making decisions in overload situations. We present new theoretical results on MKU and other schedulers for (m, k)-firm real-time tasks. Based on extensive simulations, we assess the performance of these schedulers. The results allow us to identify task set characteristics that can be used as guidelines for choosing a scheduler for a concrete use case.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms
Keywords and phrases Real-time Scheduling, (m, k)-Firm Real-Time Tasks
Digital Object Identifier 10.4230/LITES-v004-i001-a002
Received 2016-01-25 **Accepted** 2016-12-28 **Published** 2016-12-31

1 Introduction

Certain types of real-time systems can tolerate that some jobs miss their deadlines or are not executed at all. This allows to dimension the system more optimistically. Sporadically arising overload conditions are resolved either by deferring or cancelling some jobs. Consider, for example, the decoding of a video stream. If single frames are displayed too late, the quality a viewer experiences degrades, but he still can draw some benefit. Similarly, control systems can also tolerate some job losses due to their robustness. To convey the notion of such systems with relaxed real-time constraints into real-time scheduling, Jensen et al. [19] and Locke [33] replaced the binary notion of deadlines with more expressive *time-utility functions* (TUFs) and proposed a scheduler based on *earliest deadline first* (EDF) [32]. A TUF describes the value or utility a system can draw from a job execution if it is finished by a certain time, thus increasing the flexibility of real-time systems.

A problem in TUF-based real-time scheduling is that each job is viewed independently. Therefore, no guarantees can be given about the distribution of deadline misses or job cancellations (both termed *losses* in the following) for single tasks. It may even happen that jobs of a specific task are never executed [24]. Considering the above examples, it is obviously necessary that losses of jobs do not accumulate and thus a simple *Quality-of-Service* (QoS) metric is not sufficient to describe the available tolerances. Special concepts have been developed in scheduling theory

[*] Florian Kluge is now with Elektronische Fahrwerkssysteme GmbH.

that allow to constrain the distribution of deadline misses, for example the skip-over model [27], (m, k)-firm real-time tasks [18], the dynamic window-constrained scheduler [44], or weakly-hard real-time tasks [4]. All these approaches consider not only single jobs, but also the execution history of the related tasks.

Our aim is to exploit the flexibility of TUF-based real-time scheduling while concurrently providing (m, k)-firm real-time guarantees. We use *history-cognisant utility functions* (HCUFs) [24] derived from TUFs to convey a task's state to a TUF-based real-time scheduler. A *history-cognisant utility function* (HCUF) represents the utility a task has accumulated with respect to the execution of past jobs. In our previous work [25] we have proposed a heuristic algorithm for *utility-based scheduling of (m, k)-firm real-time tasks* (MKU). The MKU algorithm is an extension of Jensen et al.'s [19, 33] EDF-based scheduler. Our results in [25] show the feasibility of MKU and that it can achieve good results when compared to other schedulers for (m, k)-firm real-time tasks.

In this article, we make the following contributions: First, we report new theoretical properties on preemptive scheduling of (m, k)-firm real-time tasks. A schedulability test for tasks using fixed (m, k)-patterns is presented. Additionally, we examine the phenomenon of breakdown anomalies, where increasing the utilisation of an infeasible task set can lead to feasibility. Also, new results on the schedulability of MKU are reported. Second, we present new results of extensive simulations that enable us to assess different schedulers for (m, k)-firm real-time tasks more clearly. We use arbitrary task sets to examine the overall performance of different schedulers, the feasibility of schedulability tests, and the initialisation of a task's execution history. In further simulations, we examine tasks sets where task periods and (m, k)-constraints are restricted to practically relevant ranges.

The remainder of this article is structured as follows: In the following Section 2, we define the basic concepts used throughout this paper. Related work on TUF-based scheduling and scheduling of (m, k)-firm real-time tasks is presented in Section 3. In Section 4, we present new properties of (m, k)-firm schedules which we use in our evaluations. In Section 5, we introduce our evaluation methodology. Evaluation results are shown and discussed in Section 6. We conclude this article in Section 7.

2 Fundamentals

2.1 Task Model

An (m, k)-firm real-time task is a tuple $\tau_i = (C_i, T_i, m_i, k_i)$ with *worst-case execution time* (WCET) C_i, period T_i and (m, k)-constraint (m_i, k_i). All numbers are integers. For simulation, we assume that a task's execution time is constant. In reality, the actual execution time of a task may be lower than its WCET. We account for this fact in our simulation through the use of abstract task sets and the generation of concrete task sets for different utilisations (see Section 2.2). Tasks are initially released at time $t = 0$, i.e. the task set is synchronous, and have implicit deadlines $D_i = T_i$. Thus, jobs $\tau_{i,j}$ are generated at times $r_{i,j} = jT_i, j = 0, 1, \ldots$ and must be finished until $d_{i,j} = (j + 1)T_i$ to avoid deadline misses. Each job is subject to a firm real-time requirement: If the job is not finished by its deadline, its result is useless and the job is cancelled. In this work, we consider the preemptive scheduling of jobs on a single processor. Thereby, we assume that jobs can be scheduled independently and that no resource constraints exist. If more than one job is eligible for dispatching at a certain time, e.g. due to identical priorities, then the scheduler chooses the job with the earliest activation time. If there are still multiple eligible jobs, an arbitrary one is chosen.

A task τ_i's (m, k)-constraint is defined by (m_i, k_i), meaning that in any k_i jobs released consecutively at least m_i must be finished before their deadline. An (m, k)-firm real-time task incurs a *dynamic failure* if less than m out of k consecutive jobs meet their deadline. More formally, this can be expressed using the concept of a *k-sequence* of a task. Let $\sigma_i^j \in \{0,1\}$ denote the status of the j-th job of τ_i with $\sigma_i^j = 0$ representing a deadline miss or job cancellation, and $\sigma_i^j = 1$ standing for successful execution. Then, τ_i's state or *k-sequence* after execution of the j-th job is a string $\sigma_i = (\sigma_i^{j-k+1}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$ with $\sigma_i^l \in \{0,1\}^k$. New job states σ_i^j are shifted into σ_i from the right. The (m,k)-constraint requires that a task τ_i's k-sequence always contains at least m_i 1s. A task's *distance* from dynamic failure is the number of jobs that consecutively would have to miss their deadlines such that the task's (m,k)-constraint is no longer kept.

2.2 Abstract and Concrete Task Sets

Abstract task sets (ATSs) form the basis of the simulations presented in this article. An ATS $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is a set of abstract tasks $\alpha_i = (e_i, T_i, m_i, k_i)$ with periods T_i, (m,k)-constraints (m_i, k_i) and execution time weights e_i. A concrete task set $\tau(\alpha, U_\mathrm{T})$ is derived from an ATS α by calculating the tasks' execution times C_i such that the CTS approximates a target utilisation of U_T. The execution time weight e_i specifies, how much task τ_i contributes to the task set utilisation:

$$\frac{e_i}{\sum_{j=1}^{n} e_j} = \frac{U_i}{U_\mathrm{T}} \tag{1}$$

Thereby, $U_i = \frac{C_i}{T_i}$ is the utilisation of the task under consideration. Solving with eq. (1) for C_i yields:

$$C_i = \frac{U_\mathrm{T}}{\sum_{j=1}^{n} e_j} T_i e_i \tag{2}$$

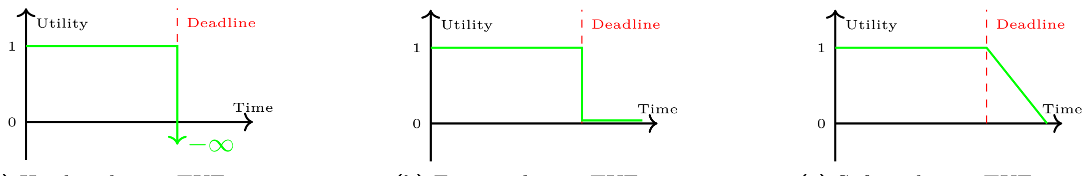
We consider only integral execution times in our simulations. How we obtain these will be clarified in Section 5.1.

3 Related Work

3.1 TUF-based Scheduling

The concept of time-utility functions was originally introduced by Jensen et al. [19] and Locke [33]. Instead of basing task scheduling solely on the binary notion of a deadline, the use of TUFs allows for a greater flexibility. A TUF describes the utility a system gains when a job is finished until a certain time. Some TUFs for well-known real-time constraints are shown in Figure 1. Hard real-time jobs (Figure 1a) must be finished by their deadline. Exceeding hard deadlines can result in catastrophic consequences which is expressed by the value $-\infty$ after the deadline. In contrast, firm real-time jobs (Figure 1b) do not yield any utility when exceeding their deadline. The entries in the k-sequence of a (m,k)-firm real-time task can be seen as the results of a firm real-time TUF. Soft real-time jobs may exceed their deadlines and still yield a decreasing utility to the system, which is shown in Figure 1c. TUFs are not restricted to these shapes. Thus, TUFs define a generic interface for the specification of task timing requirements. They allow to integrate tasks with different timing requirements in a system using only a single scheduler.

Jensen et al. [19] demonstrate the benefit of TUFs by extending EDF scheduling for overloaded task sets. If a high probability for a deadline miss is detected that would render the EDF schedule infeasible, jobs that contribute only with a low value-density (ratio of utility/value to execution time) to the system are selectively cancelled. Thus, schedulability of the system is ensured and



(a) Hard real-time TUF. **(b)** Firm real-time TUF. **(c)** Soft real-time TUF.

Figure 1 Exemplary TUFs.

accumulated utility is maximised. In literature, this approach is often referred to as *Locke's best-effort scheduling algorithm* (LBESA). Based on LBESA, Clark [11] has developed the *dependent activities scheduling algorithm* (DASA) for tasks with dependencies. Davis et al. [12] proposed an *adaptive threshold policy* for the admission of jobs, which has a lower overhead than LBESA. The notion of *dynamic value density* [1] reduces cancellations of jobs that have already started execution. Li and Ravindran [30] presented the MLBESA and MDASA algorithms that mimic the behaviour of LBESA and DASA, but come with lower complexities. TUF-based approaches have often been proposed to handle transient overloads in real-time systems [3, 26, 6, 35, 34, 13].

Many works on scheduling based on TUFs can also be found under the term *utility accrual* scheduling. The aim in utility accrual scheduling is to maximise the utility that is accrued through the execution of tasks. Insofar, the values and shapes of TUFs are a central criterion for scheduling. Chen and Muhlethaler [7] have shown that the problem of maximising value through arrangement of jobs/tasks is NP-hard. They also proposed an heuristic scheduling algorithm with a complexity of $O(n^3)$. The *utility accrual packet scheduling algorithm* by Wang and Ravindran [43] for packet scheduling in switched Ethernet comes with a lower complexity of $O(n^2)$, but is restricted to unimodal non-increasing TUFs. In contrast, the *resource-constrained utility accrual* algorithm by Wu et al. [45] can handle arbitrary TUFs and resource constraints at a complexity of $O(n^2 \log n)$. The *generic utility scheduling* algorithm by Li et al. [31] can also deal with mutual exclusion constraints, although with higher complexities of $O(n^3)$ for dispatching and $O(n^4 r)$ for scheduling. Tidwell et al. [42] model the scheduling problem as a Markov decision process that is solved offline and yields an optimal solution. The solution is used to generate a lookup table that is evaluated by an online scheduler in linear complexity. Also, works exist that investigate the use of TUF-based scheduling on multiprocessor systems [41, 10, 9, 39]. Here, especially the work of Rhu et al. [39] on the *global multiprocessor utility accrual scheduling algorithm for (m, k)-firm deadline-constrained multimedia streams* (gMUA-MK) algorithm is interesting, as they aim to schedule tasks with (m, k)-firm deadlines and TUFs on multiprocessors.

