UNIVERSITÄT AUGSBURG

Notes on the Generation of Spin-Values for Fixed (m, k)-Patterns

Florian Kluge

Report 2016-01 Januar 2016

INSTITUT FÜR INFORMATIK

D-86135 AUGSBURG

The use of fixed (m,k)-patterns for scheduling of (m,k)-firm real-time tasks has the advantage of a higher predictability compared to other approaches. However, feasibility of task sets can be influenced negatively through high interferences between jobs classified as mandatory under the (m,k)-patterns. Rotating the (m,k)-patterns of single tasks by a spin value can improve feasibility of task sets. Due to minor errors and missing information in the original presentation of this approach, a reproduction of these results is difficult. In this report we provide the necessary corrections. Our evaluation shows that, after applying these corrections, the original results can be reproduced.

Contents

1 Background

Certain types of applications do not demand the strict guarantees provided by hard real-time schedulers. They can tolerate sporadic deadline misses or job cancellations, but still require that such losses do not accumulate. Such requirements are, among others, formalised in the model of (m,k)-firm real-time tasks [2]. An (m,k)-firm real-time task is a tuple $\tau_i = (C_i, T_i, m_i, k_i)$ with execution time C_i, period T_i and (m,k)-constraint (m_i, k_i). The (m,k)-constraint specifies that at least m_i out of any k_i consecutive jobs must be executed successfully. Jobs $\tau_{i,j}$ with $j = 0, 1, \ldots$ are released at times jTi and must be finished until their deadline at $(j+1)T_i$. An (m,k)-firm real-time task incurs a dynamic failure, if less than m out of k consecutive jobs keep their deadline.

A variety of schedulers for (m,k)-firm real-time tasks has been proposed, both for preemptive and non-preemptive scheduling. In this report, we address the preemptive scheduling based on fixed (m,k)-patterns. Fixed (m,k)-patterns are proposed by Ramanathan [9] for (m,k)-firm scheduling of control applications. Jobs in this model are scheduled by fixed priority preemptive (FPP) scheduler. Initially, all tasks are assigned priorities by, e.g. using the rate-monotonic algorithm [5]. When a new job $\tau_{i,j}$ is released, the following condition is checked:

$$j = \left\lfloor \left\lceil \frac{jm_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor \tag{1.1}$$

If eq. (1.1) is fulfiled, $\tau_{i,j}$ is classified as *mandatory* and keeps τ_i's priority. Else, it is classified as *optional* and is assigned the lowest possible priority in the system. Insofar, eq. (1.1) defines a *fixed* (m,k)*-pattern* $\pi_i = \pi_{i,0}, \pi_{i,1}, ...$ with $\pi_{i,j} \in \{0,1\}, j \in \{0,1,\ldots,k_i - 1\}$ for the jobs $\tau_{i,j}$ of a task τ_i, where $\pi_{i,j} = 1$ stands for mandatory and $\pi_{i,j} = 0$ for an optional job. π_i is periodic with period k_i. A set of (m,k)-firm real-time tasks is feasible, if all mandatory instances of all tasks are executed successfully. In the following, this approach is termed *MKP*. Schedulability analysis of MKP is based on the fact that $t = 0$ is a critical instant: At this time, each task τ_i in a set of (m,k)-firm real-time tasks releases a mandatory job $\tau_{i,0}$. Using the methods of response-time analysis for FPP scheduling [1], a sufficient schedulability test can be derived [3].

The critical instant at $t = 0$ also introduces a high pessimism into schedulability analysis. Quan and Hu [7] propose to relieve the critical instant by introducing spin or rotation values for the (m,k)-patterns. For each task τ_i in a set of (m,k)-firm real-time tasks, an additional *spin parameter* s_i is defined. s_i is used in the classification of mandatory resp. optional jobs by adjusting equation 1.1. A job $\tau_{i,j}$ then is classified as mandatory, if

$$j + s_i = \left\lfloor \left\lceil \frac{(j+s_i)m_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor \tag{1.2}$$

Effectively, eq. (1.2) rotates a (m,k)-pattern π_i by s_i places to the left. In the same paper, Quan and Hu also show that the problem of finding optimal (m,k)-patterns such that a specific task

set is feasible, is NP-hard. They 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 report, we focus on the heuristic algorithm. It takes not only the interference at $t = 0$ into account, but actually any interferences that occur in a schedule. Quan and Hu use a slightly extended task model, where a task $\tau_i = \{O_i, T_i, D_i, C_i, m_i, k_i\}$ additionally has an initial activation offset O_i and a explicit deadline D_i that may be different from T_i. Involving the offset O_i in the calculations of the s_i makes the approach more general, as it can be applied to larger range of tasks. The explicit deadline D_i is not used the relevant calculations and thus has no effect on the results.

