# Rate-monotonic scheduling on uniform multiprocessors*

Sanjoy Baruah Joël Goossens

**Abstract**

The rate-monotonic algorithm is arguably one of the most popular algorithms for scheduling systems of periodic real-time tasks. The rate-monotonic scheduling of systems of periodic tasks on uniform multiprocessor platforms is considered here. A simple, sufficient test is presented for determining whether a given periodic task system will be successfully scheduled by this algorithm upon a particular uniform multiprocessor platform — this test generalizes earlier results concerning rate-monotonic scheduling upon identical multiprocessor platforms.

**Keywords.** Uniform multiprocessors; periodic tasks; global scheduling; static priorities; rate-monotonic algorithm.

## 1 Introduction

In the periodic model of hard real-time tasks, a **task** $\tau_i = (C_i, T_i)$ is characterized by two parameters – an execution requirement $C_i$ and a period $T_i$ – with the interpretation that the task generates a *job* at each integer multiple of $T_i$, and each such job has an execution requirement of $C_i$ execution units, and must complete by a deadline equal to the next integer multiple of $T_i$. We assume that preemption is permitted – an executing job may be interrupted, and its exeuction resumed later, with no loss or penalty. A periodic task system consists of several independent such periodic tasks that are to execute on a specified preemptive processor architecture. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For each task $\tau_i$, define its *utilization* $U_i$ to be the ratio of $\tau_i$'s execution requirement to its period: $U_i \stackrel{\text{def}}{=} C_i/T_i$. We define the *cumulative utilization* $U(\tau)$ of periodic task system $\tau$ (often, the word "cumulative" is dropped and this is called simply the *utilization* of $\tau$) to be the sum of the utilizations of all tasks in $\tau$: $U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. Furthermore, we define the *maximum utilization* $U_{\max}(\tau)$ of periodic task system $\tau$ to be the largest utilization of any task in $\tau$: $U_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} U_i$.

Without loss of generality, we assume that $T_i \leq T_{i+1}$ for all $i$, $1 \leq i < n$; i.e., the tasks are indexed according to period.

*Run-time scheduling* is the process of determining, during the execution of a real-time application system, which job[s] should be executed at each instant in time. Run-time scheduling algorithms are typically implemented as follows: at each time instant, assign a **priority** to each active (informally, a job becomes *active* at its ready time, and remains so until it has executed for an amount of time equal to its execution requirement, or until its deadline has elapsed) job, and allocate the available processors to the highest-priority jobs.

*Supported in part by the National Science Foundation (Grant Nos. CCR-9704206, CCR-9972105, CCR-9988327, and ITR-0082866).

For systems comprised of periodic tasks that are to execute upon a *single shared processor*, a very popular run-time scheduling algorithm is the *rate-monotonic scheduling algorithm* (Algorithm RM )[9]. Algorithm RM assigns each task a priority inversely proportional to its period – the smaller the period, the higher the priority, with ties broken arbitrarily but in a consistent manner: if $\tau_i$ and $\tau_j$ have equal periods and $\tau_i$'s job is given priority over $\tau_j$'s job once, then all of $\tau_i$'s jobs are given priority over all of $\tau_j$'s jobs.

In [1], the rate-monotonic scheduling of periodic real-time task systems upon a more general machine model than the uniprocessor model was studied. In *identical multiprocessors*, there are several identical processors upon which the jobs generated by the periodic tasks execute. As in the uniprocessor case, Algorithm RM upon identical multiprocessors assigns tasks priorities in inverse proportion to their periods, and selects for execution at each instant in time the (jobs of) the highest-priority tasks that have jobs awaiting execution. In this paper, we study the scheduling of periodic task systems upon an even more general machine model: the **uniform multiprocessor** model. In contrast to identical multiprocessors (in which all processors are assumed to be equally powerful), each processor in a uniform multiprocessor machine is characterized by a *speed* or computing capacity, with the interpretation that a job executing on a processor with speed $s$ for $t$ time units completes $(s \times t)$ units of execution. We consider the scheduling of hard-real-time systems on uniform multiprocessor platforms which allow for *preemptions* and *interprocessor migrations* (i.e., a job executing on a processor may be interrupted at any instant and its execution resumed later on the same or a different processor with no cost or penalty), but forbid *job-level parallelism* – at any instant in time, each job may be executing upon at most one processor.