3.2 (m, k)-firm Real-Time Tasks

The concept of real-time tasks with (m, k)-firm deadlines was originally proposed by Hamdaoui and Ramanathan [18]. They present a scheme for *distance-based priority* (DBP) assignment of newly generated jobs for fixed-priority scheduling. Using this scheme, the priority of a job is set depending on the task's distance from dynamic failure. Jobs that belong to a task with a short distance are assigned higher priorities. The corresponding calculations are based on the task's k-sequence (see Section 2.1). Goossens [16] points out two properties of the DBP approach and devises an exact schedulability test. The first property concerns the initialisation of the k-sequences: Goossens shows that the string $\sigma_i = 1^k$ is not optimal under DBP and may yield to an infeasible schedule. In contrast, using an error state to initialise σ_i can result in a feasible schedule. The second property is the periodicity of feasible DBP schedules. Scheduling decisions under DBP only depend on the (m, k)-constraints and k-sequences σ_i of the tasks, whose space is bounded. Let $P = \text{lcm}\{T_i \mid i = 1, \ldots, n\}$ be the hyperperiod of the task set $\tau = \{\tau_i = (C_i, T_i, m_i, k_i) \mid i = 1, \ldots, n\}$. As τ is a synchronous task set with implicit deadlines, at times $B = kP, k \in \mathbb{N}$ all jobs that were activated before t are finished, and each task releases a new job. Thus, the system is in the same state each $B = kP$, and only the k-sequences of the tasks may differ. For any task τ_i, $\sum_{j=m_i}^{k_i} \binom{k_i}{j}$ distinct k-sequences exist. The period of a feasible DBP schedule is bounded by

$$F = \prod_{i=1}^{n} \sum_{j=m_i}^{k_i} \binom{k_i}{j} \times P \tag{3}$$

as any combination of tasks and their k-sequences must be considered. Thus, if a task set τ is feasible in the interval $[0, F)$, i.e. no (m, k)-constraint is violated, then τ is always feasible. The exact schedulability test for τ consist of executing or simulating τ for this time interval and checking whether the (m, k)-constraints of all tasks are always kept. Once an (m, k)-constraint is violated, the test stops and returns that τ is not feasible. The test can be sped up by evaluating the *system state* $\sigma = (\sigma_1, \ldots, \sigma_n)$ consisting of all tasks' k-sequences at each hyperperiod boundary $B = kP, k \in \mathbb{N}$. If a certain system state σ recurs, simulation can immediately be stopped, as the schedule will repeat itself and thus is feasible. We will term this optimised test in the following as *Goossens' schedulability test* (GST).

The seminal work of Hamdaoui and Ramanathan [18] has sparked a number of further works on the scheduling of (m, k)-firm real-time tasks. Ramanathan uses the concept of (m, k)-firm real-time tasks for the specific use case of control systems [38]. A deterministic classification into mandatory and optional jobs is proposed based on static (m, k)-patterns. Mandatory jobs are scheduled with their original, e.g. rate-monotonic [32] priority while optional jobs get the lowest possible priority. In the following, we will refer to this approach as *evenly distributed (m, k)-patterns* (MKP). A set of (m, k)-firm real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is considered feasible, if at least all mandatory jobs can be executed successfully. A schedulability test is based on the fact that the first instance $\tau_{i,0}$ of any task τ_i is always classified as mandatory. Ramanathan [38] provides a sufficient schedulability condition. However, this condition contains a timing non-deterministic term which makes it hard to evaluate [20]. Jia et al. [20] propose a schedulability test, which basically implements the response time analysis [2] for the first job of any task heeding the (m, k)-patterns. For task sets with harmonic periods, the test provides exact results, for all other task sets it is only sufficient.

Quan an Hu [37] note that the classification according to [38] introduces a high pessimism into the schedulability analysis, as at time $t = 0$ a mandatory job from any task in a task set gets ready. They relieve this critical instant by introducing spin or rotation values s_i that rotate the (m, k)-patterns of each task τ_i by s_i places. Quan and Hu propose a heuristic algorithm for finding good spin parameters, and also examine the use of a genetic algorithm for the determination of spin parameters. In this work, we will use the heuristic algorithm under the term *evenly distributed (m, k)-patterns with spin values* (MKP-S). As the original presentation of the algorithm in [37] is missing important information, we apply the corrections that we describe in [22]. Spin parameters are also considered by Semprebom et al. [40] for global time slot allocation in wireless real-time networks. Additionally, the authors propose an online schedulability/admission test.

Flavia et al. [14] extend the work of Ramanathan [38] on the use of (m, k)-firm real-time tasks for control of plants. They present a method to determine offline an optimal k_i parameter for a given controller and calculate controller parameters for all $m_i \in [1 \ldots k_i]$. Depending on the actual plant state during runtime, optimal m_i are chosen and the controller parameters are set appropriately.

Cho et al. devise the *guaranteed dynamic priority assignment* (GDPA) scheme [8]. It is based on EDF scheduling, but additionally takes the tasks' distance from a failing state into account. Its aim is (1) to provide a bounded probability of violations of the (m, k)-firm constraints, and (2) to maximise the probability of kept deadlines. Under GDPA, ready jobs are not directly inserted into the EDF schedule, instead they are kept in ready list. Any time the schedule needs to be adjusted, each job's distance from dynamic failure is calculated. The EDF schedule is created by considering the jobs in increasing order of their distance from dynamic failure, i.e. critical jobs are preferred, and inserting them into the EDF schedule. If an insertion makes the schedule infeasible, the job is removed again from the schedule. Jobs in the ready list that are infeasible are cancelled. In the same paper, the *simplified guaranteed dynamic priority assignment* (GDPA-S) is proposed. It works similar to GDPA, but has a lower runtime complexity. GDPA-S keeps ready jobs in two lists, one in EDF order and another ordered in increasing distance from dynamic failure. Dispatching is performed either from the head of the EDF list, if the EDF schedule is feasible. Else, the most critical job (at the head of the second list) is dispatched. Again, infeasible jobs are cancelled immediately. Concerning non-preemptive scheduling of (m, k)-firm real-time tasks, the work on Matrix-DBP [36] provides necessary schedulability conditions. One that can also be applied to preemptive scheduling is based on the minimum load that is generated by a set of (m, k)-firm real-time tasks $\tau = \{\tau_1, \ldots \tau_n\}$:

$$U_{\mathrm{mk}} = \sum_{i=1}^{i=n} \frac{m_i}{k_i} \frac{C_i}{T_i} \tag{4}$$

The calculation of U_{mk} assumes that only m_i out of k_i jobs of any task τ_i are executed. If

$$U_{\mathrm{mk}} > 1 \tag{5}$$

then τ is not feasible.

An approach similar to the concept of (m, k)-firm real-time tasks is proposed by Gettings et al. [15] for mixed-criticality systems with weakly-hard constraints. Their *adaptive mixed criticality – weakly hard* algorithm can skip up to s out of m consecutive jobs to reduce the load from low-critical tasks in a high-criticality mode, while still ensuring a guaranteed QoS for low-criticality tasks.

3.3 The MKU Algorithm

In a previous publication [25], we have presented the *utility-based scheduling of* (m, k)*-firm real-time tasks* (MKU) algorithm that is based on HCUFs [24, 23]. In the following, we give a brief outline of its functionality, please refer to [25] for more details. MKU is based on LBESA, the main difference is the utility function that is used in the decisions about job cancellations. We use a HCUF [24, 23], that maps the execution history of a task into a single scalar value and additionally provides a prediction about the tasks future utility. A task τ_i's current utility H_{m} after the completion or cancellation of its j-th job is the arithmetic mean of its current k-sequence (see Section 2.1). The value is additionally scaled by $\frac{k_i}{m_i}$ to enable an easy comparison between tasks with different (m, k)-constraints:

$$\hat{H}_{\mathrm{m}}(\tau_i, j) = \frac{k_i}{m_i} \frac{1}{k_i} \sum_{l=0}^{k+1} \sigma_i^{j-l} = \frac{1}{m_i} \sum_{l=0}^{k_i-1} \sigma_i^{j-l} \tag{6}$$

Through the scaling, any task τ_i has the requirement that $\hat{H}_m(\tau_i, j) \geq 1$. For scheduling decisions, we use a task τ_i's potential utility $\hat{H}_{\mathrm{P}}$ under the assumption that the currently active job $\tau_{i,j}$ is cancelled and ignore the least recent job $\tau_{i,j-k_i+1}$:

$$\hat{H}_{\mathrm{p}}(\tau_i, j) = \frac{1}{m_i} \sum_{l=0}^{k_i-2} \sigma_i^{j-l} \tag{7}$$

If an overload situation occurs, the scheduler cancels jobs that have maximum $\hat{H}_{\mathrm{p}}$ values. To ensure the adherence of (m, k)-constraints, only jobs with $\hat{H}_{\mathrm{p}} > 1$ can be cancelled. If no job for cancellation can be found, the task set is infeasible.

As MKU is based on LBESA, it inherits the algorithm's complexity of $O(n^2)$ in overload situations. This is similar to the GDPA, GDPA-S and gMUA-MK approaches, but higher than for DBP and the pattern-based MKP/MKP-S schemes. However, unlike these schemes, MKU and the underlying LBESA are not restricted to (m, k)-firm real-time tasks. They allow the integration of tasks with other requirements in the same system, as long as these requirements can be expressed in terms of TUFs/HCUFs.

4 New Properties of (m, k)-firm Real-Time Tasks

In the following, we present some properties of (m, k)-firm real-time task sets that have not yet been reported in literature. First, we present exact schedulability conditions for approaches based on fixed (m, k)-patterns (Section 4.1) and for the MKU scheme (Section 4.2). Finally, we report our observations on scheduling anomalies in Section 4.3.

4.1 Feasibility of Approaches Based on Fixed (m, k)-Patterns

For (m, k)-firm real-time task sets that use fixed (m, k)-patterns defined in [38] (MKP), Jia et al. give a sufficient schedulability test [20]. As the test is only sufficient, it may reject some task sets that are actually feasible. Also, it cannot be applied to the MKP-S [37] approach, as the rotation of the patterns might move the critical instant of the task set. Nevertheless, an exact schedulability test can be derived for task sets that used fixed (m, k)-patterns. The derivation of this test is similar to the one for DBP scheduling [16] and uses the same preconditions: (1) The scheduling algorithm must be *deterministic*, and (2) it must be *memory-less*. In the context of fixed (m, k)-patterns, the second precondition means that the algorithm's decisions at any time depend only on static properties of the active tasks. In contrast to [16], the current k-sequence of a task has no influence on the schedule.

The exact schedulability test follows from the periodicity of schedules when using fixed (m, k)-patterns.

▶ **Theorem 1.** *Let a set of synchronous (m, k)-firm real-time tasks be scheduled by a fixed-priority scheduler. Priorities are derived from fixed (m, k)-patterns that classify jobs into mandatory and optional. Then the schedule is periodic with period*

$$P = \mathrm{lcm}\{k_i T_i \mid i = 1 \ldots n\}. \tag{8}$$

Proof. The priorities of jobs of a single task τ_i are derived from a fixed (m, k)-pattern of length k_i. Thus, each k_i jobs, i.e. after $k_i T_i$ cycles, the priority pattern recurs. For any two tasks τ_i, τ_j, the job and priority pattern generated by both recurs after $\mathrm{lcm}\{k_i T_i, k_j T_j\}$ cycles, as after this times both tasks' are in the same state as at the beginning (concerning their patterns). Via induction, this argument can be extended to n tasks $\tau_1, \ldots, \tau_n$. ◀

For a task set where the sufficient test by Jia et al. [20] does not indicate feasibility or where the test is not applicable (MKP-S), it suffices the simulate the schedule for at most P cycles (eq. (8)).

