The Utilization Bound of Static-Priority Preemptive Partitioned Multiprocessor Scheduling is 50%

Björn Andersson, *Member, IEEE*

***Abstract*— This paper studies static-priority preemptive scheduling on a multiprocessor using partitioned scheduling. We propose a new scheduling algorithm and prove that if the proposed algorithm is used and if less than 50% of the capacity is requested then all deadlines are met. It is known that for every static-priority multiprocessor scheduling algorithm, there is a task set that misses a deadline although the requested capacity is arbitrary close to 50%.**

***Index Terms*— real-time scheduling, partitioning, bin-packing algorithms, static-priority scheduling, preemptive scheduling, multiprocessors.**

I. Introduction

CONSIDER the problem of scheduling a set of n sporadically arriving tasks using preemptive static-priority scheduling on m processors. A task τ_i can arrive many times. These arrival times cannot be controlled by the scheduling algorithm and the scheduling algorithm learns about the exact arrival time at the arrival time — no earlier. The arrival times from the same task τ_i are separated by T_i time units or more. Every time task τ_i arrives, it needs to execute for C_i time units no later than T_i time units after its arrival; otherwise it misses a deadline. C_i is called the execution time of τ_i and for historical reasons, T_i is called the period of τ_i. The utilization bound UB_A of a scheduling algorithm A is a number such that if $\frac{1}{m} \cdot \sum_{i=1}^{n} \frac{C_i}{T_i} \leq UB_A$ then all tasks meet their deadlines when scheduled by algorithm A. The design space of preemptive static-priority multiprocessor scheduling algorithms can be categorized as partitioned vs global scheduling. Global scheduling algorithms store tasks that have arrived but not finished its execution in one queue which is shared among all processors. At every moment the m highest priority tasks among the tasks that have arrived but not finished its execution are selected for execution on the m processors using preemption and migration if necessary. Partitioned scheduling algorithms partition the set of tasks such that all tasks in a partition are assigned to the same processor. Tasks are not allowed to migrate, hence the multiprocessor scheduling problem is transformed to many uniprocessor scheduling problems. Common for all static-priority multiprocessor scheduling algorithms is that they cannot have a utilization bound greater than 50% [1], [2]. Partitioned static-priority scheduling is well-studied [1], [3]–[12] but they all have a utilization bound of at most 41% [1], [13], leaving room for improvements.

In this paper, we propose a partitioning algorithm, called R-BOUND-MP-NFR, and prove that it has a utilization bound of 50%. We hence close the problem for partitioning.

We assume that (i) tasks do not use any other resource than a processor and (ii) a task can always be preempted and there is no overhead associated with a preemption.

The remainder of this paper is organized as follows. Section II gives a background on partitioned scheduling and in particular it shows the necessary ingredient to achieve a utilization bound greater than 41%. Section III studies partitioned scheduling when periods are restricted. This restriction is removed in Section IV. Section V quantifies how many more task sets that can be guaranteed by the new utilization bound. Finally, Section VI closes with a discussion, presents conclusions and future work.

This article is based on a conference paper presented at the 15th Euromicro Conference on Real-Time Systems.

Manuscript received March 29, 2006.

The author's affiliation for this work was Department of Computer Engineering, Chalmers University of Technology, SE–412 96 Göteborg, Sweden. *ba@ce.chalmers.se*. He is now with the IPP-Hurray! Research Group, Polytechnic Institute of Porto, Rua Dr. Antonio Bernardino de Almeida 431, 4200-072 Porto, Portugal. *bandersson@dei.isep.ipp.pt*.

II. Partitioned scheduling

The partitioned method divides tasks into partitions, each having its own dedicated processor. Unfortunately, the problem of deciding whether a schedulable partition exist is NP-complete [12]. Therefore many heuristics for partitioning have been proposed, a majority of which are versions of the bin-packing algorithm[1]. These bin-packing algorithms rely on a schedulability test in order to know whether a task can be assigned to a processor or not. This reduces our problem from partitioning a set of tasks to meet deadlines into the problem of partitioning a set of tasks such that, on every processor, the schedulability test can guarantee that all tasks on that processor meet their deadlines. As a schedulability test, a natural choice is to use the knowledge that: if $\sum_{i=1}^{n_p} u_i \leq n_p \cdot (2^{1/n_p} - 1)$ and rate-monotonic is used to schedule tasks on processor p then all deadlines are met [14]. (Let u_i=C_i/T_i and n_p denote the number of tasks assigned to processor p.) This schedulability test is often used, but as shown in Example 1 below, this bound is not tight enough to allow us to design a multiprocessor scheduling algorithm with a utilization bound of 50%.

