Real-Time Systems manuscript No.
(will be inserted by the editor)

Algorithms for Implementing Elastic Tasks on Multiprocessor Platforms

A Comparative Evaluation

James Orr · Sanjoy Baruah

Received: date / Accepted: date

Abstract The elastic task model enables the adaptation of systems of recurrent real-time tasks under uncertain or potentially overloaded conditions. A range of permissible periods is specified for each task in this model; during runtime a period is selected for each task from the specified range of permissible periods to ensure schedulability in a manner that maximizes the quality of provided service. This model was originally defined for sequential tasks executing upon a preemptive uniprocessor platform; here we consider the implementation of sequential tasks upon multiprocessor platforms. We define algorithms for scheduling sequential elastic tasks under the global and partitioned paradigms of multiprocessor scheduling for both dynamic and static-priority tasks, and we provide an extensive simulation-based comparison of the different approaches.

Keywords elastic scheduling · multi-processor scheduling · real-time systems · scheduling algorithms · simulation

1 Introduction

Buttazzo et al. introduced the *elastic task model* as a way of modeling recurrent real-time tasks, such as multimedia players or adaptive control systems, whose periods can change depending on the stress on the system [6]. The authors compare real-time tasks to physical springs, where changing a task's period

This research was supported in part by NSF grants CSR-1911460 Medium "Resource Efficient Implementation of Mixed-Criticality Systems" and CSR-1814739 Small "Dynamically Customizable Safety Critical Embedded Systems'.

Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045
One Brookings Drive
St. Louis, MO 63130
Tel.: 314-935-6160
E-mail: james.orr, baruah@wustl.edu

(and therefore processor utilization) is analogous to changing the length of the spring, and keeping the system-wide processor utilization below a certain value is analogous to compressing multiple contiguous springs to below a cumulative length. As originally presented, elastic scheduling seeks to schedule a task set on a single preemptive processor. Each such task has a worst-case execution time and a range of acceptable periods, rather than a single period parameter (as in the original Liu and Layland recurrent task model [21]). Each task must be assigned a period within its acceptable range such that the overall task set utilization remains below a desired value. To determine the appropriate period value to assign each task, every task also has an *elastic coefficient* which acts as an indicator of the task's resistance to increasing its period from the minimum (and desired) period, analogous to a spring's resistance to being compressed.

In the decades since the elastic task model was introduced, real-time systems have increasingly come to be implemented upon platforms comprising multiple processors, which enables the exploitation of both inter-task and intra-task parallelism. It is therefore entirely appropriate that the elastic task model should also be extended to consider multiprocessors. We have previously extended the elastic task model to include scheduling of tasks with intra-task parallelism on heterogeneous multi-core systems under the federated scheduling paradigm [26]. We have also initiated the process of better understanding how to implement sequential tasks (i.e., those whose internal parallelism is not exposed to the scheduler) upon homogeneous multi-core systems, and have performed some experiments comparing several different algorithms that we had proposed for this purpose [24]. In this paper we expand upon our previous work [24] and perform a more thorough study of the issue of implementing sequential elastic tasks upon multi-core platforms. We present additional algorithms for scheduling systems of such tasks upon a homogeneous multiprocessor platform under both the global and partitioned paradigms of multiprocessor scheduling for tasks with both static and dynamic priority assignments. We compare the effectiveness of the different algorithms via an extensive series of simulation experiments; based upon the outcomes of these simulations, we make some recommendations regarding the choice of algorithms for the multiprocessor scheduling of sequential elastic tasks.

Organization. The remainder of this paper is structured as follows. Section 2 presents our task model. Sections 3 and 4 present algorithms for the global and partitioned scheduling of systems of sequential elastic tasks respectively. Section 5 details our experimental evaluation of the different schemes. Section 6 describes related work, and Section 7 concludes and provides future direction.

2 Task Model and Assumptions

In the elastic model for recurrent real-time processes that was proposed by Buttazzo et al. [6], each individual elastic task τ_i is characterized by four parameters C_1, $T_i^{(\min)}, T_i^{(\max)}$, and E_i, where

- C_1 is the worst-case execution time (WCET) of each job of task τ_i;
- $T_i^{(\min)}$ is the minimum –and preferred– period at which successive jobs of τ_i should be invoked;
- $T_i^{(\max)}$ is the largest acceptable period; and
- E_i is the *elasticity coefficient* – a measure of a task's resistance to changing its period, analogous to a spring's resistance to changing its length.