4.2 Schedulability under MKU

The schedulability test devised by Goossens [16] for task sets under DBP scheduling also applies to the MKU scheduler in terms of the upper bound for simulation. The test is solely based on the tasks' periods and (m, k)-constraints, and the fact that the DBP scheduler itself is memoryless. The MKU scheduler itself does not possess an internal state. Like DBP, it acts solely on the states of the tasks, namely their k-sequences to calculate a task's HCUF.

When applying GST for MKU scheduling, a further slight optimisation is possible. GST examines the system state $\sigma = (\sigma_1, \ldots, \sigma_n)$ after each hyperperiod. A repetition of σ without any task violating its (m, k)-constraint means that the schedule is cyclic and valid, as the schedule itself solely depends on σ. In its calculation of the possible HCUF $\hat{H}_\mathrm{p}$ (eq. (7)), the MKU scheduler only regards the most recent $k_i - 1$ entries of each σ_i. Insofar, it operates on a *reduced system state*:

▶ **Definition 2.** Let $\sigma^R = (\sigma_1^R, \ldots, \sigma_n^R)$ be the *reduced system state* of a set of (m, k)-firm real-time tasks. σ_i^R is obtained from a system state $\sigma_i = (\sigma_i^{j-k+1}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$ by ignoring the least recent entry, i.e. $\sigma_i^R = (\sigma_i^{j-k+2}, \ldots, \sigma_i^{j-1}, \sigma_i^j)$.

The following theorem provides the basis for an optimisation of GST for the HCUF-based scheduler:

▶ **Theorem 3.** *If during the execution of GST with MKU a reduced system state σ^R recurs at a hyperperiod boundary, a cycle in the MKU schedule has been found.*

Proof. Let σ^1, σ^2 be two system states incurred in this order during execution of GST with MKU, such that for the derived system states $\sigma^{R,1} = \sigma^{R,2}$ (in the following simply σ^R. Further, let $L(\sigma) = (\sigma_1^{j-k+1}, \sigma_2^{j-k+1}, \ldots, \sigma_n^{j-k+1})$ be the vector of a system state σ's least recent entries that are ignored by σ^R. We can distinguish two cases:

1. If $L(\sigma^1) = L(\sigma^2)$, then also $\sigma^1 = \sigma^2$. The whole system state recurred, a cycle in the schedule has been found.
2. If $L(\sigma^1) \neq L(\sigma^2)$, then there exists at least one i such that $\sigma_i^{1,j-k_i+1} \neq \sigma_i^{2,j-k_i+1}$. However, the schedule S_1 produced by MKU between σ^1 and σ^2 solely depends on $\sigma^{R,1}$, as MKU regards only the σ_i^R for its cancellation decisions. Thus, MKU will produce the same schedule $S_2 = S_1$ after σ^2, as $\sigma^{R,1} = \sigma^{R,2}$. ◀

In case 1, a real cycle has been found, as is also detected by GST. Case 2 needs some closer inspection, as it can help to speed up the schedulability test:

▶ **Corollary 4.** *Let σ^1, σ^2 be two system states with $\sigma^{R,1} = \sigma^{R,2}$, $L(\sigma^1) \neq L(\sigma^2)$ (case 2 in the proof of Theorem 3), $\sigma^R := \sigma^{R,1} (= \sigma^{R,2})$, and σ^2 occurs after σ^1. Further, let σ^3 be the next system state with $\sigma^{R,3} = \sigma^R$. Then, $L(\sigma^3) = L(\sigma^2)$ and $\sigma^3 = \sigma^2$.*

Proof. Recall that the schedule S_1 produced by MKU after σ^1 only depends on $\sigma^{R,1} = \sigma^R$. Thus, MKU will produce the same schedule $S_2 = S_1$ after σ^2. As the decisions for S_1 and the result $\sigma^{R,2}$ are solely based on $\sigma^{R,1}$, S_2 (taking the same decisions) will produce the same result $\sigma^3 = \sigma^2$, and thus $L(\sigma^3) = L(\sigma^2)$. ◀

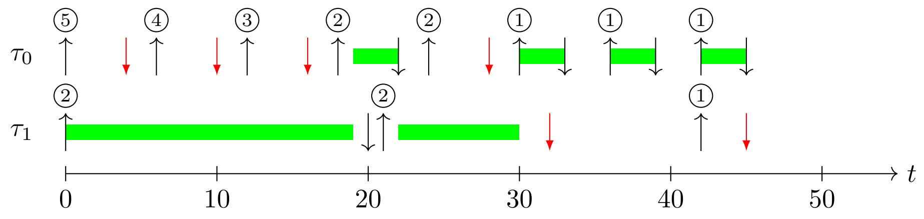
This means that the schedulability test for MKU **may** terminate at least one hyperperiod earlier compared to GST, depending on the length of the cycle.

4.3 Breakdown Anomalies

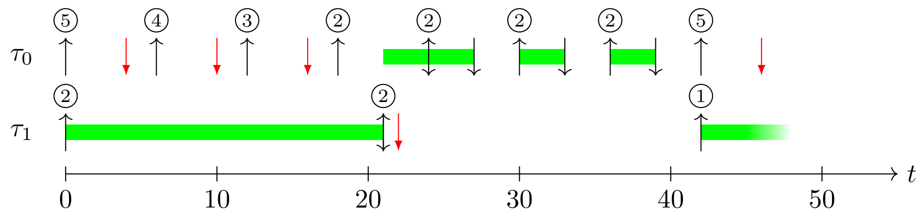
Consider an ATS (see Section 2.2), from which multiple CTSs are derived with increasing utilisations U. In hard real-time scheduling, it is possible to identify a *breakdown utilisation* [29] for an ATS,

Table 1 This ATS exhibits a breakdown anomaly at the target utilisations $U_\mathrm{T} = 1.45$ and $U_\mathrm{T} = 1.55$ when scheduled with DBP.

Task	T_i	e_i	(m,k)	$C_i^{1.45}$	$C_i^{1.55}$
τ_0	6	55	$(4,8)$	3	3
τ_1	21	95	$(1,2)$	19	21



(a) $U_\mathrm{T} = 1.45$, Task 1 violates its (1,2)-constraint at time 45.



(b) $U_\mathrm{T} = 1.55$, task set is feasible.

Figure 2 Example schedules with breakdown anomaly; for task parameters refer to Table 1; circled numbers (ⓧ) denote DBP of job; red arrows (↓) indicate job cancellations.

beyond which the derived CTSs are no longer feasible. Unfortunately, this method does not yield exact results for (m,k)-firm real-time tasks. Increasing the utilisation of a (m,k)-firm real-time task set farther beyond the breakdown utilisation can actually lead to the task set being feasible again. To make the point more clearly, consider an ATS α with a breakdown target utilisation U_B, and two constants $s_\mathrm{I} > 0, s_\mathrm{F} > 0, s_\mathrm{I} < s_\mathrm{F}$. This means that the derived CTS with target utilisation U_B is feasible, but the CTS with target utilisation $U_\mathrm{B} + s_\mathrm{I}$ is not. However, it may happen that the CTS with target utilisation $U_\mathrm{B} + s_\mathrm{F}$ is feasible again.

An exemplary ATS α that exhibits such a behaviour is shown in Table 1. We assume that both tasks' k-sequences are initialised with 1^{k_i}. The anomaly arises for target utilisations $U_\mathrm{T} = 1.45$ and $U_\mathrm{T} = 1.55$ (actual execution times are rounded to integer values). Figure 2a shows the DBP schedule for $\tau(\alpha, 1.45)$. Numbers in circles indicate the distance-based priority of each released job. Please refer to [38] for the calculations involved. Lower numbers indicate higher priorities. In time step 45, $\tau_{1,2}$ is cancelled, thus violating τ_1's $(1,2)$-constraint. Even if the scheduler cancelled the current instance of τ_0 instead, the schedule would still be infeasible, as then τ_0's $(4,8)$-constraint would be violated later. Increasing U_T for α leads to a feasible DBP schedule, as is shown in Figure 2b. In this case, the segment between $t = 0$ and $t = 42$ is repeated periodically.

The breakdown anomalies are the result of the special structure of the DBP scheme. Scheduling decisions made at a certain time do not only depend on static properties of the tasks. Instead, they are also influenced by the internal states of the tasks, which in turn depend on the past scheduling decisions. Thus, they can also occur in the MKU, GDPA and GDPA-S algorithms. The consequence of such a behaviour is that these schedulers are not sustainable [5] with regard to the tasks' execution times. If a schedulability test shows that a certain task set is feasible, we cannot be sure that it stays feasible if execution constraints are relaxed by decreasing execution times.

Such anomalies cannot happen in task sets that are scheduled using fixed (m, k)-patterns, which we express in the following theorem:

▶ **Theorem 5.** *Let a set of (m, k)-firm real-time tasks τ^1 be derived from an ATS α for a given utilisation U_1. Further, assume that τ^1 is not feasible using fixed (m, k)-patterns. Then, any task set τ^2 derived from α for a utilisation $U_2 > U_1$ is also not feasible.*

Proof. Fixed (m, k)-patterns classify jobs into mandatory and optional jobs. A schedule S is considered feasible, if all mandatory jobs are executed successfully. Infeasibility of S means that at least one mandatory job $\tau_{i,j}$ misses its deadline at time $d_{i,j}$. Assume that $\tau_{i,j}$ is released at time $a_{i,j}$. Then, only mandatory jobs with priority higher than or equal to $\tau_{i,j}$'s priority are executed in the interval $[a_{i,j}, d_{i,j}]$. Increasing the utilisation of the task set means that the tasks' execution times are increased, but periods and thus activation times and deadlines remain unchanged. Thus, the processing time demanded by higher-priority jobs in $[a_{i,j}, d_{i,j}]$ can only further increase, and thus the deadline miss would occur also in the new task set. ◀

From this theorem follows that scheduling of tasks using fixed (m, k)-patterns is sustainable with regard to execution times.

5 Evaluation Methodology

We perform extensive simulations of randomly generated task sets to compare the scheduling approaches. The simulations are conducted using the `tms-sim` framework developed in our group [21], which is available as open source software. In this section, we present the methodology we apply in our evaluations.

5.1 Task Parameters

The parameters of the ATSs are generated using the `libc` pseudo-random number generator (`rand_r()`). For the task periods T_i, two approaches are implemented: During most simulations, the periods are chosen randomly from a given interval $\{T_{\min}, \ldots, T_{\max}\}$. Additionally, we have also implemented the period generator from Goossens and Macq [17]. This generator yields task periods that have many common divisors, and a limited hyperperiod compared to randomly chosen periods. Period generation is based on a matrix of multipliers, where each row contains powers of a prime number. Period generation randomly selects one entry from each row. The actual period then is the product of the chosen entries. For our simulations, we add a restriction that periods must be > 2 for any task.

Execution time weights are chosen from an interval $[1, e_{\max}]$ where $e_{\max}$ represents the granularity of the weights. Execution times C_i are calculated according to eq. (2). As we only consider integral execution times in this work, the actual execution time C'_i of a task τ_i is obtained from C_i through rounding. Additionally, we demand that no task has zero execution time:

$$C'_i = \begin{cases} [C_i], & \text{if } [C_i] > 0 \\ 1, & \text{else} \end{cases} \tag{9}$$

Thereby, the operation $[x]$ stands for regular rounding, i.e. returns the integer value that is nearest to x. Through the rounding, the task set's actual utilisation $U = \sum_{i=0}^{n} \frac{C_i}{T_i}$ can deviate from the target utilisation. Task set generation is configured such that generated ATSs that deviate more than a constant d_{U} from an initial target utilisation are automatically discarded.