## 2 Model; useful results

In this section, we (i) introduce some definitions and formal notation concerning scheduling upon uniform multiprocessor platforms, and (ii) present without proof previously-published results from elsewhere that will be used in this paper.

**Definition 1** Let $\pi$ denote a **uniform multiprocessor platform**.

- The number of processors comprising $\pi$ is denoted by $m(\pi)$.
- For all $i$, $1 \leq i \leq m(\pi)$, the speed (the computing capacity) of the $i$'th-fastest processor in $\pi$ is denoted by $s_i(\pi)$.
- The total computing capacity of all the processors in $\pi$ is denoted by $S(\pi)$: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.

■

**Definition 2 (Greedy scheduling algorithm.)** A uniform multiprocessor scheduling algorithm is said to be greedy if it satisfies all three of the following conditions.

1. It never idles a processor when there are jobs awaiting execution.
2. If it must idle some processor (because fewer active jobs are available than processors), then it idles the slowest processors. That is, if it is the case that at some instant $t$ the $j$'th-slowest processor is idled, then the $k$'th-slowest processor is also idled at that instant $t$, for all $k > j$.
3. It always executes higher-priority jobs upon faster processors. That is, if the $j$'th-slowest processor is executing job $J$ at time $t$, it must be the case that the priority of $J$ is no smaller than the priorities of the jobs (if any) executing on the $k$'th slowest processor, for all $k > j$.

■

In the remainder of this paper, we will assume that Algorithm RM is implemented in a greedy manner upon uniform multiprocessors.

We now define two important additional parameters of uniform multiprocessor platforms; as we will see later in this report, these parameters succinctly capture the characteristics of uniform multiprocessors that are particularly relevant in determining whether a task system is successfully scheduled upon the platform by Algorithm RM.

**Definition 3 ($\lambda$ and $\mu$)**

For any uniform multiprocessor platform $\pi$, we define a parameter $\underline{\lambda(\pi)}$ as follows:

$$\lambda(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i+1}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} . \tag{1}$$

For any uniform multiprocessor platform $\pi$, we define a parameter $\underline{\mu(\pi)}$ as follows:

$$\mu(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} . \tag{2}$$

■

**Real-time job instances.** At times, we find it convenient to represent a real-time system using a model that is somewhat more general than the periodic task model. We will then assume that a hard-real-time **instance** is modelled as a collection of independent jobs. Each **job** $J_j = (r_j, c_j, d_j)$ is characterized by an arrival time $r_j$, an execution requirement $c_j$, and a deadline $d_j$, with the interpretation that this job needs to execute for $c_j$ units over the interval $[r_j, d_j)$. (Thus, the periodic task $\tau_i = (C_i, T_i)$ generates an infinite sequence of jobs with parameters $(k \cdot T_i, C_i, (k+1) \cdot T_i)$, $k = 0, 1, 2, \ldots$; in the remainder of this paper, we will sometimes use the symbol $\tau$ itself to denote the infinite set of jobs generated by the tasks in periodic task system $\tau$.)

**Definition 4 ($\mathbf{W(A, \pi, I, t)}$.)** Let $I$ denote any collection of jobs, and $\pi$ any uniform multiprocessor platform. For any algorithm $A$ and time instant $t \geq 0$, let $W(A, \pi, I, t)$ denote the amount of work done by algorithm $A$ on jobs of $I$ over the interval $[0, t)$, while executing on $\pi$.■

The following theorem was proved in [7]; we will be using it later in this paper.

**Theorem 1 (From [7])** Let $\pi_o$ and $\pi$ denote uniform multiprocessor platforms. Let $A_o$ denote any uniform multiprocessor scheduling algorithm, and $A$ any greedy uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms $\pi_o$ and $\pi$:

$$S(\pi) \geq S(\pi_o) + \lambda(\pi) \cdot s_1(\pi_o) \tag{3}$$

then for any collection of jobs $I$ and any time-instant $t \geq 0$,

$$W(A, \pi, I, t) \geq W(A_o, \pi_o, I, t) . \tag{4}$$

## 3 RM-feasibility upon uniform processors

For any scheduling algorithm $A$ and any processor platform $\pi$, we say that $\tau$ is **$A$-feasible** upon $\pi$ if $\tau$ will meets all deadlines when scheduled upon $\pi$ by algorithm $A$. We say $\tau$ is **feasible** upon $\pi$ if it can be scheduled to meet all deadlines upon $\pi$ by an optimal algorithm. In this section, we obtain sufficient conditions for determining whether a whether any periodic task system is RM-feasible upon any given uniform multiprocessor platform.