Example 1: Consider $m+1$ tasks with $T_i = 1$ and $C_i = \sqrt{2} - 1 + \epsilon$ to be scheduled on m processors. For this system, there must be a processor p which is assigned two tasks. On that processor the utilization is $\sum_{i=1}^{n_p} C_i/T_i = 2 \cdot (\sqrt{2} - 1 + \epsilon)$

[1]The bin-packing algorithm works as follows: (1) sort the tasks according to some criterion; (2) select the first task and an arbitrary processor; (3) attempt to assign the selected task to the selected processor by applying a schedulability test for the processor; (4) if the schedulability test fails, select the next available processor; if it succeeds, select the next task; (5) goto step 3.

which is greater than $2 \cdot (\sqrt{2} - 1)$. Hence, there is no way to partition tasks so that all tasks can be guaranteed by this schedulability test to meet deadlines. We can do this reasoning for every m and every ϵ. By letting $\epsilon \to 0$ and $m \to \infty$ we can see that UB for algorithms that are based on this schedulability test cannot be greater than $\sqrt{2} - 1$, which is approximately 41%. □

Note that the task set in Example 1 could actually be guaranteed by a necessary and sufficient schedulability test to meet deadlines (provided that ϵ is not too large). It is known that if all tasks are harmonic[2] then the uniprocessor utilization bound is 100%[3], and then the task set in Example 1 could be assigned with two tasks on one processor. A uniprocessor schedulability test that could exploit this information could allow a multiprocessor scheduling algorithm to achieve a utilization bound of 50%. This is what we will do.

R-BOUND [10] is a uniprocessor schedulability test which exploits harmonicity. Let r_p denote the fraction between the maximum and the minimum period among the tasks assigned to processor p. If we restrict our attention to the case when $\forall p : 1 \leq r_p < 2$ (we will relax this restriction later), we have the following theorem (from [10]).

Theorem 1: Let $B(r_p, n_p) = n_p(r_p^{1/n_p} - 1) + 2/r_p - 1$. If $\sum_{i=1}^{n_p} C_i/T_i \leq B(r_p, n_p)$ and rate-monotonic is used to schedule tasks on processor p then all deadlines are met.

R-BOUND-MP is a previously known multiprocessor scheduling algorithm that exploits R-BOUND [10]. R-BOUND-MP combined R-BOUND with a first-fit bin-packing algorithm. However, its utilization bound is not known and it is difficult to analyze. For this reason, in order to show which utilization bound a partitioned scheduling algorithm can achieve, we will design two derivatives of R-BOUND-MP. First, we will consider an algorithm R-BOUND-MP-NFRNS (R-BOUND-MP with next-fit-ring noscaling) and prove its utilization bound when $1 \leq \frac{\max_{\tau_i \in \tau} T_i}{\min_{\tau_i \in \tau} T_i} < 2$. (τ denotes the set of all n tasks.) Then we will consider the algorithm R-BOUND-MP-NFR (R-BOUND-MP with next-fit-ring) and prove its utilization bound when periods are not restricted.

III. Restricted periods

In this section, we assume that $1 \leq \frac{\max_{\tau_i \in \tau} T_i}{\min_{\tau_i \in \tau} T_i} < 2$ holds. Clearly it means that no matter how we assign tasks to processors, it holds that $\forall p : 1 \leq r_p < 2$ and hence Theorem 1 can be used. We will use the algorithm R-BOUND-MP-NFRNS, illustrated in Algorithm 1. It works as follows: (i) sort tasks in ascending order of periods, that is, the task with the shortest period is considered first, (ii) use Theorem 1 as a schedulability test on each uniprocessor, (iii) assign tasks with the next-fit bin-packing algorithm and (iv) when a task cannot be assigned to processor m, try to assign it on processor 1, if this does not work then declare FAILURE. If the algorithm terminates and has partitioned the whole task set

[2]In a harmonic task set, the periods T_i and T_j of any two tasks τ_i and τ_j are related as follows: either T_i is an integer multiple of T_j, or T_j is an integer multiple of T_i.