Most of the scheduling approaches that we will be considering in this paper have *utilization-based* schedulability conditions: only the utilization parameters of tasks appear in these schedulability conditions. We therefore find it convenient to convert the period parameters of each task (the $T_i^{(\min)}$ and $T_i^{(\max)}$ parameters) to corresponding utilization parameters $U_i^{(\max)}$ and $U_i^{(\min)}$ respectively:

$$U_i^{(\max)} = C_i/T_i^{(\min)}$$
$$U_i^{(\min)} = C_i/T_i^{(\max)}$$

In the remainder of this manuscript, we have therefore chosen to characterize each task τ_i by three parameters rather than four, as follows

$$\tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i)$$

In this work we seek to schedule a set of n such independent sequential elastic tasks $\Gamma = \tau_1 ... \tau_n$ on m homogeneous processors.

Example 1 Running example for this paper.

τ_1 has $C_1 = 4$, $T_1^{(\min)} = 5$, $T_1^{(\max)} = 20$, and $E_1 = 1$. Translates to $\tau_1 = (0.8, 0.2, 1)$.

System Γ comprises four tasks, with parameters as follows:

τ_i	$U_i^{(\max)}$	$U_i^{(\min)}$	E_i
τ_1	0.8	0.2	1
τ_2	0.8	0.2	2
τ_3	0.8	0.2	3
τ_4	0.8	0.2	4

to be scheduled on $m = 2$ processors. □

In systems such as the one illustrated in Example 1 above where the $U_i^{(\max)}$ parameters of the tasks sum to greater than the number of processors, it is not possible to schedule each task to receive an actual utilization equal to its desired maximum utilization. We seek instead to schedule such systems such that the utilization of each task τ_i is as close to its desired maximum utilization as possible. Let U_i denote the actual utilization "allocated" to each task τ_i; we would like to have U_i be as close to $U_i^{(\max)}$ as possible (equivalently, minimize $(U_i^{(\max)} - U_i)$) for all tasks τ_i. The elasticity (E_i) parameters denote

the relative tolerances of the different tasks to receiving a lower-than-desired utilization: the amounts by which tasks' allocated utilizations are reduced from their desired maximums should be in proportion to their elasticity coefficients:

$$\forall\ i, j,\quad \left(\frac{U_i^{(\max)} - U_i}{E_i}\right) = \left(\frac{U_j^{(\max)} - U_j}{E_j}\right) \tag{1}$$

Letting λ denote the ratio $\frac{U_i^{(\max)} - U_i}{E_i}$ in Expression 1 above:

$$\lambda \stackrel{\text{def}}{=} \left(\frac{U_i^{(\max)} - U_i}{E_i}\right) \tag{2}$$

the optimization objective in elastic scheduling is to have λ be as small as possible (equivalently, have U_i be as close to $U_i^{(\max)}$ as possible).

Example 2 The example instance of Example 1, scheduled using fluid scheduling (Section 3.1). The desired solution has $U_1 = 0.68, U_2 = 0.56, U_3 = 0.44$, and $U_4 = 0.32$, since

$$U_1 + U_2 + U_3 + U_4 = 0.68 + 0.56 + 0.44 + 0.32 = \mathbf{2}$$

and it may be verified that

$$\left(\frac{0.80 - 0.68}{1}\right) = \left(\frac{0.80 - 0.56}{2}\right) = \left(\frac{0.80 - 0.44}{3}\right) = \left(\frac{0.80 - 0.32}{4}\right) = 0.12$$

In this example, the value of λ (as defined in Expression 2) is 0.12 □

Recall that $U_i^{(\min)}$ denotes a lower bound on the acceptable values of U_i, in the sense that no task τ_i may be allocated an actual utilization smaller than U_i.

Example 3 Suppose that the parameter $U_4^{(\min)}$ of the example task system of Example 1 were equal to 0.5 (rather than 0.2 as specified in Example 1). The actual allocation $U_1 = 0.68, U_2 = 0.56, U_3 = 0.44$, and $U_4 = 0.32$ of Example 2 is then no longer a valid one since U_4, at 0.32 is smaller than its minimum acceptable value of 0.5. So the algorithm of Buttazzo et al. [6] would assign U_4 a value equal to $U_4^{(\min)}$ of 0.5, and seek to assign the $(2 - 0.5 =)$ **1.5** units of remaining utilization amongst τ_1, τ_2, and τ_3 in proportion to their elasticity parameters. This results in assigning U_1, U_2, and U_3 the values of $0.65, 0.5$, and 0.35 respectively since

$$U_1 + U_2 + U_3 = 0.65 + 0.5 + 0.35 = \mathbf{1.5}$$

and

$$\left(\frac{0.80 - 0.65}{1}\right) = \left(\frac{0.80 - 0.50}{2}\right) = \left(\frac{0.80 - 0.35}{3}\right) = 0.15$$