Consider a periodic task system $\tau$ and a uniform multiprocessor platform $\pi$, and suppose that $\tau$ and $\pi$ satisfy the following relationship:

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \tag{5}$$

Let us consider the RM-scheduling of $\tau$ upon $\pi$.

Without loss of generality, assume that ties are broken by Algorithm RM such that $\tau_i$ has greater priority than $\tau_{i+1}$ for all $i$, $1 \leq i < n$. Notice that whether jobs of $\tau_k$ meet their deadlines under Algorithm RM depends only upon the jobs generated by the tasks $\{\tau_1, \tau_2, \ldots, \tau_k\}$, and are completely unaffected by the presence of the tasks $\tau_{k+1}, \ldots, \tau_n$. For $k = 1, 2, \ldots, n$, let us define the task-set $\tau^{(k)}$ as follows:

$$\tau^{(k)} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_k\}.$$

**Lemma 1** Task system $\tau^{(k)}$ is feasible on a uniform multiprocessor platform $\pi_o$ satisfying the conditions

1. $S(\pi_o) = U(\tau^{(k)})$, and
2. $s_1(\pi_o) = U_{\max}(\tau^{(k)})$.

**Proof:** Set $\pi_o$ equal to a uniform multiprocessor consisting of $k$ processors, with computing capacities equal to $C_1/T_1$, $C_2/T_2$, …, $C_k/T_k$ respectively. It is easy to see that $\tau^{(k)}$ is feasible upon this $\pi_o$: an optimal scheduling algorithm OPT would simply schedule each task exclusively upon the processor that has computing capacity equal to that task's utilization.■

¿From Condition 3 Lemma 1 above, and the fact that Algorithm RM is greedy, it follows that if parameters $S(\pi)$ and $\lambda(\pi)$ of uniform multiprocessor platform $\pi$ satisfy

$$S(\pi) \geq U(\tau^{(k)}) + \lambda(\pi) \cdot U_{\max}(\tau^{(k)}) \tag{6}$$

then it will be true that

$$\begin{aligned} & W(\mathsf{RM}, \pi, \tau^{(k)}, t) \geq W(\text{OPT}, \pi_o, \tau^{(k)}, t) \\ \equiv \quad & \left( W(\mathsf{RM}, \pi, \tau^{(k)}, t) \geq t \cdot \left( \sum_{j=1}^{k} U_j \right) \right) \end{aligned} \tag{7}$$

where $\pi_o$ and OPT are as described in the proof of Lemma 1.

Recall that we are assuming that $\pi$ and $\tau$ satisfy Condition 5. Since

$$\left(2 \cdot U(\tau) \geq 2 \cdot U(\tau^{(k)}) \geq U(\tau^{(k)}) \geq\right) \text{ and } \quad (\mu(\pi) \geq \lambda(\pi)\ ) \ ,$$

we may conclude that

$$\begin{aligned} & S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \ \text{(By Condition 5)} \\ \Rightarrow \quad & S(\pi) \geq U(\tau^{(k)}) + \lambda(\pi) \cdot U_{\max}(\tau^{(k)}) \end{aligned}$$

Hence, Inequality 7 is seen to be true for all $t \geq 0$; i.e., *at any time-instant t, the amount of work done on $\tau^{(k)}$ by Algorithm* RM *executing on $\pi$ is at least as much as the amount of work done on $\tau^{(k)}$ by* OPT *on $\pi_o$.*

**Lemma 2** All jobs of $\tau_k$ meet their deadlines when $\tau^{(k)}$ is scheduled upon $\pi$ using Algorithm RM.

**Proof:** Let us assume that the first $(\ell - 1)$ jobs of $\tau_k$ have met their deadlines under Algorithm RM; we will prove below that the $\ell$'th job of $\tau_k$ also meets its deadline. The correctness of Lemma 2 will then follow by induction on $\ell$, starting with $\ell = 1$.