[3]This is easy to see by dropping the ceiling in the equations/inequalities in exact schedulability tests [15], [16].

Algorithm 1 R-BOUND-NP-NFRNS, a task-to-processor assignment algorithm.

```
 Input: A task set $\tau$.
 Output: An assignment of a task to a processor.
 1: Sort tasks such that $T_1 \leq T_2 \leq \ldots \leq T_n$.
 2: i := 1
 3: j := 1
 4: while ($i \leq n$) loop
 5:   If no task has been assigned to processor j then
 6:     assign task $\tau_i$ to processor j.
 7:     i := i + 1
 8:   else
 9:     Let $\tau_{pj1}$ denote the first task that
10:       was assigned to processor $j$.
11:     Let $\alpha_j$ denote $T_i/T_{pj1}$.
12:     Let $n_j$ denote the number of tasks assigned to processor $j$.
13:     Let $UPROCESSOR_j$ denote the sum of the
14:       utilization of all tasks assigned to processor $j$.
15:     $UBOUND_j := (n_j + 1) \cdot (\alpha_j^{1/(n_j+1)} - 1) + 2/\alpha_j - 1$
16:     if $UPROCESSOR_j + C_i/T_i \leq UBOUND_j$ then
17:       assign task $\tau_i$ to processor j.
18:       i := i + 1
19:     else
20:       if j=m then
21:         Let $n_1\prime$ denote the number of tasks assigned to
22:           processor 1.
23:         Let $UPROCESSOR_1\prime$ denote the sum of the
24:           utilization of all tasks assigned to processor 1.
25:         Let $UBOUND_1\prime = (n_1\prime + 1) \cdot (2^{1/(n_1\prime+1)} - 1)$
26:         if $UPROCESSOR_1\prime + C_i/T_i \leq UBOUND_1\prime$ then
27:           assign task $\tau_i$ to processor 1.
28:           i := i + 1
29:         else
30:           declare failure.
31:         end if
32:       else
33:         j := j + 1
34:       end if
35:     end if
36:   end if
37: end loop
38: declare success.
```

then the algorithm declares SUCCESS. Example 2 illustrates the workings of our algorithm R-BOUND-MP-NFRNS.

Example 2: Consider 4 tasks with $\{(T_1 = 1, C_1 = 0.1), (T_2 = 1.1, C_2 = 0.935), (T_3 = 1.2, C_3 = 0.084), (T_4 = 1.3, C_4 = 0.26)$ to be scheduled on 2 processors using R-BOUND-MP-NFRNS. The algorithm sorts the tasks in ascending order of periods. In this example, sorting does not change the indices. We can compute the utilizations of tasks: $u_1 = 0.1$,$u_2 = 0.85$,$u_3 = 0.07$ and $u_4 = 0.2$.

The current processor is processor 1. (The variable j, initialized on line 3 in Algorithm 1 keeps track of this.) Tasks are now assigned in order. τ_1 is assigned to processor 1. Then τ_2 is attempted to be assigned to processor 1, but it fails because the $T_2/T_1 = 1.1$, and $n_1 + 1 = 2$ gives a utilization bound 0.915 for these two tasks, and the sum of utilization of these two tasks is 0.95. Hence τ_2 is assigned to processor 2.

Now, processor 2 is the current processor. τ_3 is attempted to be assigned to processor 2, and it succeeds because $T_3/T_2 = 1.2/1.1 = 1.09$, and $n_2 + 1 = 2$ gives a utilization bound 0.922 for these two tasks, and the sum of utilization of these two tasks is 0.92.