In the following, we use the term MKP-S to refer to the MKP algorithm with applied spin values derived from the heuristic algorithm. For MKP-S, Quan and Hu report improvements of up to 87.5 % compared to MKP in terms of feasibility of random task sets. In our work, we have reimplemented MKP-S. Initially, due to errors and missing information in [7], we were unable to reproduce the results presented there. The more comprehensive report by the same authors [8] also cannot shed light on these points.

The aim of this report is to complete the information that is missing in [7] such that the results presented there can be reproduced. We proceed as follows: In chapter 2 we provide the missing information. Based on this, we have performed evaluations to compare MKP and MKP-S. The results are presented in chapter 3. In chapter 4, we conclude this report.

2 Corrections

In the following, we present the correction that we apply to the original presentation of the MKP-S heuristic. The whole algorithm is discussed only superficially, we will delve only into those details where we see need for correction. Please refer to [7] for a precise description of MKP-S.

As already introduced in chapter 1, we use the numbers $0, 1, \ldots$ for indexing as it simplifies the presentation in places. Keep this in mind when comparing our descriptions with those in [7], where numbering of jobs starts with 1.

Some equations given in [7] to calculate execution interferences between tasks contain some minor errors that are hard to detect. In the following section 2.1, we go through those equations. Corrections are marked in **bold** font. The algorithms for the calculation of the total/maximum execution interference and spin values are missing important information resp. contain minor errors. Completed/corrected versions are presented in sections 2.2 and 2.3.

2.1 Execution Interference of a Single Job

The calculation of spin values is based on the execution interference between tasks. Therefore, the interference that single jobs $\tau_{i,j}$ experience from another task τ_h is regarded, where τ_h has higher priority than τ_i. This situation is depicted in figure 2.1. The release times of jobs are denoted $r_{h,x}$ for task τ_h, resp. $r_{i,x}$ for task τ_i.

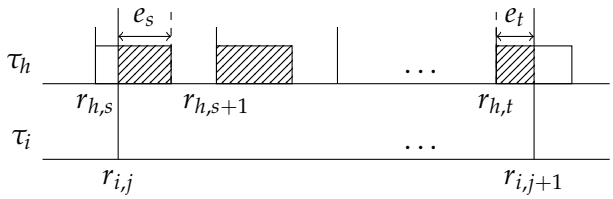

Figure 2.1: Execution interference of τ_h experienced by $\tau_{i,j}$ [7]

The total execution interference $F_{i,j}^h$ (shaded areas in fig. 2.1) that job $\tau_{i,j}$ experiences from task τ_h is calculated as

$$F_{i,j}^h = e_s + l_{i,j}^h \mathbf{C_h} + e_t \tag{2.1}$$

where $l_{i,j}^h$ is the number of mandatory instances of τ_h between release time and deadline of $\tau_{i,j}$. Start and tail interferences are calculated according to the following equations:

$$e_s = \pi_{h,s} \cdot \mathbf{max}\{C_h + r_{h,s} - r_{i,j}, 0\} \tag{2.2}$$

$$e_t = \pi_{h,t} \cdot \min\{C_h, r_{i,j+1} - r_{h,t}\} \tag{2.3}$$

2.2 Execution Interference of a Task

The execution interference $F_{i,j}^h$ (eq. (2.1)) experienced by single jobs $\tau_{i,j}$ is used to calculate maximum execution interference $\mathcal{F}_i^h$ that τ_i experiences from τ_h. Quan and Hu present an algorithm (algorithm 1 in [7]) that performs this calculation. In the original presentation, a term x is introduced but never used neither explained. However, it is crucial to correct operation of the algorithm.