The $\ell$'th job of $\tau_k$ arrives at time-instant $(\ell - 1)T_k$, has a deadline at time-instant $\ell T_k$, and needs $C_k$ units of execution. From Inequality 7, we have

$$W(\mathsf{RM}, \pi, \tau^{(k)}, (\ell - 1)T_k) \geq (\ell - 1)T_k \left( \sum_{j=1}^{k} U_j \right) \tag{8}$$

Also, at least $(\ell - 1) \cdot T_k \cdot (\sum_{j=1}^{k-1} U_j)$ units of this execution by Algorithm RM was of tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$ — this follows from the fact that exactly $(\ell - 1)T_k U_k$ units of $\tau_k$'s work has been generated prior to instant $(\ell - 1)T_k$; the remainder of the work executed by Algorithm RM must therefore be generated by $\tau_1, \tau_2, \ldots, \tau_{k-1}$.

The cumulative execution requirement of all the jobs generated by the tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$ that arrive prior to the deadline of $\tau_k$'s $\ell$'th job is bounded from above by

$$\begin{aligned} & \sum_{j=1}^{k-1} \left\lceil \frac{\ell T_k}{T_j} \right\rceil C_j \\ < \; & \sum_{j=1}^{k-1} \left( \frac{\ell T_k}{T_j} + 1 \right) C_j \\ = \; & \ell T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \end{aligned}$$

As we have seen above (the discussion following Inequality 8) at least $(\ell - 1) \cdot T_k \cdot \sum_{j=1}^{k-1} U_j$ of this gets done prior to time-instant $(\ell - 1)T_k$; hence, at most

$$\left( T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \tag{9}$$

remains to be executed *after* time-instant $(\ell - 1)T_k$.

The amount of processor capacity left unused by $\tau_1, \ldots, \tau_{k-1}$ during the interval $[(\ell - 1)T_k, \ell T_k)$ is therefore no smaller than

$$S(\pi) \cdot T_k - \left( T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \tag{10}$$

Observe that *not all of this capacity is available* to $\tau_k$'s $\ell$'th job; in particular, at any instant at which several processors are simultaneously available, $\tau_k$ can execute upon only one of these processors and the excess capacity upon the remaining processors is wasted. So how much of this processor capacity identified in Expression (10) above is guaranteed to be available to $\tau_k$? To determine this observe that, since Algorithm RM is a greedy scheduling algorithm, if several processors are simultaneously available, then $\tau_k$ will execute upon the *fastest* available processor. Suppose that $\tau_k$ is executing upon the $j$'th-fastest processor at some instant in time (for some $j$, $1 \leq j \leq m(\pi)$); then the $(j+1)$'th-, $(j+2)$'th-, ..., $m$'th-fastest processors may all be idled at this instant, but all processors that are faster than the $j$'th faster processor are busy and do not contribute to the excess capacity computed above. That is, the fraction of excess capacity that is used at this instant is at least $(s_j(\pi)/[s_j(\pi) + s_{j+1}(\pi) + \cdots + s_{m(\pi)}(\pi)])$. By definition of the parameter $\mu(\pi)$ (Definition 3), this is $\geq \frac{1}{\mu(\pi)}$. Therefore, a lower bound on the amount of execution available to $\tau_k$'s $\ell$'th job over $[(\ell - 1)T_k, \ell T_k)$ is given by

$$\frac{1}{\mu(\pi)} \cdot \left[ S(\pi) \cdot T_k - \left( T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \right]$$

For the $\ell$'th job of $\tau_k$ to meet its deadline, it suffices that this amount be at least as large at $\tau_k$'s execution requirement; i.e.,

$$\frac{1}{\mu(\pi)} \cdot \left[ S(\pi) \cdot T_k - \left( T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \right] \geq C_k$$

which is true by definition of $U_{\max}(\tau)$:

$$U_{\max}(\tau) \geq U_k$$

$$\equiv \frac{1}{\mu(\pi)} \cdot [U_{\max}(\tau) \cdot \mu(\pi)] \geq U_k$$

*(By Condition 5, and the fact that* $\sum_{j=1}^{k-1} U_j \leq U(\tau)$*)* $\Rightarrow$

$$\frac{1}{\mu(\pi)} \cdot \left[ S(\pi) - 2 \cdot \sum_{j=1}^{k-1} U_j \right] \geq U_k$$

*(Since* $T_k \geq T_j$ *for* $j < k$*)* $\Rightarrow$

$$\frac{1}{\mu(\pi)} \cdot \left[ S(\pi) - \left( \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} \frac{C_j}{T_k} \right) \right] \geq U_k$$

and hence proves the property. ■

**Theorem 2** Given a periodic task system $\tau$ and a uniform multiprocessor platform $\pi$,

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau)$$

is a sufficient condition for ensuring that $\tau$ is RM-feasible upon $\pi$.