Processor 2 is still the current processor. τ_4 is attempted to be assigned to processor 2, but it fails because $\max(T_2, T_3, T_4)/\min(T_2, T_3, T_4) = 1.3/1.1 = 1.18$ and $n_2 + 1 = 3$ gives a utilization bound 0.86 for these three tasks, and the sum of utilization of these three tasks is 1.12. Since processor 2 is the last processor and τ_4 failed, we make an attempt to assign τ_4 to the first processor, that is, processor 1. This succeeds because $n_1 + 1 = 2$ gives a utilization bound 0.828 for these two tasks, and the sum of utilization of these two tasks is 0.3. Hence τ_2 is assigned to processor 1. □

Now that we have stated the algorithm R-BOUND-MP-NFRNS and seen its operation in an example, we are ready to prove its performance. Theorem 2 does that.

Theorem 2 (Utilization bound of R-BOUND-MP-NFRNS): If R-BOUND-MP-NFRNS is used and $T_1 \leq T_2 \leq \ldots \leq T_n$ and $T_n/T_1 < 2$ and $\frac{1}{m}\sum_{i=1}^{n} u_i \leq 1/2$, then R-BOUND-MP-NFRNS will find a partitioning (declare SUCCESS).

Proof: We will derive a lower bound on the utilization of task sets that declared failure. We will do so by first phrasing necessary conditions on a task set that declared failure. We will then formulate a minimization problem which offers a lower bound on the utilization of a task set that declared failure. And then, we will state a sequence of other minimization problems where the objective function to each of them is a lower bound on the objective function to a previous minimization problem in the sequence.

Let us consider any arbitrary task set that caused R-BOUND-MP-NFRS to declare failure. If it was not the last task (the one with the longest period) that failed, then we can always remove the task that had a higher index than the failed task, and then the utilization would be lower. Hence, we can assume that it was the task with the greatest index that failed. Let τ_{failed} denote that task.

We will now consider the situation when R-BOUND-MP-NFRS failed and use the following notation. Let τ_{pjk} be the task that is the k^{th} task assigned to processor j. Let α_1 denote T_{p21}/T_{p11}. Let α_2 denote T_{p31}/T_{p21}. $\ldots$ Let α_m denote T_{failed}/T_{pm1}. Let n_j denote the number of tasks that are assigned to processor j. n_1 requires further explanation because we assign tasks to processor 1 in two states: first when no processor has been assigned a task, and later when all processors have been assigned a task. We let $n_1\prime$ denote the number of tasks assigned to processor 1 when R-BOUND-MP-NFRS declared failure. n_1 denotes the number of tasks assigned to processor 1 when τ_{p21} was assigned to processor 2.

Task τ_{p21} could not be assigned to processor 1 because the schedulability test in Theorem 1 failed. Hence, on processor 1 it holds that:

$$u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + u_{p21} > (n_1 + 1)(\alpha_1^{\frac{1}{n_1+1}} - 1) + \frac{2}{\alpha_1} - 1 \tag{1}$$

In the same way, on processor 2, it holds that:

$$u_{p21} + (\sum_{k=2}^{n_2} u_{p2k}) + u_{p31} > (n_2 + 1)(\alpha_2^{\frac{1}{n_2+1}} - 1) + \frac{2}{\alpha_2} - 1 \tag{2}$$

And so on, until processor m, where it holds that:

$$u_{pm1} + (\sum_{k=2}^{n_m} u_{pmk}) + u_{failed} > (n_m + 1)(\alpha_m^{\frac{1}{n_m+1}} - 1) + \frac{2}{\alpha_m} - 1 \tag{3}$$

Our algorithm R-BOUND-MP-NFRS attempts to assign τ_{failed} to processor 1. It fails so the schedulability test must have failed. Here we do not know anything about the relationships between the periods (other than $1 \leq \frac{T_{failed}}{T_i} < 2$). Hence we have:

$$u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed} > (n_1\prime + 1) \cdot (2^{\frac{1}{n_1\prime+1}} - 1) \tag{4}$$

Since we want to derive a utilization bound we have the following problem:

$$\begin{aligned} minimize\ U_s = \frac{1}{m} \cdot (&u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + \\ &u_{p21} + (\sum_{k=2}^{n_2} u_{p2k}) + \\ &\ldots + \\ &u_{pm1} + (\sum_{k=2}^{n_m} u_{pmk}) + (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed}) \end{aligned}$$

subject to Inequalities 1–4 and subject to

$$0 < u_{pij} \leq 1, \forall i, j \tag{5}$$

$$\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = T_{failed}/T_{p11} < 2 \tag{6}$$

$$1 \leq \alpha_i, \forall i \tag{7}$$

Note that the constraints Inequality 6 and Inequality 7 follow immediately from $T_1 \leq T_2 \leq \ldots \leq T_n$ and $T_n/T_1 < 2$, which we assumed in the theorem.