Here, λ (as defined in Expression 2) takes on the value 0.15. □

By algebraic simplification of Expression 1, we have

$$U_i \leftarrow U_i^{(\max)} - \lambda E_i$$

However, as illustrated in Example 3 we also require $U_i \geq U_i^{(\min)}$. For a given value of λ, we therefore define $U_i(\lambda)$ as follows:

$$U_i(\lambda) \leftarrow \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)}\right) \tag{3}$$

Example 4 Consider again the example instance of Example 1 as modified in Example 3. For $\lambda = 0.15$, we have

$$U_1(0.15) = \max(0.8 - 0.15 \times 1, 0.2) = \max(0.8 - 0.15, 0.2) = \max(0.65, 0.2) = 0.65$$
$$U_2(0.15) = \max(0.8 - 0.15 \times 2, 0.2) = \max(0.8 - 0.30, 0.2) = \max(0.50, 0.2) = 0.50$$
$$U_3(0.15) = \max(0.8 - 0.15 \times 3, 0.2) = \max(0.8 - 0.45, 0.2) = \max(0.35, 0.2) = 0.35$$
$$U_4(0.15) = \max(0.8 - 0.15 \times 4, 0.5) = \max(0.8 - 0.60, 0.5) = \max(0.20, 0.5) = 0.50$$

as claimed in Example 3. □

The problem considered. Note that for a given value of λ, an elastic task $\tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i)$ is just a "regular" Liu and Layland task with utilization $U_i(\lambda)$ as given by Expression 3 above. For each of the multiprocessor scheduling strategies we will study in this paper, the question we ask is: given a task system

$$\Gamma = \left\{\tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i)\right\}_{i=1}^{n}$$

that is to be scheduled upon an m-processor platform, what is the smallest value of λ for which the Liu and Layland task system comprising n tasks with utilizations $U_1(\lambda)$, $U_2(\lambda)$, ..., $U_n(\lambda)$ is successfully schedulable by that particular scheduling strategy?

3 Global Scheduling

Under the global paradigm of multiprocessor scheduling for recurrent tasks, individual tasks are not restricted to executing upon specific processors. Instead, a newly-arrived job of a task may begin execution upon any available processor and a preempted job may resume execution at a later point in time upon any processor, not just the one it had been executing upon prior to preemption. We consider four different global scheduling algorithms: fluid (Section 3.1), Earliest Deadline First (Section 3.2) and an algorithm called PriD [15] that can be thought of as a generalization of EDF (Section 3.3) for fixed-job priority scheduling, and Global Rate Monotonic for fixed-task priority scheduling (Section 3.4).

3.1 Fluid Scheduling

The *fluid scheduling* paradigm of multiprocessor real-time scheduling permits that individual tasks be assigned a fraction f, $0 \leq f \leq 1$, of a processor at each instant in time (in contrast to non-fluid schedules, in which each task may execute either upon zero processors or upon a single processor at each instant). Fluid scheduling is a convenient abstraction that considerably simplifies many multiprocessor real-time scheduling problems; techniques are known (see, e.g, [23,16,3,20]) for converting fluid schedules to non-fluid ones for many problems and under a wide range of conditions and circumstances.

Fluid scheduling of Liu and Layland tasks – a review. Consider some Liu and Layland task system Γ, and let U_i denote the utilization of $\tau_i \in \Gamma$. It has been shown [16] that a necessary and sufficient condition for Γ to be fluid-schedulable upon a multiprocessor platform comprising m unit-speed processors is that

$$max_{\tau_i \in \Gamma}\{U_i\} \leq 1 \tag{4}$$

and

$$\left(\sum_{\tau_i \in \Gamma} U_i\right) \leq m \tag{5}$$

Any task system satisfying Conditions 4 and 5 can be fluid-scheduled by simply assigning each job of τ_i a fraction U_i of one of the m processors at each instant between its release date and its deadline.

Extension to period-elastic tasks. The actual utilization U_i of each task $\tau_i \in \Gamma$ when Γ is fluid-scheduled can be computed in the manner that was illustrated in Examples 3 and 4:

1. Let U_{tot} denote the total amount of computing capacity remaining to be allocated. Initially, $U_{\text{tot}} \leftarrow m$.
   Let Γ' denote the tasks for which actual utilizations (U_i's) have not been computed. Initially, $\Gamma' \leftarrow \Gamma$.
2. We compute U_i values for all the tasks in Γ' by (i) ignoring the lower bounds (the $U_i^{(\min)}$ values); and (ii) ascertaining that the U_i values so computed do not sum to more than U_{tot}.
   In the absence of the lower bounds, note that elastic scheduling seeks the smallest value of λ such that each task $\tau_i \in \Gamma'$ can be assigned a utilization U_i equal to

   $$U_i^{(\max)} - \lambda E_i \tag{6}$$

   Hence the total utilization for all the tasks equals

   $$\sum_{\tau_i \in \Gamma'} U_i^{(\max)} - \lambda \sum_{\tau_i \in \Gamma'} E_i$$