Algorithm 2.1: Calculation of execution interference $\mathcal{F}_i^h$ experienced by τ_i from τ_h

Input : $\tau_i = \{O_i, T_i, D_i, C_i, m_i, k_i\}, \tau_h = \{O_h, T_h, D_h, C_h, m_h, k_h\}, \pi_i, \pi_h, h < i$
Output: $\mathcal{F}_i^h$

1 $\mathcal{F}_i^h = 0$;
2 $g = \gcd(k_i T_i, k_h T_h)$;
3 **for** $j = 0 \dots k_i - 1$ **do**
4 **if** $\pi_{i,j} = 1$ **then**
5 $x = (O_i + (jT_i - O_h) \bmod g$;
6 **while** $x < k_h T_h$ **do**
7 $F_{i,j}^h = e_s + l_{i,j}^h C_h + e_t$; // Calculate $F_{i,j}^h$ according to eq. (2.1)
8 **if** $\mathcal{F}_i^h < F_{i,j}^h$ **then**
9 $\mathcal{F}_i^h = F_{i,j}^h$;
10 $x = x + g$;

The original algorithm is reproduced in algorithm 2.1, assuming that job numbering starts with 0. The algorithm considers all mandatory instances of τ_i. The (m,k)-pattern π_i has length i and then is repeated, i.e. if instance $\tau_{i,j}, j \in \{0, 1, \dots, k_i - 1\}$ is mandatory (resp. optional), than all instances $\tau_{i,nl}, n \in \mathbb{N}$ are also mandatory (resp. optional). So, at least for the purpose of algorithm 2.1, a job $\tau_{i,j}$ can actually represent a group of jobs. Thus, it suffices to consider only k_i consecutive instances (line 3). From those, only the mandatory instances need to be regarded (line 4), as failure of optional instances has no effect on feasibility. The loop in lines 6 to 10 regards all possible (groups of) instances of τ_h that could interfere with $\tau_{i,j}$. Thereby, the variable x represents the activation offset between a reference instance of τ_h and $\tau_{i,j}$. For initialisation and evolution of x, please refer to [7] (especially lemmas 1 and 2). Without loss of generality, we assume that the reference instance is $\tau_{h,0}$. As x increases up to $k_h T_h$, all possible instances of τ_h are regarded. The meaning of x in this context is illustrated in figure 2.2.

Based on this information, we can now identify the instances of τ_h that are relevant for the calculation of the execution interference $F_{i,j}^h$ in line 7. Keep in mind that the following calculations actually regard groups of jobs $\tau_{i,nj}, n \in \mathbb{N}$ (resp. $\tau_{h,nl}$) represented by jobs $\tau_{i,j}$ (resp. $\tau_{h,l}$). From these groups, some concrete jobs can be found that actually exhibit the interferences shown in figure 2.2. The start interference e_s originates from the that instance $\tau_{h,s}$ that is activated just before $r_{i,j}$, but has its deadline after $r_{i,j}$. Then, s can be calculated as:

$$s = \left\lfloor \frac{x}{T_h} \right\rfloor \tag{2.4}$$

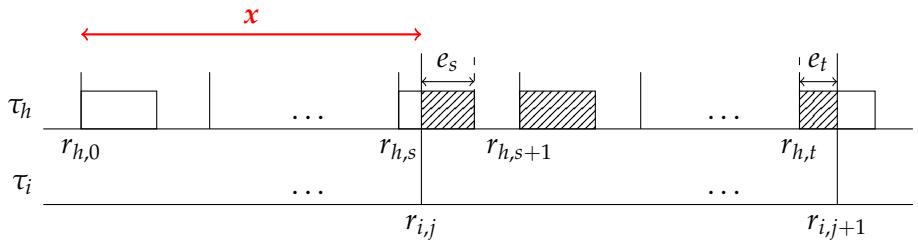

Figure 2.2: Finding instances of τ_h interfering with $\tau_{i,j}$

If $r_{h,s} = r_{i,j}$, then $e_s = 0$, else it is calculated according to eq. (2.2). Similarly, $\tau_{h,t}$ is the last job of τ_h that is released before $r_{i,j+1} = r_{i,j} + T_i$, and thus

$$t = \left\lfloor \frac{x + T_i}{T_h} \right\rfloor \tag{2.5}$$

is used in the calculation of e_t according to eq. (2.3). The special case $r_{h,t} = r_{i,j+1}$ needs not be regarded separately. $l^h_{i,j}$ is obtained by examining the (m,k)-pattern π_h for all instances $\tau_{h,l}$ with $l \in \{s+1, \ldots, t-1\}$.