We make a relaxation on Inequalities 1–4 by replacing $>$ by $\geq$, relax Inequality 5 to $0 \leq u_{pij}$ and relax Inequality 6 by replacing $<$ by $\leq$.

One can see that $(n_i + 1)(\alpha_i^{1/(n_i+1)} - 1)$ monotonically decreases with increasing n_i. We can compute $\lim_{n_i \to \infty}(n_i + 1)(\alpha_i^{1/(n_i+1)} - 1) = \ln \alpha_i$. Hence we have:

$$(n_i + 1)(\alpha_i^{1/(n_i+1)} - 1) \geq \ln \alpha_i \tag{8}$$

In the same way, we have:

$$(n_i\prime + 1)(2^{1/(n_i\prime+1)} - 1) \geq \ln 2 \tag{9}$$

Using Inequality 8 and Inequality 9, we can relax Inequalities 1–4. All these relaxations change the constraints such that a point which satisfied all constraints will also satisfy the new constraints. We now have the problem:

$$\begin{aligned} minimize\ U_s = \frac{1}{m} \cdot (u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + \\ u_{p21} + (\sum_{k=2}^{n_2} u_{p2k}) + \\ \ldots + \\ u_{pm1} + (\sum_{k=2}^{n_m} u_{pmk}) + (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed}) \end{aligned}$$

subject to:

$$u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + u_{p21} \geq \ln \alpha_1 + 2/\alpha_1 - 1 \quad (10)$$

$$u_{p21} + (\sum_{k=2}^{n_2} u_{p2k}) + u_{p31} \geq \ln \alpha_2 + 2/\alpha_2 - 1 \quad (11)$$

$$\ldots$$

$$u_{pm1} + (\sum_{k=2}^{n_m} u_{pmk}) + u_{failed} \geq \ln \alpha_m + 2/\alpha_m - 1 \quad (12)$$

$$u_{p11} + (\sum_{k=2}^{n_1} u_{p1k}) + (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed} \geq \ln 2 \quad (13)$$

$$0 \leq u_{pij}, \forall i, j \quad (14)$$

$$\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m \leq 2 \quad (15)$$

$$1 \leq \alpha_i, \forall i \quad (16)$$

Note that we are not interested in finding every global minimizer. We simply want to find a global minimizer. Hence, at a minimizer, we could always move to a new point (with primed variables) which satisfies all constraints and does not increase the objective function in the following way:

$$u_{pi1}\prime = u_{pi1} + \sum_{k=2}^{n_i} u_{pik} \quad (17)$$

$$u_{pik}\prime = 0, \forall k \geq 2 \quad (18)$$

Note that u_{pij} is permitted to be greater than 1.

If $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m < 2$ then we can increase any α_i so that $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2$. This clearly does not affect the objective function. Neither does it violate any constraints because $\frac{\partial(\ln \alpha_i + 2/\alpha_i - 1)}{\partial \alpha_i}$ can be computed to $\frac{1}{\alpha_i^2} \cdot (\alpha_i - 2)$ and this is non-positive because $\alpha_i \leq 2$. $\alpha_i \leq 2$ follows from $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2$ and $1 \leq \alpha_i$. Hence we have the problem:

$$\begin{aligned} minimize\ U_s = \frac{1}{m} \cdot (u_{p11} + u_{p21} + \ldots + u_{pm1} + \\ (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed}) \end{aligned}$$

subject to:

$$u_{p11} + u_{p21} \geq \ln \alpha_1 + 2/\alpha_1 - 1 \quad (19)$$

$$u_{p21} + u_{p31} \geq \ln \alpha_2 + 2/\alpha_2 - 1 \quad (20)$$

$$\ldots$$

$$u_{pm1} + u_{failed} \geq \ln \alpha_m + 2/\alpha_m - 1 \quad (21)$$

$$u_{p11} + (\sum_{k=n_1+1}^{n_1\prime} u_{p1k}) + u_{failed} \geq \ln 2 \quad (22)$$