Table 2 Task models, schedulers, and schedulability tests used in the experimental evaluation.

Model	Abbr.	Reference	Test
FPP Scheduler			
Distance-based priority	DBP	[18]	GST
Fixed (m, k)-patterns	MKP	[38]	[20], Sect. 4.1
MKP with pattern rotation	MKP-S	[37]	Sect. 4.1
EDF-based Schedulers			
Guaranteed Dynamic Priority Assignment	GDPA	[8]	GST
Simplified GDPA	GDPA-S	[8]	GST
Global Multiprocessor Utility Accrual scheduling for (m, k)-firm deadline-constraints	gMUA-MK	[39]	GST
Utility-based (m, k)-tasks	MKU	[25]	GST

The k_i parameters are chosen from an interval $\{k_{\mathrm{min}}, \ldots, k_{\mathrm{max}}\}$. For the m_i parameters, we have again implemented two approaches: Either, they can be chosen from $\{1, \ldots, k_i\}$. This approach can yield m_i values (compared to k_i) that can seem quite unrealistic. Therefore, we allow to limit the m_i to meaningful ranges. An additional parameter $r_{\mathrm{m}} \in [0, 1]$ can be specified to lower-bound m_i. The actual m_i parameter then is chosen from $\{\lceil r_{\mathrm{m}} k_i \rceil, \ldots, k_i\}$.

To the best of our knowledge, there is not yet an efficient way to find good intialisations for the k-sequences. Checking all possible initialisation values is not feasible, as $2^{\sum_{i=1}^{n} k_i}$ schedules would have to be examined. Therefore, we use 1^{k_i} as initial k-sequence, which might be as good or bad as any other (possibly random) choice.

5.2 Simulation

Simulations are performed using the exact schedulability tests. For MKP and MKP-S, we use the methods described by Jia et al. [20] (sufficient condition) and the one introduced in Section 4.1. For all other schedulers, GST [16] is applied. Simulation of a CTS is performed as deemed necessary by the schedulability tests. In the simulations, we search for the breakdown utilisations [29] of ATSs under different schedulers. This search works as follows: A single ATS is repeatedly used to generate CTSs. The first ATS is generated using a target utilisation $U_{\mathrm{T}} = U_{\mathrm{B}}$. If the CTS is found to be schedulable for a certain scheduler, U_{T} is increased by a utilisation step s_{U}. Using this updated U_{T}, a new CTS is derived from the ATS and another simulation is performed. This process is repeated until the CTS is no longer schedulable. The last U_{T} that yields a feasible CTS is called the *breakdown utilisation*. To account for breakdown anomalies (see Section 4.3), U_{T} is increased further and the derived CTSs are simulated, too. This process stops when the derived CTSs no longer fulfil the necessary schedulability condition $U_{\mathrm{mk}} \leq 1$ (see eq. (5)).

An overview of the task models and schedulers used for evaluation can be found in Table 2. For our simulations, we have adjusted the gMUA-MK approach to immediately cancel jobs that are removed from a schedule due an overload. Due to the assumption of constant execution times, they would be cancelled anyway. Like in the MKU approach, we use the TUF for firm real-time tasks (see Figure 1b).

Table 3 Parameters for task set generation and simulation.

Symbol	Description	Value
$\{T_{\text{min}}, \dots, T_{\text{max}}\}$	Range for task periods (ignored when Goossens' and Macq's period generator [17] is used, sect. 6.2.2, 6.2.3)	$\{5, \dots, 60\}$
e_{max}	Granularity of execution time weights	100
$\{k_{\text{min}}, \dots, k_{\text{max}}\}$	Range for the k_i parameter	$\{2, \dots, 10\}$
$\{m_{\text{min}}, \dots, m_{\text{max}}\}$	Range for m_i paramters	$\{2, \dots, k_i\}$
r_{m}	If specified, restricts m_i to $\{\lceil r_{\text{m}} k_i \rceil, \dots, k_i\}$ (only sect. 6.2.1, 6.2.3)	$\{0.1, 0.2, \dots, 0.9\}$
$U_{\text{T}} = U_{\text{B}}$	Target/base utilisation of generated task sets	1.05
d_{U}	Maximum allowed deviation from U_{T}	0.05
s_{U}	Utilisation step for breakdown utilisation search	0.01 / 0.1

5.3 Parameters & Aims of the Evaluation

5.3.1 Parameters

An overview of the parameters used for task set generation and simulation can be found in Table 3. They are passed to `tms-sim` via the command line or a parameter file. The period and k parameter ranges are chosen such as to be comparable with other works, e.g. [37]. The d_{U} parameter is only used during generation of an ATS. If the CTS generated for the base utilisation $U_{\text{B}} = U_{\text{T}}$ is not inside the interval $U_{\text{T}} \pm d_{\text{U}}$, the ATS is discarded. The r_{m} parameters are only used when mentioned explicitly. All other experiments are based on the predefined $\{m_{\text{min}}, \dots, m_{\text{max}}\}$ interval. For a fine-grained analysis of the schedulers' behaviour, we set $s_{\text{U}} = 0.01$, as this enables a good identification of breakdown anomalies. In a second round of simulations where we examine the m_i parameter and task periods in more detail, we use $s_{\text{U}} = 0.1$.

5.3.2 Aims

In our experimental evaluations, we aim to answer the following questions:

1. Our prior results [25] (subject to consolidation) indicate remarkably performance differences between the different schedulers, when using the ratio of task sets that are feasible as a performance metric. How do the different schedulers compare against each other, when an exact schedulability test is applied (Section 6.1.1)?
2. How pessimistic is Goossens' feasibility interval (eq. (3), [16])? Even for small task periods, m_i and k_i values, the interval can get quite large. How high are the savings in terms of simulated time introduced through GST? Also, we examine the practical relevance of the optimised schedulability test for MKU that can be derived from Theorem 3 and Corollary 4. The results are presented in Section 6.1.2.
3. How relevant are breakdown anomalies (Section 6.1.3)?
4. As Goossens [16] shows, the initialisation of the k-sequences of tasks can impact the feasibility of a task set. However, it is open how to find good initialisation values. We explore the possibility of cross-initialisation of k-sequences between different schedulers: If a task set with given initial k-sequence is feasible only under one of two schedulers, simulation of the successful scheduler necessarily runs into a cycle of k-sequences when applying GST. Is it possible to use one of the recurring k-sequences as initial value for execution with the hitherto failing scheduler (Section 6.1.4)?

5. Due to cancellations, processing time already spent by the cancelled jobs is lost. How much performance is lost by the different schedulers (Section 6.1.5)?
6. In general, our evaluations are based on arbitrary task sets with random parameters which may not always have practical counterparts. If we restrict task periods and/or m parameters to realistic ranges/values, does this have any influence on the above questions? Both aspects are examined in Section 6.2.

6 Results

In [25], we have presented initial results on the performance of the some of the schedulers listed in Table 2. These results are based on the simulation of random task sets for a fixed number of time steps. If during this simulation time no violation of an (m, k)-constraint is detected, the task set is classified as feasible. This approach can yield false positive results, as an infeasibility may also happen just after the given number of time steps. Nevertheless, these results gave a first impression of the performance of the schedulers: Best results were achieved with the DBP and MKU approaches, followed by MKP and MKP-S. Least performance was exhibited by GDPA. Due to a bug in the implementation of the simulator, approaches based on FPP scheduling (DBP, MKP, MKP-S) exhibited a lower performance in [25] than they actually have. Also, MKU showed better performance than DBP for moderate overloads. We will consolidate these results in the following by using the exact schedulability tests as appropriate for the different schedulers. The results presented in this section are based on two groups of simulations. In the first group (Section 6.1), arbitrary task sets are examined in order to answer the first five questions laid down in Section 5.3.2. The second group (Section 6.2) deals with the use of realistic task parameters (last question in Section 5.3.2). These are only examined from the performance point of view. A discussion of all results follows in Section 6.3.

6.1 Arbitrary Task Sets and Exact Schedulability Test

As already stated above, the simulation of a task set for a fixed number of time steps can yield false positive results. For a closer examination of the different approaches, we therefore apply the exact schedulability tests (see Table 2). We combine this with a search for the breakdown utilisation of an ATS (see Section 5.2). ATSs are executed beyond the breakdown point to account for breakdown anomalies (Section 4.3). CTSs derived from these ATSs are simulated as deemed necessary by the schedulability tests. The results presented in the following are base on the simulation of 500 ATSs that are executed with all schedulers in Table 2. Beyond performance ratings of the scheduling approaches, we also investigate the performance of GST, breakdown anomalies, and cross-initialisation of k-sequences between schedulers.

6.1.1 Scheduler Performance

The overall performance of all schedulers is shown in Figure 3. Due to breakdown anomalies, it may be possible that some ATSs are actually feasible again for higher utilisation, which is discussed later in Section 6.1.3. If we compare these numbers with the results of the rough estimation presented in [25], we note that the actual success ratio of all schedulers is lower, which is just to be expected due to false positives in the original results. Nevertheless, the ratios between the different schedulers remain nearly unchanged. The DBP and MKU approaches still exhibit outstanding performance. So, the rough estimation at least allows for qualitative comparison of the schedulers. Concerning the bug in the implementation of the FPP scheduler, the results now show that for

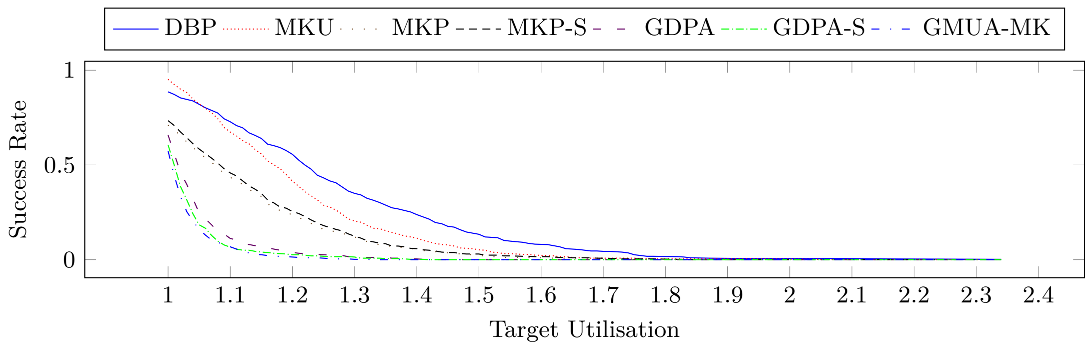


Figure 3 Ratio of ATSs that are schedulable up to a certain target utilisation U_{T}.

