Some Insights on Fixed-Priority Preemptive Non-Partitioned Multiprocessor Scheduling

Björn Andersson and Jan Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
{*ba,janjo*}*@ce.chalmers.se*

1 Introduction

Fixed-priority preemptive scheduling of independent periodic tasks on a homogeneous multiprocessor is solved using one of two different methods based on how tasks are assigned to the processors at run-time. In the *partitioned* method, all instances of a task are executed on the same processor, where the processor used for each task is determined before run-time by a partitioning algorithm. In the *non-partitioned* method, a task is allowed to execute on any processor, even when resuming after having been preempted. Two fundamental properties have been shown for the addressed problem [1]. First, the problem of deciding whether a task set is schedulable is NP-hard for both methods. Second, there are task sets which are schedulable with an optimal priority assignment with the non-partitioned method, but are unschedulable with an optimal partitioning algorithm and conversely.

Among the two methods, the non-partitioned method has received considerably less attention, mainly because it is believed to suffer from several scheduling-related shortcomings. The most well-known of these is *Dhall's effect*, a scheduling dilemma wherein some task sets may be unschedulable on multiple processors even though they have a low utilization [2]. Another shortcoming is that existing necessary and sufficient schedulability tests all have exponential time complexity [3], and existing sufficient tests have polynomial complexity but are pessimistic. It has also been shown that the RM (rate-monotonic) priority-assignment scheme is not optimal [1, 2], and no optimal priority-assignment schemes with polynomial time complexity have been found.

In this paper, we present an in-depth analysis of the non-partitioned method in terms of its scheduling-related properties. We (i) identify a set of anomalies for preemptive scheduling with migration, which are the first ever reported in the open research literature, (ii) identify several difficulties in conveying techniques from uniprocessor scheduling to the multiprocessor case, and (iii) conjecture that there may exist priority-assignment schemes that can contribute to circumventing Dhall's effect, something that has believed to be inherently impossible with the non-partitioned method.

2 Concepts and System model

We consider the problem of scheduling a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n independent, periodically-arriving real-time tasks on m identical processors. Each task $\tau_i \in \tau$ is described by the pair (T_i, C_i). A task arrives periodically with a period of T_i. Each time a task arrives, a new *instance* of the task is created. Each instance has a constant execution time of C_i. A *critical instant* for a task τ_i is an arrival time of an instance such that the response time is maximized. Each task has a deadline, which is the time of the next arrival of the task. Each task has a global, unique and fixed priority. The tasks in τ are numbered in the order of decreasing priority, that is, τ_1 has the highest priority. Of all tasks that have arrived, but not completed, the m highest-priority tasks are executed[1] in parallel on the m processors.

The *utilization* of a task is the ratio of the task's execution time to its period. The utilization U of a task set is then $U = \sum_i C_i/T_i$. To express the average utilization per processor for a task set executing on m processors, we use the *system utilization* $U_s = U/m$. A task is *schedulable* if all its instances complete no later than their deadlines. A task set is schedulable if all its tasks are schedulable. A task τ_i is *saturated* if it is schedulable, but any increase in C_i makes τ_i unschedulable. A task set is *fully utilized* if the task set is schedulable, but there is at least one task such that if it increases its execution time, then the task set is unschedulable.

We consider a system where tasks are independent, arrive periodically, require no other resources than the processors, and can always be preempted. The cost of preemption is assumed to be zero, even if a task is resumed on another processor than the task was preempted on (that is, the cost of migration is also assumed to be zero).

3 Task Interference and its Implications

For both uniprocessor and non-partitioned multiprocessor fixed-priority scheduling, the execution of a task is only delayed by its higher-priority tasks. The amount of this delay is denoted *interference*. The interference on a task τ_i is the intersection of execution of its higher-priority tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$. For uniprocessor scheduling this interference can be computed by knowing how many instances of higher-priority tasks execute during a time interval. For non-partitioned multiprocessor scheduling this interference only occurs for $\tau_{m+1}, \tau_{m+2}, \ldots$ and can be computed as the sum of all intervals when m higher-priority tasks execute in parallel on the m processors, thus delaying the execution of the remaining tasks.

[1] At each instant of time, the processor chosen for each of the m tasks is arbitrary. If less than m tasks should be executed simultaneously, some processors will be idle.