$$0 \leq u_{pij}, \forall i, j \quad (23)$$

$$\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2 \quad (24)$$

$$1 \leq \alpha_i, \forall i \quad (25)$$

We can add $(\sum_{k=n_1+1}^{n_1\prime} u_{p1k})$ to the lhs of Equality 21; every feasible point will remain feasible in this way. Then, we can always move to a new point (with variables having "new" in its superscript) which satisfies all constraints and does not increase the objective function in the following way:

$$u_{failed}^{new} = u_{failed} + \sum_{k=n_1+1}^{n_i\prime} u_{p1k} \quad (26)$$

$$u_{p1k}^{new} = 0, \forall k \geq n_1 + 1 \quad (27)$$

Note that u_{pij} and u_{failed} is permitted to be greater than 1.

Now the term $(\sum_{k=n_1+1}^{n_1\prime} u_{p1k})$ has disappeared from Equality 22. Note that in Inequalities 19–22, each variable u_{pik} and u_{failed} show up in exactly two constraints. Summing the left-hand side of Inequalities 19–22 and dividing by two gives us a lower bound on the objective function. We can also relax the problem by dropping Equality 23 and Equality 25. Hence we have the problem:

$$\begin{aligned} minimize\ U_s = \frac{1}{2m} \cdot (\ln 2 + \ln \alpha_1 + 2/\alpha_1 - 1 \\ + \ln \alpha_2 + 2/\alpha_2 - 1 + \ldots + \ln \alpha_m + 2/\alpha_m - 1) \end{aligned}$$

subject to:

$$\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2 \quad (28)$$

A necessary condition for a local minimizer is that the gradient of the Lagrangian function is zero (see for example Theorem 14.1 in [17]). Let λ denote the Lagrange multiplier for $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2$. Using this gives us that a necessary condition for a local minimizer is:

$$\frac{1}{m} \cdot (\frac{1}{2} \cdot (1/\alpha_1 - 2/\alpha_1^2)) - \lambda \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \ldots \cdot \alpha_m = 0$$

$$\frac{1}{m} \cdot (\frac{1}{2} \cdot (1/\alpha_2 - 2/\alpha_2^2)) - \lambda \cdot \alpha_1 \cdot \alpha_3 \cdot \alpha_4 \cdot \ldots \cdot \alpha_m = 0$$

Algorithm 2 Scale Task Set.

```
Input: A task set $\tau$. Output: Another task set $\tau\prime$.
1: $q = \max(T_1, T_2, \ldots, T_n)$
2: for each $i \in \tau$
3:     $T_i\prime = T_i \cdot 2^{\log_2(q/T_i)}$
4:     $C_i\prime = C_i \cdot 2^{\log_2(q/T_i)}$
5: end for
6: sort tasks in $\tau\prime$ in increasing period
7: return $\tau\prime$
```

$$\frac{1}{m} \cdot (\frac{1}{2} \cdot (1/\alpha_3 - 2/\alpha_3^2)) - \lambda \cdot \alpha_1 \cdot \alpha_2 \cdot \alpha_4 \cdot \ldots \cdot \alpha_m = 0$$

$$\ldots$$

$$\frac{1}{m} \cdot (\frac{1}{2} \cdot (1/\alpha_m - 2/\alpha_m^2)) - \lambda \cdot \alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \ldots = 0$$

Since a global minimizer is a local minimizer, the conditions are also necessary for a global minimizer.

Rewriting each of them and using $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_m = 2$ yields:

$$\frac{1}{m} \cdot (1 - 2/\alpha_1) = 4\lambda$$

$$\frac{1}{m} \cdot (1 - 2/\alpha_2) = 4\lambda$$

$$\ldots$$

$$\frac{1}{m} \cdot (1 - 2/\alpha_m) = 4\lambda$$

This implies that:

$$\alpha_1 = \alpha_2 = \ldots = \alpha_m$$

We now have the following problem:

$$minimize\ U_s = \frac{1}{2m} \cdot (\ln 2 + m \cdot (\ln \alpha_1 + 2/\alpha_1 - 1))$$

subject to $\alpha_1^m = 2$.