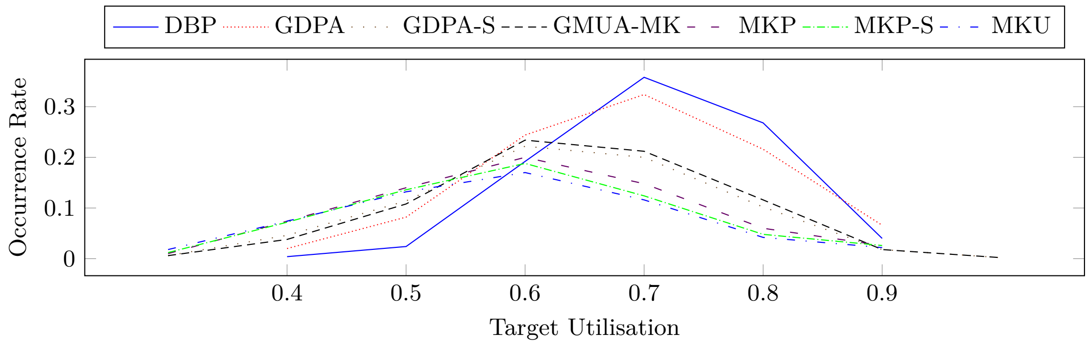


Figure 4 Breakdown (m, k)-utilisations U_{mk} (classified by rounding to nearest tenth).

most target utilisations DBP yields a better performance than MKU. Also, the MKP and MKP-S approaches exhibit a higher performance than estimated in [25].

Beyond providing a necessary schedulability condition, the (m, k)-utilisation U_{mk} does not help further to estimate the feasibility of a task set. Figure 4 shows the occurrence rate of the (m, k)-utilisations at the breakdown point of a task set, classified by rounding to the nearest tenth. The class $U_{\mathrm{mk}} = 0$ stands for ATSs that are not feasible at all. Obviously, a high (m, k)-utilisation does not per se prohibit feasibility, although this is achieved by only few ATSs. Most ATSs have a breakdown (m, k)-utilisation in the interval $[0.55, 0.85)$.

6.1.2 Performance of Schedulability Tests

The exact schedulability test for the approaches based on fixed (m, k)-patterns proposed in Section 4.1 yields a large feasibility interval. However, simulations must only be performed if the sufficient test [20] fails. In contrast, the feasibility interval for DBP [16] tends to exceed the MKP feasibility interval by far. In the following, we concentrate on the gains that are obtained through GST.

The DBP feasibility interval defines a very high bound for the number of hyperperiods that must be simulated successfully until feasibility of a task set can safely be assumed. Our experiments show that this bound is quite pessimistic and the optimisation incorporated in GST yields great value for the schedulability test. In all schedulers using GST, infeasibility is detected for most task sets ($> 99\,\%$) during the first hyperperiod, only few take longer. In the experiment at hand, the longest simulation to detect infeasibility takes three hyperperiods; in other simulations we observed durations of up to six hyperperiods. Feasibility is mostly found after the second hyperperiod,

Table 4 Number of ATSs that exhibit at least one breakdown anomaly.

	DBP	GDPA	GDPA-S	gMUA-MK	MKU
Absolute	20	11	9	11	40
Relative (%)	4.0	2.2	1.8	2.2	8.0

again with only few CTSs needing more time (up to 18 hyperperiods can be observed). For feasible CTSs, the first hyperperiod can be seen as a warm-up phase: At the start, the k-sequences have an arbitrary initialisation, in our case 1^k. These k-sequences are very unlikely to recur, as at least some jobs necessarily must be cancelled due to the overload. So, during the warm-up phase a good initialisation for the k-sequences is found, which leads into a recurring system state.

The gain from the optimised schedulability test for MKU that follows from Theorem 3 and Corollary 4 is only marginal. From 10270 CTSs in the experiment that are feasible under MKU, only 25 (0.2 %) would finish earlier.

6.1.3 Breakdown Anomalies

Extending the simulations described previously beyond the ATSs' breakdown points yields the following results. As already proven in Theorem 5, the schedulers based on fixed (m, k)-patterns (MKP, MKP-S) do not exhibit any anomalies. For the other schedulers, Table 4 gives the numbers of ATSs that exhibit at least one breakdown anomaly. With 8 % of the ATSs, MKU exhibits the largest number of anomalies. Thus, like all unsustainable algorithms, it should be treated with care.

6.1.4 Cross-Initialisation of k-Sequences

As noted by Goossens [16], the choice of the initial k-sequence can have significant impact on the feasibility of a task set. So far, no efficient algorithm is available that can derive a meaningful initialisation. We note that disjoint sets of CTS exist in our simulation results that are feasible only under one of any two approaches, but not under both. Feasibility in this case means that, after an initial warm-up phase which started with each task's k-sequence σ_i being initialised to 1^{k_i}, a system state $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ (see Section 3.2) periodically recurs. For a CTS that is feasible under two scheduling approaches, the corresponding system states may be different.

Our idea is to use a periodically recurring system state of a CTS that is only feasible under one of two schedulers for initialisation of the same CTS under the other scheduler. This approach might especially be interesting to improve the performance of approaches like GDPA. More formally, we assume a task set τ with $\sigma_i = 1^{k_i}, i = 1, 2, \ldots, n$ that is feasible under a scheduler $\mathit{SC}_\mathrm{F} \in \mathit{Schedulers} = \{\mathrm{DBP}, \mathrm{MKU}, \mathrm{GDPA}, \mathrm{GDPA\text{-}S}, \mathrm{gMUA\text{-}MK}\}$, but not under another scheduler $\mathit{SC}_\mathrm{I} \in \mathit{Schedulers}$. Thus, the execution of τ using SC_F finally runs into a cycle where a system state σ^F with $\sigma_i^\mathrm{F} \neq 1^{k_i}$ recurs periodically. This follows from the fact that the task set is overloaded and thus some jobs must be cancelled. Now we derive a task set τ' from τ by initialising each task τ_i''s k-sequence with the corresponding data from σ^F. τ' then is simulated under SC_I using GST to check whether the new initial k-sequence leads to a valid schedule. This approach is not applicable to the MKP and MKP-S approaches, as these disregard tasks' k-sequences.

In our simulations, we can identify significant numbers of candidate task sets only in the DBP and MKU approaches. Concerning the transfer of their final system state σ^F to other schedulers, only negligible successes can be achieved. For example, 263 distinct task sets are feasible in our simulations under DBP, but not under GDPA. Using their final system state for execution under

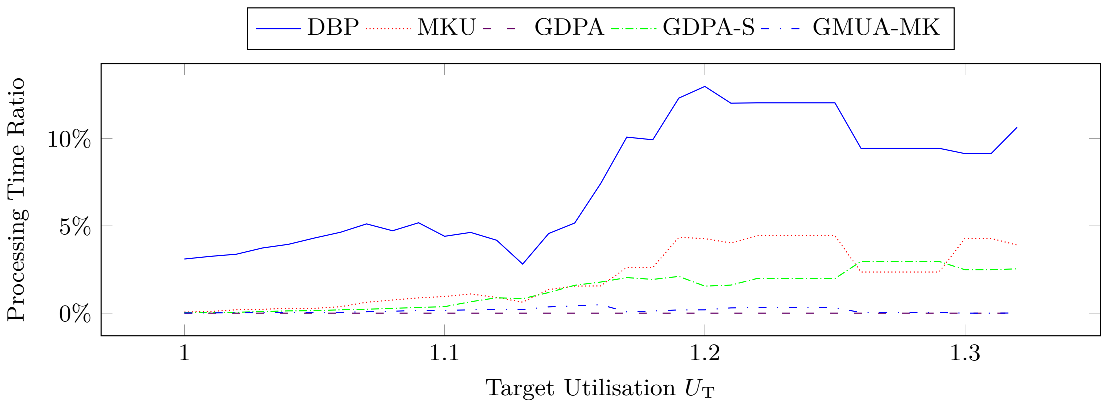


Figure 5 Mean lost processing time through EC (only feasible task sets; scaled by number of hyperperiods and hyperperiod length; only task sets where all schedulers successful).

GDPA leads to feasibility for only one task set. For a cross-initialisation from DBP to gMUA-MK, only 2 out of 274 candidates gain feasibility. Other transfers yield similar results. Thus, the cross-initialisation approach does not promise to lead to significant improvements concerning feasibility.

6.1.5 Cancellation of Running Jobs

Cancelling a job that has already started execution leads to the already consumed processing time being lost. Figure 5 shows the mean ratio of processing time that is lost due to cancellation of executing jobs. The schedulers based on fixed (m, k)-patterns (MKP, MKP-S) are omitted for two reasons: (1) If the sufficient schedulability test is successful, no simulation is performed, so no numbers are available for some task sets. (2) Only optional jobs (having lowest possible priority) are allowed to be cancelled; processing time that would be lost could be reclaimed by (possibly non-real-time) tasks that are running above the lowest possible priority, but still below the priorities of the (m, k)-firm real-time tasks.

Only CTSs that are feasible under all schedulers are included in the figures. The numbers are calculated in the following manner: For each CTS, the number of lost time steps is scaled by the task set's hyperperiod and the number of hyperperiods it is executed. From these numbers, the average is calculated for each target utilisation.

Figure 5 shows that approaches that have a rather low performance (in terms of their ability to find feasible schedules), tend to loose only a minor portion of processing time due to cancellation of already executing jobs. Most interestingly, in the GDPA no processing time is lost at all. This can be explained through the special technique in which GDPA calculates its schedule: Jobs with a high distance from dynamic failure are considered later for insertion into the EDF schedule than those that are near to a dynamic failure. If the insertion makes the EDF schedule infeasible, the job is removed again and deferred (but not yet cancelled!). Jobs with high distance to dynamic failure tend to be considered rather late for the schedule, and thus have a higher probability to lead to infeasibility, as the schedule might already be rather "full" through more critical jobs. Thus they are deferred without being executed, until they are cancelled due to missing their deadline.

The losses in the gMUA-MK approach stays well below 3 %. The costs incurred by GDPA-S are similar to those of MKU. Compared to the other approaches, DBP loses the highest amount of processing time.

The results allow us also to deduct that the higher flexibility of the DBP and MKU approaches (in terms of finding feasible schedules) is bought at the cost of a higher amount of lost processing

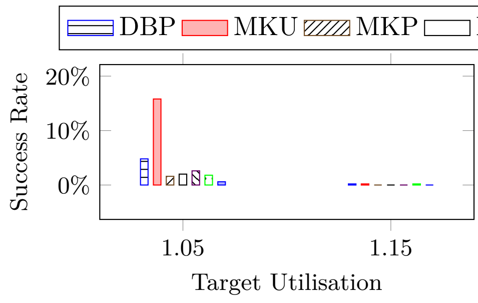


Figure 6 Performance with restricted m_i, $r_\text{m} = 0.8$.

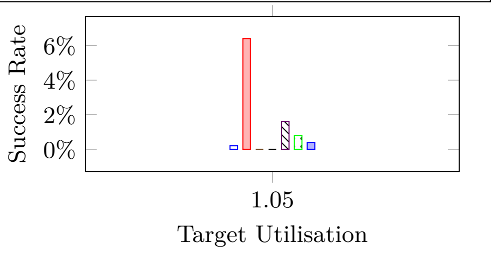


Figure 7 Performance with restricted m_i, $r_\text{m} = 0.9$.

time (here up to 13 %). Thereby, lower costs are incurred by MKU. This is due to the fact that MKU cancels jobs in a more anticipatory manner as soon as an overload pends somewhere in the schedule. Jobs in DBP are only cancelled when they can no longer meet their deadline.

6.2 Realistic Periods and m Parameters

The results presented so far are based on task sets with quite arbitrary parameters, concerning especially the task periods and m parameters. In reality, one would not find such a great variability: In real applications, task periods within a task set can be tuned to be harmonic or at least have many common divisors, and they span several orders of magnitude (see e.g. [28]). Also, it seems unrealistic to have tasks with a low ratio $\frac{m}{k}$ which would mean that most jobs could be skipped. In the following, we examine the behaviour of DBP and MKU under more realistic conditions by restricting m parameters, task periods, and both. For all results, the utilisation step is set to $s_\text{U} = 0.1$. Breakdown anomalies are ignored, i.e. an ATS is simulated only until the first infeasible CTS (these may differ for different schedulers!).