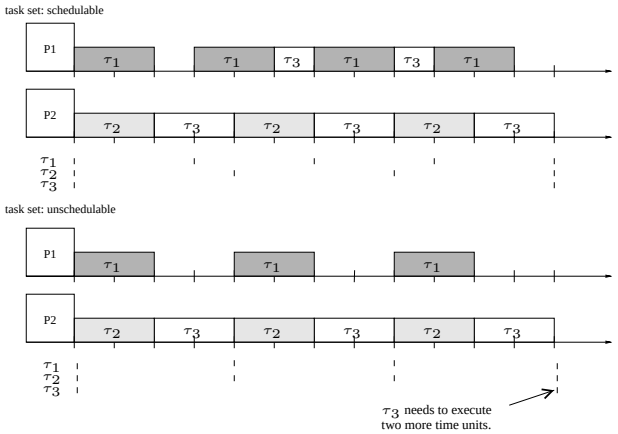

Figure 1: When τ_1 increases its period to 4, τ_3 becomes unschedulable. This is because τ_3 is saturated and its interference increases from 4 to 6.

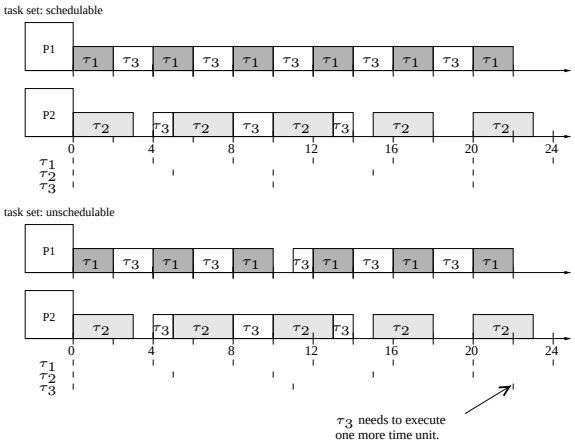

Figure 2: When τ_3 increases its period to 11, the second instance of τ_3 misses its deadline. This is because the interference increases from 3 to 5 and τ_3 is already saturated.

For uniprocessor scheduling, every increase in processor demand[2] (for example increasing C_j or decreasing T_j) of a higher-priority task τ_j causes the same interference or less interference on a lower-priority task. However, for multiprocessor scheduling, the interference is not only dependent on the processor demand of higher-priority tasks, but also on the time when they execute. This latter phenomenon gives rise to a series of counter-intuitive observations.

3.1 Scheduling anomalies

In real-time scheduling, it is often the case that the deadline miss ratio highly depends on the system load, that is, the requested processor utilization of tasks in the system. A commonly-used conclusion from this is that increasing the period will decrease the utilization, which in turn decreases the deadline miss ratio. Below, we present the first scheduling anomaly for preemptive real-time scheduling with migration (anomalies have previously only been shown for non-preemptive scheduling [4] and preemptive non-migrating scheduling [5]), that may invalidate such intuition.

Our first anomaly concerns a situation where a decrease in processor demand from higher-priority tasks can, because of the change in the time when the tasks execute, increase the interference on a lower-priority task.

Observation 1 (Preemptive anomaly, directed) *For fixed-priority preemptive non-partitioned multiprocessor scheduling, there exist schedulable task sets such that if the period of a task τ_j increases, a task τ_i with a lower priority will be unschedulable.*

Example 1 (See Figure 1) *Task τ_3 misses its deadline when task τ_1 increases its period. This can happen for the following schedulable task set (with priorities assigned according to RM):* $(T_1 = 3, C_1 = 2), (T_2 = 4, C_2 = 2), (T_3 = 12, C_3 = 8)$.

The second anomaly concerns a situation where a decrease in processor demand of a task negatively affects the task itself. When the period of the task changes, the arrival times of the task also change. As a consequence, the time interval within which we need to accumulate interference by higher-priority tasks may change. In some cases, the new interference may be larger than before.

Observation 2 (Preemptive anomaly, reflexive) *For fixed-priority preemptive non-partitioned multiprocessor scheduling, there exist schedulable task sets such that if the period of a task τ_i increases, the same task τ_i will be unschedulable.*

Example 2 (See Figure 2) *Consider the following schedulable task set (with priorities assigned according to RM):* $(T_1 = 4, C_1 = 2), (T_2 = 5, C_2 = 3), (T_3 = 10, C_3 = 7)$. *If we increase T_3, the resulting task set becomes unschedulable.*

Note that, in the examples above, the relative priorities of tasks do not change as a result of the increased periods. Therefore, the presented anomalies are true anomalies in the sense that they are directly triggered by changes in period.