Rewriting yields:

$$minimize\ U_s = \frac{\ln 2}{2m} + \frac{1}{2} \cdot (\ln (2^{1/m}) + \frac{2}{2^{1/m}} - 1)$$

We compute $\frac{\partial U_s}{\partial m} < 0$ and $\lim_{m \to \infty} U_s = 1/2$. Hence we have that $U_s \geq 1/2$.

This states the theorem. ■

IV. Not restricted periods

In this section, we will see that if task periods are not restricted as they were in the previous section then it is possible to scale the periods and execution times of all tasks such that the restriction holds. This is meaningful because we will use a theorem which claims that if the scaled task set meets all deadlines then the task set which is not scaled also meets its deadlines.

Consider two task sets τ and $\tau\prime$. τ is not restricted. $\tau\prime$ is computed from τ according to Algorithm 2. Note that Algorithm 2 does not change the utilization of tasks. In addition we know that (from [10]):

Theorem 3: Given a task set τ, let $\tau\prime$ be the task set resulting from the application of the algorithm Scale Task Set to τ. If $\tau\prime$ is schedulable on one processor using rate-monotonic scheduling, then τ is schedulable on one processor with rate-monotonic scheduling.

Now let R-BOUND-MP-NFR (R-BOUND-MP with next-fit-ring) be an algorithm which works as follows. First, each task in τ is transformed according to Algorithm 2 into $\tau\prime$ and then the tasks in $\tau\prime$ are assigned according to R-BOUND-MP-NFRNS. We can see that every task in τ has a corresponding task in $\tau\prime$, so τ_i is assigned to the processor where $\tau_i\prime$ is. We are now ready to state our utilization bound of R-BOUND-MP-NFR when tasks are not restricted.

Theorem 4: If R-BOUND-MP-NFR is used and $\sum_{i=1}^{n} u_i \leq m/2$, then R-BOUND-MP-NFR will find a partitioning (declare SUCCESS).

Proof: The proof is by contradiction. Suppose that the theorem was false. Then there would exist a task set τ with $\sum_{i=1}^{n} u_i \leq m/2$ which failed. The first thing that R-BOUND-MP-NFR does is to scale the task set, so a scaled task set $\tau\prime$ will also declare failure when scheduled by R-BOUND-MP-NFRNS. Since u_i of a task does not change when it is scaled, we have that $\tau\prime$ (which failed) has $\sum_{i=1}^{n} u_i \leq m/2$. But this is impossible according to Theorem 2. ■

V. Quantifying the number of task sets that can be guaranteed

Previous sections, showed that the new algorithm increases the utilization bound of partitioning, from 41% to 50%. We will now see how many extra task sets that can be guaranteed to meet deadlines thanks to the increase in utilization bound. We do so using an approach from previous work on analysis of uniprocessor scheduling [21]. Let u_i be defined as $u_i = C_i/T_i$ and let $u = < u_1, u_2, \ldots, u_n >^T$. Then, the measure of the region of all task sets that is guaranteed by a utilization bound UB is defined as:

$$L_n(UB \cdot m) = \{u \in R^n : u_i \geq 0, \sum_{i=1}^{n} u_i \leq UB \cdot m\} \quad (29)$$

From [21] we obtain:

$$|L_n(A)| = \frac{A^n}{n!} \quad (30)$$

Combining Equality 30 and Equality 29 yields $L_n(UB \cdot m) = \frac{(UB \cdot m)^n}{n!}$. Analogous to [21], let the gain of the new test be defined as: $\rho_n = \frac{L_n(0.5 \cdot m)}{L_n((\sqrt{2}-1) \cdot m)}$. This gives us:

$$\rho_n = \left(\frac{0.5}{\sqrt{2}-1}\right)^n \approx 1.207106783^n \quad (31)$$

We can see that the gain approaches infinity as n approaches infinity. This is in contrast with work on uniprocessor schedulability analysis [21], which offered a finite gain $\sqrt{2}$. Hence, we conclude that the new bound offers a significant increase in the number of task sets that can be guaranteed as compared to the previously known best bound.