6.2.1 Restricted m Parameters

If generation of the m parameter of a task is restricted to an interval $[r_\text{m}k_i, k_i]$, we get some interesting results. In the following evaluations, we examine values $r_\text{m} \in \{0.1, 0.2, \ldots, 0.9\}$. Although, in our view, reasonable r_m values would rather be in the upper part of this set, we also need to look at low values, as we will see soon.

For each value of r_m, 500 ATSs are generated and simulated again with all schedulers. Like before, we use the breakdown utilisation of the ATSs and the number of feasible CTSs for each target utilisation to assess the performance of the schedulers. Breakdown anomalies occur in these simulations only rarely (≤ 2 % of the ATSs per r_m value), and are therefore ignored.

We show the success rates of the schedulers for $r_\text{m} = 0.8$ and $r_\text{m} = 0.9$ in Figures 6 and 7. Further diagrams for the other r_m values can be found in appendix B. For most schedulers, the performance ratio between any two stays similar to that found in the above simulations. As expected, the overall performance decreases with increasing r_m. This can most clearly be observed by a decrease of the maximum target utilisation for which feasible CTSs exist. Also, the number of feasible CTSs at $U_\text{T} = 1.05$ decreases rapidly with increasing r_m.

However, the DBP and MKU schedulers exhibit a more interesting behaviour, when examined in this detail compared to the simulations in 6.1. For very moderate overloads ($U_\text{T} = 1.05$), MKU achieves in most simulations, where $r_\text{m} \geq 0.3$, a better average performance than DBP. It seems that in these situations the advantages of EDF, which MKU is based on, over fixed-priority

$$\begin{pmatrix} 1 & 1 & 2 & 2 & 4 & 4 & 8 & 8 & 16 & 32 \\ 1 & 1 & 3 & 3 & 3 & 3 & 9 & 9 & 9 & 9 \\ 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 25 & 25 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 49 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 11 & 11 & 11 \end{pmatrix}$$

Figure 8 Matrix used for generation of realistic task periods.

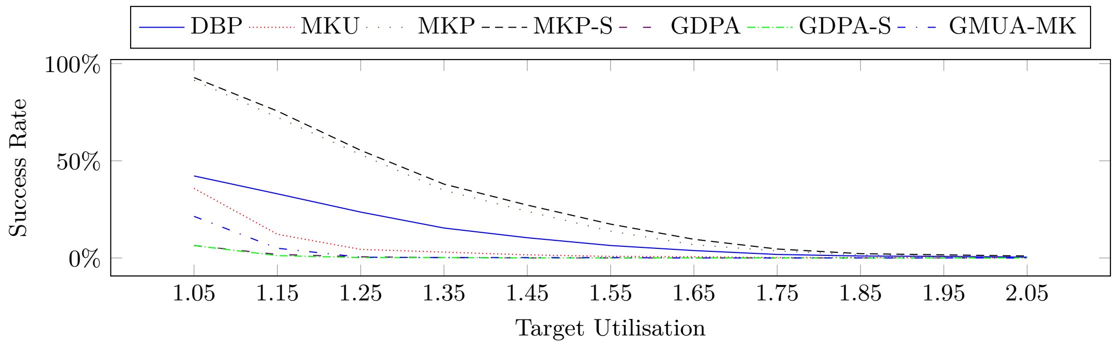


Figure 9 Ratio of task sets with realistic periods that are feasible up to a certain target utilisation U_T.

scheduling can still surface despite the overload. Applying the cross-initialisation technique (see Section 6.1.4) to transfer successful k-sequences from MKU to the more runtime-efficient DBP does not yield any successes for $r_\mathrm{m} \geq 0.6$, which in our view defines the most relevant range of m parameters.

6.2.2 Realistic Periods

In real applications, periods are often harmonic or have at least many common divisors. Also, they usually span several orders of magnitude. To imitate such circumstances, we use the period generator proposed by Goossens and Macq [17], which is aimed to generate task sets with limited hyperperiods (see Section 5.1). For our experiments, we use the matrix shown in Figure 8. With this matrix, we get periods in a range from 3 to 3,880,800. The maximum hyperperiod is also 3,880,800. All other task parameters are chosen as shown in Table 3.

Simulations are again performed for 500 ATSs. Figure 9 shows the ratio of ATSs that are feasible for a given target utilisation under the DBP and MKU approaches. Values for $U_\mathrm{T} > 2.05$ are omitted, as they are very small.

The larger range of periods seem to have a rather detrimental effect on most schedulers. Compared to the periods used in Section 6.1, their performance is more than halved. However, some exceptions exist: For a moderate overload at $U_\mathrm{T} = 1.05$, gMUA-MK actually improves, but deteriorates fast for larger U_T. The MKP and MKP-S schedulers actually achieve much better results. These can be attributed to the larger range of periods in the single task sets. As both approaches also use *rate monotonic* (RM) priorities [32] for their mandatory jobs, the critical instance at time $t = 0$ is greatly relieved: In the previous evaluations (Section 6.1), all jobs released at this time have similar deadlines and thus compete for processing time in the same interval. In contrast, with the extended period ranges, low-priority tasks (having long deadlines) can profit from multiple activations of tasks with shorter period within their period, as they can easily supersede the optional instances of the short-period tasks. Parts of the improvement may also stem from the fact that some task sets in this simulation have harmonic periods. For such

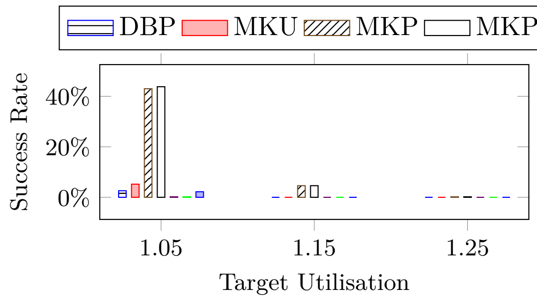


Figure 10 Performance with restricted m_i ($r_\mathrm{m} = 0.8$) and realistic periods.

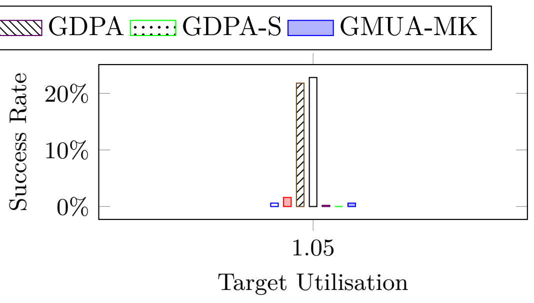


Figure 11 Performance with restricted m_i ($r_\mathrm{m} = 0.9$.) and realistic periods

task sets, it is known that the utilisation bound for schedulability under fixed priorities increases to $U \leq 1.0$, compared to the Liu/Layland bound of $U \approx 0.69$. Concerning the deterioration of DBP, we note that DBP in many cases makes decisions that are sub-optimal for such task sets. Depending on (m, k)-constraints, it happens that DBP prefers a task with higher period over one with lower period due to the priority assignment being solely based on distance from dynamic failure. In such task sets, this often leads to multiple consecutive low-period jobs (with actually high RM priority) being not executed at all and thus a violation of (m, k)-constraints.

6.2.3 Combination

Finally, we examine the combination of restricting tasks' m parameters and periods. Task parameters are generated as in Section 6.2.1 except for the periods, for which we employ Goossens' and Macq's approach [17] already used in Section 6.2.2. Again, 500 ATSs are generated and simulated. Performance numbers are again based on the breakdown utilisations, ignoring the rarely occurring breakdown anomalies.

Exemplarily, we show the success rates of the schedulers for $r_\mathrm{m} = 0.8$ and $r_\mathrm{m} = 0.9$ in Figures 10 and 11. Further diagrams for the other r_m values can be found in appendix B. They can be interpreted as a combination of the results of the previous two experiments. The performance of all approaches is clearly dominated by the larger variance of periods. The MKP/MKP-S approaches still achieve outstanding performance due to the higher variation of task periods within the task sets. The other approaches suffer from both the regular periods and the high r_m parameter.

6.3 Discussion

Our results make several points in regard to which scheduler should be used for which kind of set of (m, k)-firm real-time tasks. First, if task periods span several orders of magnitude, as is often the case for industrial applications, the schedulers based on fixed (m, k)-patterns can achieve better performance (see Section 6.2.2). They have the additional advantage that a simple schedulability test [20] is available. If task periods can be tuned to be harmonic, the test yields exact results. Second, for task periods that are in the same order of magnitude, better results are achieved using one of the DBP or MKU schedulers (see Section 6.1). Depending on how strongly the task set is constrained by (m, k)-parameters, either one of the two tends to yield better results. If constraints are very harsh, i.e. if the m_i are very near to the k_i, then the performance tends to be higher under MKU, and vice versa for DBP. However, it may still happen that, e.g. a strongly constrained task set is feasible under DBP, but not under MKU. So the final choice of a scheduler must be based on a accurate examination of the task set considering all schedulers available.

7 Conclusions

In this article, we have extended our prior work on HCUF-based scheduling of (m, k)-firm real-time tasks and examined several schedulers for (m, k)-firm real-time tasks. For existing schedulers for (m, k)-firm real-time tasks, we pointed out some new properties, namely an exact schedulability test for scheduling based on fixed (m, k)-patterns and the existence of breakdown anomalies in approaches like DBP. Concerning our HCUF-based heuristic MKU, we presented new formal results on the schedulability.

In an experimental evaluation, we examined the schedulers under several points of view. Therefore, extensive simulations of randomly generated task sets were performed using the different schedulers and different generation approaches. The simulations are based on the search for breakdown utilisation [29] of abstract task sets. Our results show that the HCUF-based heuristic MKU can achieve a similar performance as DBP [18], which has the best performance among all schedulers regarded if task periods within a task set are roughly in the same order of magnitude and (m, k)-constraints are very heterogeneous. Both approaches were able to find feasible schedules for up to 80 % of the generated task sets. They also show the advantage of the optimisation that GST [16] introduces for testing the exact schedulability condition for (m, k)-firm real-time task sets under DBP, as GST can reduce the simulation time significantly. The results show further that no clear relation exists between a task sets' (m, k)-utilisation and its feasibility. The occurrence of breakdown anomalies in our results indicate that care must be taken when using one of the DBP, MKU, GDPA, GDPA-S or gMUA-MK schedulers for an actual system: if the actual execution time of tasks is smaller than assumed during schedulability analysis, the task set may become infeasible under these schedulers. Depending on the used scheduler, about 2-8 % of the task sets were affected by this problem in our simulations. We also tackled the problem of finding good initialisations of tasks' k-sequences, as these can impact feasibility [16]. Using results produced by a feasible schedule as initialisation for a task set under another scheduler, where it is infeasible so far, could yield only minor improvements. The examination of processing time lost due to job cancellations gives some surprising results: Under this metric, the GDPA/GDPA-S [8] and gMUA-MK [39] schedulers achieved best performance (less than 2 % loss), while DBP lost up to 13 % of the processing time. The MKU approach lies somewhere between these numbers.

In further simulations, we restricted task set generation to realistic parameters. A lower bound for the m parameter prohibited the generation of tasks whose jobs are scarcely executed. As could be expected, having the m_i parameters of tasks near their k_i parameters resulted in a decrease of all schedulers' performances. However, we also observed that in such a scenario with very strong (m, k)-constraints, MKU can actually achieve better results than DBP. By using the period generator proposed by Goossens and Macq [17], periods spanning multiple orders of magnitude inside a task set were generated. In such task sets, the performance of MKU and DBP degraded significantly due to sub-optimal decisions. Concurrently, the schedulers based on fixed (m, k)-patterns (MKP [38] and MKP-S [37]) could achieve much higher performance (feasibility for up to ≈ 93 % of the generated CTSs).