One way of circumventing the anomalies is to use a schedulability test that only accepts task sets that cannot suffer from the anomalies. Theorem 1 provides such a schedulability test (proof in [6]).

Theorem 1 (Circumventing anomalies) *If for each task τ_i in a task set there exists a ${R_i}^{UB} \leq T_i$ such that:*

$$C_i + \frac{1}{m} \sum_{j \in hp(i)} \Big(\lfloor \frac{{R_i}^{UB}}{T_j} \rfloor C_j + \min({R_i}^{UB} - \lfloor \frac{{R_i}^{UB}}{T_j} \rfloor T_j, C_j) \Big) \leq {R_i}^{UB}$$

then the task will still be schedulable regardless of whether the task's own period T_i, or a higher-priority task's period T_j, increases.

An ${R_i}^{UB}$ can be obtained by inserting ${R_i}^{UB} = 0$ in the left-hand side of Equation 1, and use fix-point iteration, until Equation 1 is satisfied or ${R_i}^{UB} > T_i$.

[2] We define *processor demand* of a task in a time interval as the maximum amount of processing time required by the task's instances in the interval.

These two anomalies have two major implications. First, if a task arrives sporadically, less frequently than a certain bound, the task set may be unschedulable even if the task set would be schedulable if the task arrived periodically. Second, higher-level adaption techniques, such as feedback control scheduling, can no longer assume that a task set will continue to be schedulable when a period of a task increases. A possible remedy is to adjust the execution times of tasks rather than their periods, or use the schedulability test in Theorem 1.

3.2 Absence of transitivity

Another difficulty with the non-partitioned method is that some basic assumptions in uniprocessor scheduling no longer hold for the multiprocessor case. Below, we present two observations associated with such non-transitivity.

The first observation concerns when a critical instant of a task will occur. For the uniprocessor case, a critical instant occurs when a task arrives at the same time as its higher-priority tasks, which means that both the processor demand and the interference from higher-priority tasks are maximized. For the multiprocessor case, the fact that the processor demand from higher-priority tasks is maximized in an interval does not imply that the corresponding interference is maximized.

Observation 3 (Critical instant [7, 8]) *For fixed-priority preemptive non-partitioned multiprocessor scheduling, there exist task sets where a critical instant of one of the tasks does not occur when it arrives at the same time as its higher-priority tasks.*

The second observation concerns the complexity of finding an optimal priority assignment for the non-partitioned method. Deriving priorities optimally for the uniprocessor case can be made according to the "test for lowest priority viability" algorithm [9]. A fundamental assumption in that algorithm is that, although different priority orderings of higher-priority tasks give different schedules, the interference on a lower-priority task is not affected. For the multiprocessor case, different priority orderings of higher-priority tasks also give different schedules. However, the interference on a lower-priority task may also change and thereby affect schedulability. Consequently, even if we could use schedulability tests that are necessary and sufficient, it is no longer possible to find an optimal priority assignment by using the "test for lowest priority viability" approach.

Observation 4 (Dependence on order of higher priority) *For fixed-priority preemptive non-partitioned multiprocessor scheduling, there exist task sets (see Example 3) for which the response time of a task depends not only on T_i and C_i of its higher-priority tasks, but also on the relative priority ordering of the those tasks.*

Example 3 *Consider the following schedulable task set: $(T_1 = 3, C_1 = 1), (T_2 = 3, C_2 = 1), (T_3 = 3, C_3 = 2), (T_4 = 4, C_4 = 2)$. If we assign priorities to these tasks according to RM (and give τ_3 lower priority than both τ_1 and τ_2), the task set is schedulable. However, τ_4 will be unschedulable if we swap the priority ordering of τ_2 and τ_3.*

4 Circumventing Dhall's effect

While the partitioned method relies on well-known optimal uniprocessor priority-assignment schemes, it is not clear as to what priority-assignment scheme should be used for the non-partitioned method. It is known that RM does not work well for the non-partitioned method [2]. Assume that the task set $(T_1 = 1, C_1 = 2\epsilon), (T_2 = 1, C_2 = 2\epsilon), \ldots, (T_m = 1, C_m = 2\epsilon), (T_{m+1} = 1 + \epsilon, C_{m+1} = 1)$ should be scheduled using RM on m processors. In this case, τ_{m+1} will have the lowest priority and will only be scheduled after all other tasks have executed in parallel. The task set is unschedulable and as $\epsilon \to 0$, the utilization becomes $U = 1$ no matter how many processors are used, that is, the system utilization $U_s = U/m$ decreases towards zero as m increases.