**Proof:** When $\tau$ and $\pi$ satisfy the condition of the theorem (which is exactly Condition 5), we have shown (in Lemma 2 above) that Algorithm RM schedules $\tau^{(k)}$ in such a manner that all jobs of the lowest-priority task $\tau_k$ complete by their deadlines. The correctness of Theorem 2 immediately follows, by induction on $k$. ■

We can apply Theorem 2 to identical multiprocessors to obtain a result similar to a result in [1]:

**Corollary 1** Any periodic task system in which each task's utilization is no more than one-third, and the sum of the utilizations of all the tasks is no more than $m/3$, is successfully scheduled by Algorithm $\mathsf{RM}$ upon $m$ unit-capacity processors.

**Proof:** Let $\tau$ denote the periodic task system: by the antecedents of the corollary, $U(\tau) \leq m/3$ and $U_{\max}(\tau) \leq 1/3$. Let $\pi$ denote an $m$-processor identical multiprocessor platform comprised of unit-capacity processors; by definition,

$$\mu(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} \quad = \quad \left\{ \frac{m}{1} \right\} \quad = \quad m \, .$$

By Theorem 2, $\tau$ is $\mathsf{RM}$-feasible upon $\pi$ if

$$\begin{aligned} & S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \\ \equiv \quad & \left( m \geq 2 \cdot \frac{m}{3} + m \cdot \frac{1}{3} \right) \\ \equiv \quad & (m \geq m) \ , \end{aligned}$$

which is true. ■

# 4 Context & Conclusions

Ou contribution in this paper can be summarized as follows: *we have obtained here the first non-trivial feasibility test for a* **static-priority** *scheduling algorithm that adopts a* **global***, rather than partitioned, approach to task allocation.* We now explain these terms, and thus place our findings in the proper context with respect to related research.

**Dynamic and static priorities** As stated in the introduction, run-time scheduling algorithms typically assign a priority to each active job at at each time instant, and allocate the available processors to the highest-priority jobs. With respect to certain run-time scheduling algorithms, it is possible that some tasks $\tau_i$ and $\tau_j$ both have active jobs at times $t_1$ and $t_2$ such that at time $t_1$, $\tau_i$'s job has higher priority than $\tau_j$'s while at time $t_2$, $\tau_j$'s job has higher priority than $\tau_i$'s. Run-time scheduling algorithms that permit such "switching" of the order of priorities between tasks are known as **dynamic** priority algorithms. The earliest deadline first scheduling algorithm (Algorithm $\mathsf{EDF}$) [9, 5] is an example of a dynamic priority algorithm. By contrast, **static** priority algorithms (such as Algorithm $\mathsf{RM}$, studied in this paper) must satisfy the constraint that for every pair of tasks $\tau_i$ and $\tau_j$, whenever $\tau_i$ and $\tau_j$ both have active jobs, it is always the case that the same task's jobs have priority.

**Partitioned and global scheduling.** In designing scheduling algorithms for multiprocessor environments, one can distinguish between at least two distinct approaches. In **partitioned** scheduling, all jobs generated by a task are required to execute on the *same* processor. **Global scheduling**, by contrast, permits *task migration* (i.e., different jobs of an individual task may execute upon different processors) as well as job-migration: an individual job that is preempted may resume execution upon a processor different from the one upon which it had been executing prior to preemption.

It has been proven by Leung and Whitehead [8] that the partitioned and global approaches to static-priority scheduling on identical multiprocessors are *incomparable*, in the sense that (i) there are task systems that are feasible on $m$ identical processors under the partitioned approach but for which no priority assignment exists which would cause all jobs of all tasks to meet their deadlines under global scheduling on the same $m$ processors; and (ii) there are task systems that are feasible on $m$ identical processors under the global approach, but which cannot be partitioned into $m$ distinct subsets such that each individual partition is uniprocessor static-priority feasible. This result of Leung and Whitehead [8] provides a very strong motivation to study both the partitioned and the non-partitioned approaches to static-priority multiprocessor (identical as well as uniform) scheduling, since it is provably true that neither approach is strictly better than the other.