We draw the following conclusions from our results: When the periods in a task set span several orders of magnitude, as is e.g. the case for automotive systems [28], then an approach using fixed (m, k)-patterns should be preferred. When task periods are roughly the same order of magnitude, then DBP or MKU can yield a better performance, but care must be taken for schedulability anomalies, as both algorithms are not sustainable.

In this article, we restrict ourselves to mapping (m, k)-firm constraints with HCUF, but the scheduling approach is not limited to these (m, k)-HCUFs. We expect that also other HCUFs can be used in the future, e.g. to communicate applications' current state and requirements such that the scheduler can adjust its decisions.

References

1 Saud Ahmed Aldarmi and Alan Burns. Dynamic value-density for scheduling real-time systems. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 270–277. IEEE Computer Society, 1999. doi:10.1109/EMRTS.1999.777474.

2 Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

3 Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis E. Shasha. On-line scheduling in the presence of overload. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 100–110. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185354.

4 Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001. doi:10.1109/12.919277.

5 Alan Burns and Sanjoy K. Baruah. Sustainability in real-time scheduling. *JCSE*, 2(1):74–97, 2008. URL: http://jcse.kiise.org/PublishedPaper/year_abstract.asp?idx=15.

6 Giorgio C. Buttazzo, Marco Spuri, and Fabrizio Sensini. Value vs. deadline scheduling in overload conditions. In *16th IEEE Real-Time Systems Symposium, Palazzo dei Congressi, Via Matteotti, 1, Pisa, Italy, December 4-7, 1995, Proceedings*, pages 90–99. IEEE Computer Society, 1995. doi:10.1109/REAL.1995.495199.

7 Ken Chen and Paul Mühlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10(3):293–312, 1996. doi:10.1007/BF00383389.

8 Hyeonjoong Cho, Yongwha Chung, and Daihee Park. Guaranteed dynamic priority assignment scheme for streams with (m, k)-firm deadlines. *ETRI Journal*, 32(3):500–502, June 2010. doi:10.4218/etrij.10.0109.0544.

9 Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. Utility accrual real-time scheduling for multiprocessor embedded systems. *J. Parallel Distrib. Comput.*, 70(2):101–110, 2010. doi:10.1016/j.jpdc.2009.10.003.

10 Hyeonjoong Cho, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. On multiprocessor utility accrual real-time scheduling with statistical timing assurances. In Edwin Hsing-Mean Sha, Sung-Kook Han, Cheng-Zhong Xu, Moon-hae Kim, Laurence Tianruo Yang, and Bin Xiao, editors, *Embedded and Ubiquitous Computing, International Conference, EUC 2006, Seoul, Korea, August 1-4, 2006, Proceedings*, volume 4096 of *Lecture Notes in Computer Science*, pages 274–286. Springer, 2006. doi:10.1007/11802167_29.

11 Raymond Keith Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, August 1990.

12 Robert I. Davis, Sasikumar Punnekkat, Neil C. Audsley, and Alan Burns. Flexible scheduling for adaptable real-time systems. In *1st IEEE Real-Time Technology and Applications Symposium, Chicago, Illinois, USA, May 15-17, 1995*, pages 230–239. IEEE Computer Society, 1995. doi:10.1109/RTTAS.1995.516220.

13 Wanfu Ding and Ruifeng Guo. Design and evaluation of sectional real-time scheduling algorithms based on system load. In *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008, Zhang Jia Jie, Hunan, China, November 18-21, 2008*, pages 14–18. IEEE Computer Society, 2008. doi:10.1109/ICYCS.2008.208.

14 Felicioni Flavia, Jia Ning, Françoise Simonot-Lion, and Yeqiong Song. Optimal on-line (m, k)-firm constraint assignment for real-time control tasks based on plant state information. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*, pages 908–915. IEEE, 2008. doi:10.1109/ETFA.2008.4638504.

15 Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In Julien Forget, editor, *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, pages 237–246. ACM, 2015. doi:10.1145/2834848.2834850.

16 Joël Goossens. (m, k)-firm constraints and DBP scheduling: Impact of the initial k-sequence and exact feasibility test. In *16th International Conference on Real-Time and Network Systems (RTNS'08)*, pages 61–66, October 2008.

17 Joël Goossens and Christophe Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the 9th International Conference on Real-Time Systems (RTS'01)*, pages 133–148, March 2001.

18 Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignement technique for streams with (m, k)-firm deadlines. *IEEE Trans. Computers*, 44(12):1443–1451, 1995. doi:10.1109/12.477249.

19 E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *6th Real-Time Systems Symposium (RTSS '85), December 3-6, 1985, San Diego, California, USA*, pages 112–122, December 1985.

20 Ning Jia, Ye-Qiong Song, and Françoise Simonot-Lion. Task Handler Based on (m,k)-firm Constraint Model for Managing a Set of Real-Time Controllers. In Nicolas Navet, Françoise Simonot-Lion, and Isabelle Puaut, editors, *15th International Conference on Real-Time and Network Systems - RTNS 2007*, pages 183–194, Nancy, France, 2007. URL: http://hal.inria.fr/inria-00189899.

21 Florian Kluge. `tms-sim` – timing models scheduling simulation framework – release 2014-12. Technical Report 2014-07, University of Augsburg, December 2014. doi:10.13140/2.1.1251.2321.

22 Florian Kluge. Notes on the generation of spin-values for fixed (m, k)-patterns. Technical Report 2016-01, University of Augsburg, January 2016.

23 Florian Kluge, Mike Gerdes, Florian Haas, and Theo Ungerer. A generic timing model for cyber-physical systems. In *Workshop Reconciling Performance and Predictability (RePP'14)*, Grenoble, France, April 2014. doi:10.13140/2.1.1820.4165.

24 Florian Kluge, Florian Haas, Mike Gerdes, and Theo Ungerer. History-cognisant time-utility-functions for scheduling overloaded real-time control systems. In *Proceedings of 7th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2013)*, Sophia Antipolis, France, October 2013.

25 Florian Kluge, Markus Neuerburg, and Theo Ungerer. Utility-based scheduling of (m, k) - firm real-time task sets. In Luís Miguel Pinho, Wolfgang Karl, Albert Cohen, and Uwe Brinkschulte, editors, *Architecture of Computing Systems - ARCS 2015 - 28th International Conference, Porto, Portugal, March 24-27, 2015, Proceedings*, volume 9017 of *Lecture Notes in Computer Science*, pages 201–211. Springer, 2015. doi:10.1007/978-3-319-16086-3_16.

26 Gilad Koren and Dennis E. Shasha. D$^{\text{over}}$; an optimal on-line scheduling algorithm for overloaded real-time systems. In *Proceedings of the Real-Time Systems Symposium - 1992, Phoenix, Arizona, USA, December 1992*, pages 290–299. IEEE Computer Society, 1992. doi:10.1109/REAL.1992.242650.

27 Gilad Koren and Dennis E. Shasha. An approach to handling overloaded systems that allow skips. In *16th IEEE Real-Time Systems Symposium, Palazzo dei Congressi, Via Matteotti, 1, Pisa, Italy, December 4-7, 1995, Proceedings*, pages 110–119. IEEE Computer Society, 1995. doi:10.1109/REAL.1995.495201.

28 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmark for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2015), July 7, 2015, Lund, Sweden*, July 2015.

29 John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*, pages 166–171. IEEE Computer Society, 1989. doi:10.1109/REAL.1989.63567.

30 Peng Li and Binoy Ravindran. Fast, best-effort real-time scheduling algorithms. *IEEE Trans. Computers*, 53(9):1159–1175, 2004. doi:10.1109/TC.2004.61.

31 Peng Li, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *IEEE Trans. Computers*, 55(4):454–469, 2006. doi:10.1109/TC.2006.47.

32 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

33 Carey Douglass Locke. *Best-effort decision-making for real-time scheduling.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1986. URL: http://douglocke.com/Downloads/BE.pdf.

34 Pedro Mejía-Alvarez, Rami G. Melhem, and Daniel Mossé. An incremental approach to scheduling during overloads in real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000), Orlando, Florida, USA, 27-30 November 2000*, pages 283–294. IEEE Computer Society, 2000. doi:10.1109/REAL.2000.896017.

35 Daniel Mossé, Martha E. Pollack, and Yagíl Ronén. Value-density algorithms to handle transient overloads in scheduling. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 278–286. IEEE Computer Society, 1999. doi:10.1109/EMRTS.1999.777475.

36 Enrico Poggi, Yeqiong Song, Anis Koubaa, and Zhi Wang. Matrix-dbp for (m, k)-firm real-time guarantee. In *Real-Time Systems Conference RTS'2003, Paris (France)*, pages 457–482, 2003.

37 Gang Quan and Xiaobo Sharon Hu. Enhanced fixed-priority scheduling with (m, k)-firm guarantee. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS 2000), Orlando, Florida, USA, 27-30 November 2000*, pages 79–88. IEEE Computer Society, 2000. doi:10.1109/REAL.2000.895998.

38 Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):549–559, 1999. doi:10.1109/71.774906.

39 J.-H. Rhu, J.-H. Sun, K. Kim, H. Cho, and J.K. Park. Utility accrual real-time scheduling for (m, k)-firm deadline-constrained streams on multiprocessors. *Electronics Letters*, 47(5):316–317, 2011. doi:10.1049/el.2010.7980.

40 Tiago Semprebom, Carlos Montez, and Francisco Vasques. (m, k)-firm pattern spinning to improve the GTS allocation of periodic messages in IEEE 802.15.4 networks. *EURASIP J. Wireless Comm. and Networking*, 2013:222, 2013. doi:10.1186/1687-1499-2013-222.

41 Sivakumar Swaminathan and Govindarasu Manimaran. A Reliability-Aware Value-Based Scheduler for Dynamic Multiprocessor Real-Time Systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 39–, Washington, DC, USA, 2002. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=645610.661233.

42 Terry Tidwell, Robert Glaubius, Christopher D. Gill, and William D. Smart. Optimizing expected time utility in cyber-physical systems schedulers. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*, pages 193–201. IEEE Computer Society, 2010. doi:10.1109/RTSS.2010.28.

43 Jinggang Wang and Binoy Ravindran. Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis. *IEEE Trans. Parallel Distrib.*

Syst., 15(2):119–133, 2004. doi:10.1109/TPDS.2004.1264796.

44 Richard West and Karsten Schwan. Dynamic window-constrained scheduling for multimedia applications. In *IEEE International Conference on Multimedia Computing and Systems, ICMCS 1999, Florence, Italy, June 7-11, 1999. Volume II*, pages 87–91. IEEE Computer Society, 1999. doi:10.1109/MMCS.1999.778145.

45 Haisang Wu, Binoy Ravindran, E. Douglas Jensen, and Umut Balli. Utility accrual scheduling under arbitrary time/utility functions and multiunit resource constraints. In *in IEEE Real-Time and Embedded Computing Systems and Applications*, pages 80–98, 2004.