We observe that if τ_{m+1} could somehow be assigned a higher priority, the given task set would be schedulable. To obtain such a priority assignment, assign task priorities according to the difference between period and execution time of each task. Then, τ_{m+1} would be assigned the highest priority, and the task set would still be schedulable even if the execution time of any single task would increase slightly.

Based on this observation, we propose a new priority assignment scheme, called *TkC*, where the priority of a task τ_i is assigned according to the weighted difference between its period and its execution time, that is, $T_i - k \cdot C_i$ (ties are broken arbitrarily), where k is a global *slack factor*. Note that TkC (when $k = 0$) can also represent RM.

4.1 The basic problem

For $k \leq 0$, we recognize Dhall's effect. Even for all $0 < k \leq 1$, something similar to Dhall's effect can occur. Assume that the task set, $(T_1 = 1, C_1 = \epsilon), (T_2 = 1, C_2 = \epsilon), \ldots, (T_m = 1, C_m = \epsilon), (T_{m+1} = \frac{1}{\epsilon^2} + \epsilon, C_{m+1} = \frac{1}{\epsilon^2}(1 - \frac{\epsilon}{2}))$, where $\frac{1}{\epsilon}$ is a positive integer, should be scheduled using TkC with $0 < k \leq 1$ on m processors. As $\epsilon \to 0$, the task set is unschedulable with a utilization $U = 1$.

Selecting a value of k which is slightly larger than 1, we still experience something similar to Dhall's effect. Assume that the task set, $(T_1 = 1, C_1 = \frac{k-1}{k} + \epsilon), (T_2 = 1, C_2 = \frac{k-1}{k} + \epsilon), \ldots, (T_m = 1, C_m = \frac{k-1}{k} + \epsilon), (T_{m+1} = \frac{1}{\epsilon^2} + \frac{k-1}{k} + \epsilon, C_{m+1} = \frac{1}{\epsilon^2}(1 - \frac{k-1}{k} - \frac{\epsilon}{2}))$, where $\frac{1}{\epsilon}$ is a positive integer, should be scheduled using TkC with $1 \leq k$ on m processors. As $\epsilon \to 0$, the task set is unschedulable with a utilization $U = 1$.

On the other hand, selecting a too large k is a bad idea since task priorities will then be selected such that the tasks with the longest execution time obtains the highest priority. Assume that the task set $(T_1 = 1, C_1 = \frac{1}{1+k} + \epsilon), (T_2 = 1, C_2 = \frac{1}{1+k} + \epsilon), \ldots, (T_m = 1, C_m = \frac{1}{1+k} + \epsilon), (T_{m+1} = \frac{1}{1+k} + \epsilon, C_{m+1} = \epsilon^2)$ should be scheduled using TkC with $k \to \infty$ on m processors. As $\epsilon \to 0$, this task set is unschedulable with a utilization $U = 1$ no matter how many processors are used.

4.2 A simple solution

For TkC to be useful we need to select a good value of k. We observe that the task sets presented that were unschedulable with a low system utilization all have in common that: (i)

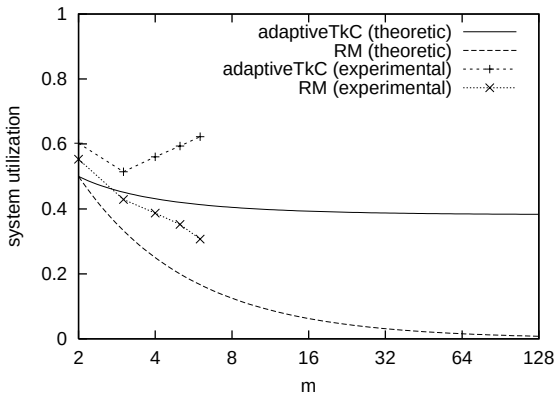

Figure 3: Theoretical and experimental results indicate that adaptiveTkC can circumvent Dhall's effect, while RM degrades towards zero as m increases.

the number of tasks is one greater than the number of processors, and (ii) all highest-priority tasks have the same period and execution time. Then, it is natural to try to optimize the value of k for these task sets (we call them constrained task sets). We conjecture that the constrained task sets represent a worst-case scenario, in the sense that they are the fully utilized task sets with the least system utilization.