**Related work.** Real-time scheduling upon uniform multiprocessors is a relatively new area of research. In [2], the dynamic-priority scheduling of periodic task systems upon uniform multiprocessors subject to certain preemption constraints (formalized as the *integer boundary constraint*) was studied: the general problem was proven intractable. In [7], on-line scheduling of collections of independent jobs upon uniform multiprocessors was considered, and sufficient conditions obtained for determining whether an instance of jobs known to be feasible upon a particular uniform multiprocessor platform would be EDF-feasible upon a different uniform multiprocessor platform (where "EDF" denotes the Earliest Deadline First scheduling algorithm). to our knowledge, there has been no prior work done on static-priority scheduling upon uniform multiprocessors

The partioned approach to static-priority scheduling upon has been extensively studied [6, 3, 4, 10] for identical multiprocessors. Much of this research has adopted the approach of using bin-packing like algorithms for partitioning a given set of periodic tasks among a set of processors such that each partition is uniprocessor feasible under the rate-monotonic algorithm; e.g., Dhavri and Dhall [3] presented an efficient algorithm which they proved would partition a set of periodic tasks into no more than twice as many partitions that an optimal algorithm would (equivalently, they devised an efficient algorithm for partitioned static-priority multiprocessor scheduling that uses at most twice as many processors as an optimal algorithm would). More recently, Oh and Baker [10] presented a partitioned static-priority multiprocessor scheduling algorithm that schedules any task system with utilization $\leq m(\sqrt{2}-1)$ on $m$ processors — this represents a utilization of approximately $42\%$ of the capacity of the multiprocessor platform. We expect that these results can be generalized to uniform multiprocessors, by applying techniques that have been developed for the bin-packing problem when bin-sizes are not the same.

The global approach to the static-priority scheduling of periodic task systems upon identical multiprocessors was studied in [1], and conditions were derived for determining whether a given periodic task system will be scheduled to meet all deadlines by Algorithm $\mathsf{RM}$ upon $m$ unit-capacity processors. It was shown that any periodic task system in which each task's utilization is no more than one-third, and the sum of the utilizations of all the tasks is no more than $m/3$, is successfully scheduled by Algorithm $\mathsf{RM}$ upon $m$ unit-capacity processors — to our knowledge, this is the first (and so far, only) non-trivial utilization-based static-priority feasibility condition for the global scheduling of periodic real-time tasks upon multiprocessors.

**This paper.** We have considered here the preemptive global static-priority scheduling of systems of periodic tasks on a uniform multiprocessor platform. We have obtained studied the behaviour of one such global static-priority scheduling algorithm — Algorithm $\mathsf{RM}$ — upon uniform multiprocessor platforms (since Algorithm $\mathsf{RM}$ has proven tremendously popular in uniprocessor scheduling, this seemed a reasonable starting point to our study). We have been reasonably successful in our study, in the sense that we obtained simple sufficient conditions for determining whether any given periodic task system will be succesfully scheduled by Algorithm $\mathsf{RM}$ upon a given uniform multiprocessor platform.

## References


[1] ANDERSSON, B., BARUAH, S., AND JANSSON, J. Static-priority scheduling on multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 193–202.

[2] BARUAH, S. Scheduling periodic tasks on uniform processors. In *Proceedings of the Euromicro Workshop on Real-time Systems* (Stockholm, Sweden, June 2000), pp. 7–14.

[3] DAVARI, S., AND DHALL, S. K. On a real-time task allocation problem. In *Proceedings of the 19th Hawaii International Conference on System Science* (Honolulu, January 1985).

[4] DAVARI, S., AND DHALL, S. K. An on-line algorithm for real-time tasks allocation. In *Proceedings of the Real-Time Systems Symposium* (1986), pp. 194–200.

[5] DERTOUZOS, M. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress* (1974), pp. 807–813.

[6] DHALL, S. K., AND LIU, C. L. On a real-time scheduling problem. *Operations Research 26* (1978), 127–140.

[7] FUNK, S., GOOSSENS, J., AND BARUAH, S. On-line scheduling on uniform multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 2001), IEEE Computer Society Press, pp. 183–192.

[8] LEUNG, J., AND WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

[9] LIU, C., AND LAYLAND, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (1973), 46–61.

[10] OH, D.-I., AND BAKER, T. P. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing 15* (1998), 183–192.