2.3 Calculation of Spin Values

The calculation of actual spin values s_i (algorithm 2 in [7]) is partially based on lemma 3 in [7]. However, the lemma assumes that the π_i are shifted to the right, but eq. (1.2) actually introduces a left-shift of the π_i. Also, it ignores initial offsets that hitherto were used in the paper. Therefore, it should be adjusted as follows:

Lemma 1. *For τ_i with the (m,k)-patterns defined through eq.* (1.2)*, the number of mandatory jobs of τ_i is the largest in $[O_i + (k_i - s_i)T_i, O_i + (k_i - s_i)T_i + t]$ compared with those within any other interval of the same length t.*

If O_i (with $O_i > 0$) were ignored, one could eventually construct a $t = T_i + \epsilon, \epsilon > 0$ for which the claim would not hold. Regardless of the changes, the proof for this lemma can still be obtained by applying lemma 4 in [9].

Due to the different shift directions, also algorithm 2 in [7] for the calculations of the s_i must be adjusted. In line 17 of this algorithm, an offset O'_j is calculated. This calculation should be changed to:

$$O'_j = O_j + (k_j - s_j)T_j \tag{2.6}$$

3 Review of Results

To review the results concerning MKP-S presented in [7], we perform new simulations. Therefore, we have integrated the MKP and MKP-S schedulers into the scheduling simulator tms-sim [4]. For task set generation, we use the same parameters as [7]: Each task set consist of 5 tasks. Periods are randomly selected from $[10, 50]$, deadlines are assumed to be equal to the periods. k_i is chosen randomly from $[2, 10]$, and m_i from $[1, k_i]$. Execution times C_i are generated based on execution time weights e_i that are randomly chosen from $[1, 500]$. Based on these, executions are calculated such that the task sets have utilisations $U_T \pm 0.05$ with $U_T \in \{1.05, 1.15, \ldots, 1.75\}$. In total, we generate abstract 1,000 task sets. From these, concrete task sets are derived for each U_T. If a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ does not fulfil the necessary schedulability condition $\sum_{i=1}^{n} \frac{m_i C_i}{k_i T_i} \leq 1$ [6], it is discarded. Further speed-ups are achieved by checking for schedulability using the sufficient condition devised by Jia et al. [3]. Our simulations are less extensive than those presented in the original paper [7], but for most U_T they confirm the original results at least qualitatively.

The results of our simulations are shown in table 3.1. The last column shows the original results from [7] for the range defined by the U_T values ± 0.05. For $U_T \leq 1.35$ we can confirm the results at least qualitatively, although we achieve lower the improvements. For higher U_T, the number of feasible task sets in our simulations is too low for statistical significance, but also indicates qualitative improvements.

Table 3.1: Experimental results

U_T	MKP	MKP-S	Improvement	Improvement [7]
1.05	487	525	8.0 %	15.96 %
1.15	266	287	7.9 %	
1.25	143	161	12.6 %	17.32 %
1.35	66	75	13.6 %	
1.45	25	34	36.0 %	32.34 %
1.55	8	12	50.0 %	
1.65	4	6	50.0 %	87.50 %
1.75	1	1	0	

4 Conclusion

The introduction of spin parameters for fixed (m,k)-patterns can improve the schedulability for sets of (m,k)-firm real-time tasks. Due to minor errors and missing information in the original works [8, 7], the results presented there were not reproducible. In this report we provide the necessary corrections. We apply these to an implementation of the MKP-S scheduler. Our simulation results show that the improvements reported in [7] can be reproduced at least qualitatively.

5 Bibliography

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sept. 1993.

[2] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, 1995.

[3] N. Jia, Y.-Q. Song, and F. Simonot-Lion. Task Handler Based on (m,k)-firm Constraint Model for Managing a Set of Real-Time Controllers. In N. Navet, F. Simonot-Lion, and I. Puaut, editors, *15th International Conference on Real-Time and Network Systems - RTNS 2007*, pages 183–194, Nancy, France, 2007.

[4] F. Kluge. `tms-sim` – timing models scheduling simulation framework – release 2014-12. Technical Report 2014-07, University of Augsburg, Dec. 2014.

[5] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[6] E. Poggi, Y. Song, A. Koubaa, and Z. Wang. Matrix-dbp for (m, k)-firm real-time guarantee. In *Real-Time Systems Conference RTS'2003, pp457-482, Paris (France)*, 2003.

[7] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *The 21st IEEE Real-Time Systems Symposium, 2000. Proceedings.*, pages 79–88, 2000.

[8] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. Technical Report 00-09, Department of Computer Science & Engineering, University of Notre Dame, 2000.

[9] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, 1999.