To counter Dhall's effect and their similar effects ($0 \leq k \leq 1$ and small $k > 1$), we should select a large value of k. To counter the effect of the task set with a low utilization for large k, we should select a low value of k.

We select a value of k such that the conflicting task sets (small k versus large k) obtain the same system utilization, that is $\frac{k-1}{k} + \frac{1}{mk} = \frac{1}{1+k}$. Theorem 2 proposes such a scheme called *adaptiveTkC* (proof in [6]):

Theorem 2 (Selecting the best k) *Consider $m \geq 2$ processors and $n = m+1$ tasks. The m highest priority tasks have the same period and execution time. The tasks are sorted with respect to $T_i - k \cdot C_i$ and the task with the least $T_i - k \cdot C_i$ obtains the highest priority. The following k maximizes the least system utilization of fully utilized task sets:*

$$k = \frac{1}{2} \cdot \frac{m - 1 + \sqrt{5\,m^2 - 6\,m + 1}}{m}$$

and the corresponding least system utilization of fully utilized task sets is:

$$U_s = 2\,\frac{m}{3\,m - 1 + \sqrt{5\,m^2 - 6\,m + 1}}$$

We see that the theoretically-derived lower bound of the system utilization is never lower than $\lim_{m\to\infty} U_s > 0.38$, which shows that we have managed to circumvent Dhall's effect for the constrained task set.

In order to further strengthen our hypothesis that Dhall's effect can also be avoided for general task sets, we have performed an extensive experimental study [6] using adaptiveTkC. We simulated millions of randomly-generated task sets (not only constrained) with varying levels of system utilization (even ones below 0.38), and assumed system sizes up to 6 processors. The experimental results, and their theoretical counterparts, are shown in Figure 3. The plot "adaptiveTkC (theoretic)" shows the least system utilization of fully utilized constrained task sets, while the plot "RM (theoretic)" shows the system utilization of the task set: $n = m+1$, $T_1 = T_2 = \ldots = T_m = 1$, $C_1 = C_2 = \ldots = C_m = 2\epsilon$, $T_{m+1} = 1+\epsilon$, $C_{m+1} = 1$, when $\epsilon \to 0$. The plots "adaptiveTkC (experimental)" and "RM (experimental)" show the least system utilization of the unschedulable experimental (not only constrained) task sets. We draw the following conclusions. First, both the theoretical and experimental results corroborate the anticipated behavior of RM, namely that the system utilization should degrade towards zero as m increases. More importantly, though, the theoretical and experimental results indicate that adaptiveTkC seems to circumvent Dhall's effect (since the system utilization is never lower than 0.38).

One may question if it is worth to consider the non-partitioned method when the bound of system utilization provided by the adaptiveTkC can be as low as 38%. However, we should have in mind that the best bound of system utilization for the partitioned method is only 41% [10]. Recently, we have also shown [11] that adaptiveTkC offers better average-case performance than the best heuristics for partitioning with similar complexity. This fact gives an incentive to derive a bound of system utilization for general task sets.

References

[1] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[2] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, January/February 1978.

[3] J. Y.-T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4(2):209–219, 1989.

[4] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

[5] R. Ha and J. W.-S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, pages 162–171, Poznan, Poland, June 21–24, 1994.

[6] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. Technical Report 00-10, Dept. of Computer Engineering, Chalmers University of Technology, S–412 96 Göteborg, Sweden, April 2000.

[7] S. Lauzac, R. Melhem, and D. Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Proc. of the EuroMicro Workshop on Real Time Systems*, pages 188–195, Berlin, Germany, June 17–19, 1998.

[8] L. Lundberg. Multiprocessor scheduling of age contraint processes. In *Proc. of the IEEE Int'l Conference on Real-Time Computing Systems and Applications*, Hiroshima, Japan, October 27–29, 1998.

[9] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Dept. of Computer Science, University of York, York, England Y01 5DD, December 1991.

[10] D. Oh and T.P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, September 1998.

[11] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proc. of the IEEE Int'l Conference on Real-Time Computing Systems and Applications*, Cheju Island, Korea, December 12–14, 2000.