VI. Discussions and Future work

We have proven a tight utilization bound for static-priority preemptive partitioned static-priority scheduling. Our bound of 50% for partitioned static-priority scheduling is no worse than the best bound of partitioned scheduling using EDF on each uniprocessor [18]. This implies that although dynamic priorities are beneficial in scheduling algorithms with migration (see for example algorithm PF [19]), they offer no benefit in non-migrative scheduling if utilization bound is the performance metric of choice. We left open two important questions in partitioned scheduling: (i) Can R-BOUND-MP (the original algorithm, not our R-BOUND-MP-NFR) achieve a utilization bound of 50%? and (ii) Can other bin-packing schemes, which do not exploit harmonicity, achieve a utilization bound of 50%?

We assumed the restriction that a task has a deadline which is equal to its period. It would be interesting to create an algorithm for task set where this restriction does not hold. Unfortunately, for such a case, the notion of utilization bound does not apply; we have to resort to another performance metric. One such metric is the *competitive factor*. We say that a partitioning algorithm A has a competitive factor X_A if it can schedule every task set that any other scheduling algorithm A' can schedule if the processors provided to algorithm A is X_A times faster than the processors provided to the other algorithm. We can see that an algorithm with a utilization bound UB_A has a competitive factor $X_A = 1/UB_A$. Using this relationship gives our new algorithm a competitive factor of 2. Recently, the scheduling of tasks with static-priority preemptive scheduling using deadline monotonic [12] without the restriction on the deadline was considered [20, page 328]. Unfortunately, even allowing pseudo-polynomial time-complexity, the competitive factor of the proposed algorithm was 3.

Acknowledgment

The author is grateful to the reviewers for suggested improvements of the paper. This work was supported by the Swedish Foundation for Strategic Research via the national Swedish Real-Time Systems research initiative ARTES and by the Portuguese Science Foundation, Fundação para Ciência e Tecnologia (FCT).

References

[1] 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.

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, December 5–7, 2001.

[3] S. K. Dhall. *Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champain, 1977.

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

[5] S. Davari and S.K. Dhall. On a real-time task allocation problem. In *19th Annual Hawaii International Conference on System Sciences*, pages 8–10, Honolulu, Hawaii, 1985.

[6] S. Davari and S.K. Dhall. An on-line algorithm for real-time task allocation. In *Proc. of the IEEE Real-Time Systems Symposium*, volume 7, pages 194–200, New Orleans, LA, December 1986.

[7] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, December 1995.

[8] Y. Oh and S. H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, 9(3):207–239, November 1995.

[9] Y. Oh and S. H. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report 95-16, Department of Computer Science, University of Virginia, March 1995.

[10] S. Lauzac, R. Melhem, and D. Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. In *Proc. of the IEEE Int'l Parallel Processing Symposium*, pages 511–518, Orlando, Florida, March 1998.

[11] S. Sáez, J. Vila, and A. Crespo. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. In *10th Euromicro Workshop on Real Time Systems*, pages 53–60, Berlin, Germany, June 17–19, 1998.

[12] 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.

[13] J. M. López, J. L. Díaz, and D. F. García. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proc. of the EuroMicro Conference on Real-Time Systems*, pages 67–75, Delft, The Netherlands, June 13–15 2001.

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

[15] M. Joseph and P. Pandya. Finding response times in a real-time system. *Computer Journal*, 29(5):390–395, October 1986.

[16] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average behavior. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, California, December 5–7, 1989.

[17] S. G. Nash and A. Sofer. *Linear and Nonlinear optimization*. McGraw-Hill, 1996. ISBN 0-07-046065-5.

[18] J.M. López, M. García, J.L. Díaz, and D.F. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. of the 12th EuroMicro Conference on Real-Time Systems*, pages 25–33, Stockholm, Sweden, June 19–21, 2000.

[19] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[20] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the Real-Time Systems Symposium,*, pages 321–329, Miami, Florida, December 5–8, 2005.

[21] E. Bini, G. C. Buttazzo, and G. M. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *Proc. of the EuroMicro Conference on Real-Time Systems*, pages 59–66, Delft, The Netherlands, June 13–15 2001.