A Acronyms

ATS abstract task set
CTS concrete task set
DASA dependent activities scheduling algorithm
DBP distance-based priority
EDF earliest deadline first
GDPA guaranteed dynamic priority assignment
GDPA-S simplified guaranteed dynamic priority assignment
gMUA-MK global multiprocessor utility accrual scheduling algorithm for (m, k)-firm deadline-constrained multimedia streams
GST Goossens' schedulability test
HCUF history-cognisant utility function
LBESA Locke's best-effort scheduling algorithm
MKP evenly distributed (m, k)-patterns
MKP-S evenly distributed (m, k)-patterns with spin values
MKU utility-based scheduling of (m, k)-firm real-time tasks
QoS Quality-of-Service
RM rate monotonic
TUF time-utility function
WCET worst-case execution time

B Additional Results

This appendix contains all performance results that are found during the simulation of task sets with restricted m parameter (see Section 6.2.1), and restricted m parameter and realistic periods (see Section 6.2.3).

B.1 Restricted m Parameter

The diagrams for $r_\mathrm{m} = 0.8$ and $r_\mathrm{m} = 0.9$ are omitted, as they are already shown in Figures 6 and 7. All other results from Section 6.2.1 are displayed in Figures 12 to 18.

B.2 Restricted m Parameter and Realistic Periods

Figures 19 to 21 show the results from the evaluations discussed in Section 6.2.3.

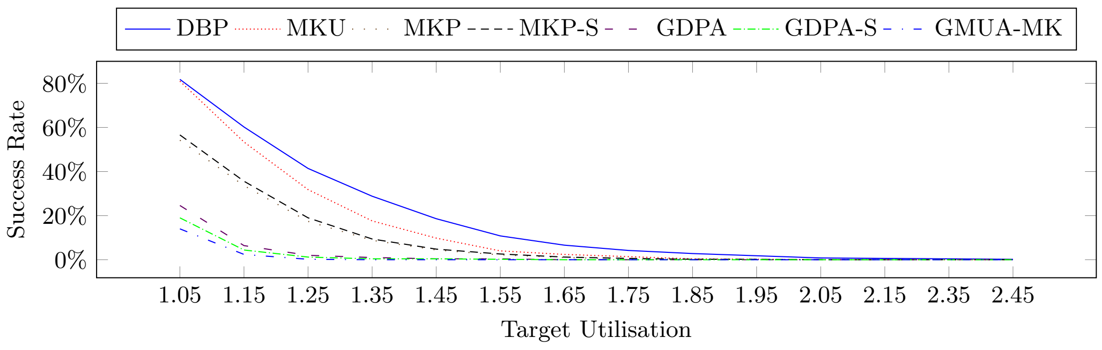


Figure 12 Performance with restricted m_i, $r_{\rm m} = 0.1$.

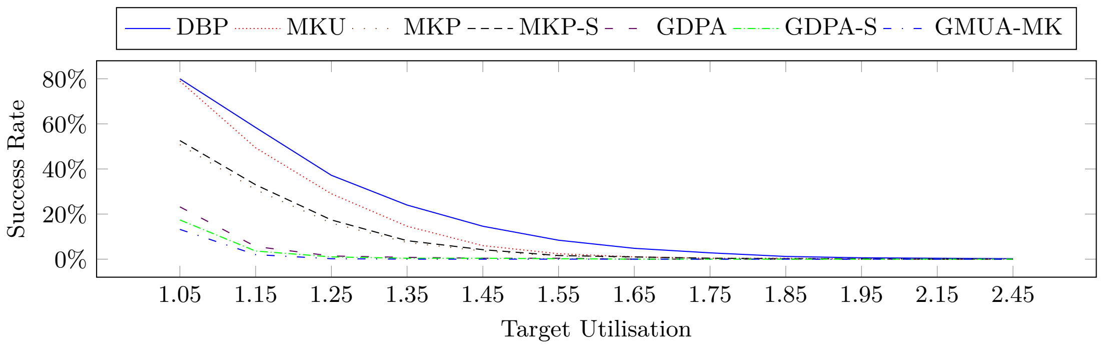


Figure 13 Performance with restricted m_i, $r_{\rm m} = 0.2$.

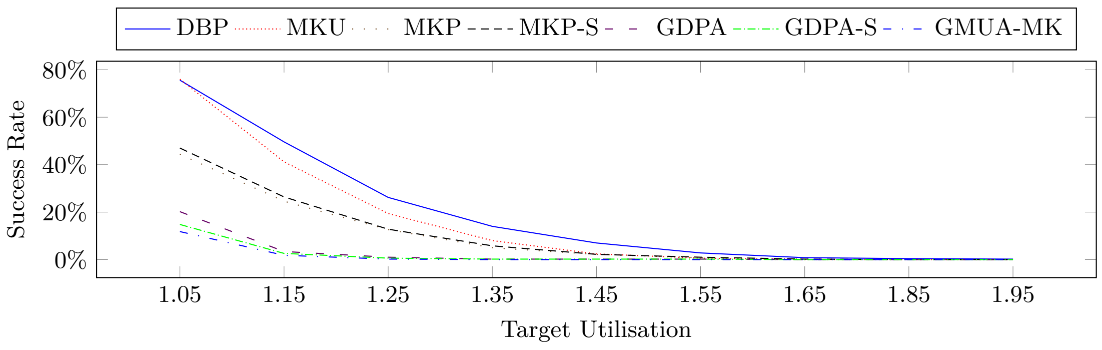


Figure 14 Performance with restricted m_i, $r_{\rm m} = 0.3$.

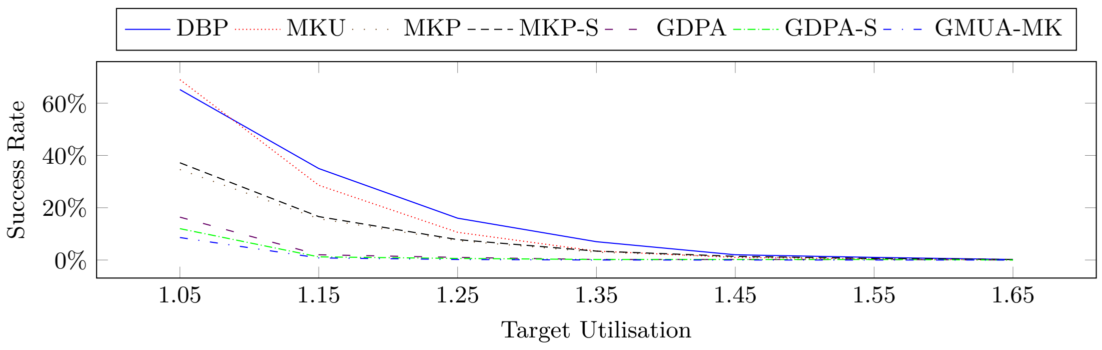


Figure 15 Performance with restricted m_i, $r_{\rm m} = 0.4$.

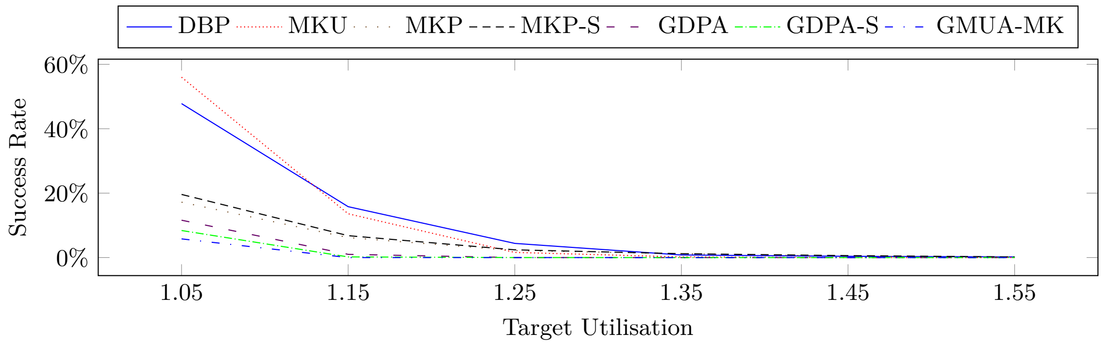


Figure 16 Success rates for $r_m = 0.5$.

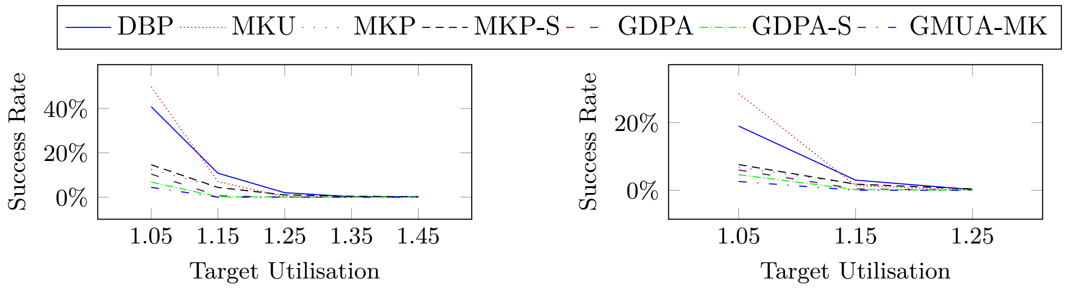


Figure 17 Performance with restricted m_i, $r_m = 0.6$.

Figure 18 Performance with restricted m_i, $r_m = 0.7$.

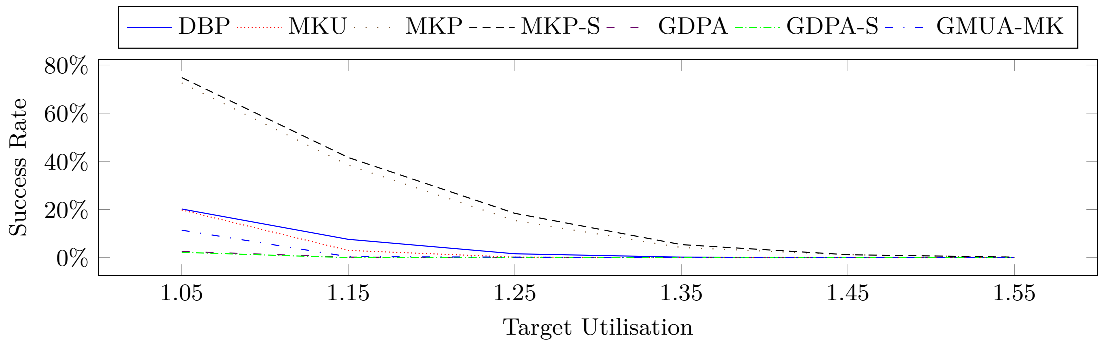


Figure 19 Performance with restricted m_i ($r_m = 0.5$.) and realistic periods.

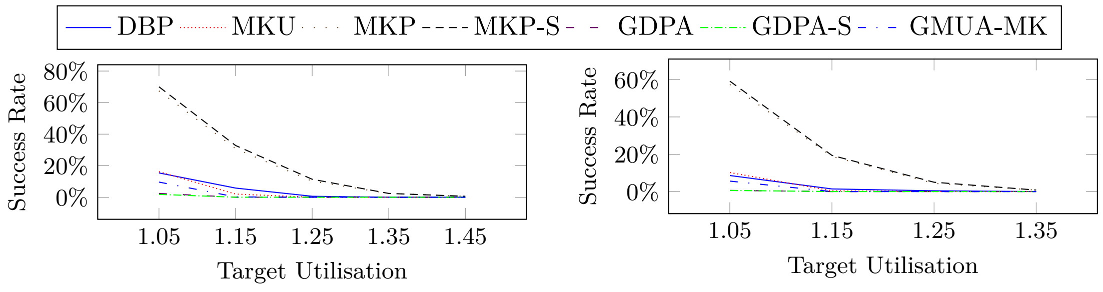


Figure 20 Performance with restricted m_i ($r_m = 0.6$) and realistic periods.

Figure 21 Performance with restricted m_i ($r_m = 0.7$.) and realistic periods.