This total utilization is required to be $\leq$ the total available computing capacity U_{tot}:

$$\sum_{\tau_i \in \Gamma'} U_i^{(\max)} - \lambda \sum_{\tau_i \in \Gamma'} E_i \leq U_{\text{tot}}$$
$$\Leftrightarrow \lambda \geq \frac{\left(\sum_{\tau_i \in \Gamma'} U_i^{(\max)}\right) - U_{\text{tot}}}{\sum_{\tau_i \in \Gamma'} E_i}$$

The smallest λ satisfying the expression above is given by

$$\lambda \leftarrow \left\lceil \frac{\left(\sum_{\tau_i \in \Gamma'} U_i^{(\max)}\right) - U_{\text{tot}}}{\sum_i E_i} \right\rceil$$

This value of λ is used to compute the value of U_i for each $\tau_i \in \Gamma$ according to Expression 6 above.

3. If each such computed U_i value is $\geq$ the corresponding $U_i^{(\min)}$ value, then these are the desired U_i values and we are done.
4. Else, there are some $\tau_i \in \Gamma'$ for which the computed U_i value is strictly smaller than the corresponding $U_i^{(\min)}$ value. For each such τ_i
   - we assign it an actual utilization equal to its $U_i^{(\min)}$ value: $U_i \leftarrow U_i^{(\min)}$;
   - we subtract this assigned utilization from the total available capacity: $U_{\text{tot}} \leftarrow U_{\text{tot}} - U_i^{(\min)}$; and
   - we remove this task from the set Γ' of tasks for which it remains to compute the actual utilization: $\Gamma' \leftarrow \Gamma' \setminus \{\tau_i\}$.
5. We then repeat the process from Step 2 onwards.

Example 5 Reference Examples 3 and 4 here. □

The idea outlined above was codified in the original elastic scheduling paper [6], Buttazzo et al. as the iterative algorithm Task_Compress(Γ,U_d) that assigns a period to each task in Γ such that the total system utilization stays below a desired value U_d, that has running time $\Theta(|\Gamma|^2)$ — this algorithm is reproduced in this paper as Algorithm 1. It is evident that Algorithm 1 is, in essence, determining the smallest value of λ for which

$$\left(\sum_{i=1}^{n} U_i(\lambda)\right) \leq U_d \ ,$$

where the $U_i(\lambda)$s are as defined according to Expression 3. Observe, too, that Algorithm 1 never *increases* the actual utilization assigned to any any task τ_i to beyond $U_i^{(\max)}$ — this follows from the observation that in Line 19, the value assigned to the actual utilization —the parameter U_i— is obtained by *subtracting* a positive quantity from $U_1^{(\max)}$. Hence given an elastic task system Γ of sequential tasks that is to be fluid-scheduled upon m unit-speed processors, we can determine the effective utilizations of the individual tasks that satisfy

Algorithm 1 Task_Compress(Γ,U_d)

1: $U^{(\max)} = \sum_{i=1}^{n} C_i/T_i^{(\min)}$
2: $U^{(\min)} = \sum_{i=1}^{n} C_i/T_i^{(\max)}$
3: **if** $U_d < U^{(\min)}$ **then**
4: **return** INFEASIBLE
5: **end if**
6: $ok = 0$
7: **while** $ok == 0$ **do**
8: $U_f = E_v = 0$
9: **for** *each* τ_i **do**
10: **if** $E_i == 0$ **or** $T_i == T_i^{(\max)}$ **then**
11: $U_f = U_f + U_i$
12: **else**
13: $E_v = E_v + E_i$
14: **end if**
15: **end for**
16: $ok = 1$
17: **for** *each* $\tau_i \in \Gamma_v$ **do**
18: **if** $E_i > 0$ **and** $T_i < T_i^{(\max)}$ **then**
19: $U_i = U_i^{(\max)} - (U^{(\max)} - U_d + U_f) * E_i/E_v$
20: $T_i = C_i/U_i$
21: **if** $T_i > T_i^{(\max)}$ **then**
22: $T_i = T_i^{(\max)}$
23: $ok = 0$
24: **end if**
25: **end if**
26: **end for**
27: **end while**
28: **return** FEASIBLE

Conditions 4 and 5, and therefore bear witness to the fluid-schedulability of Γ, by simply calling the procedure Task_Compress(Γ,U_d) of Algorithm 1 with $U_d \leftarrow m$. The instance Γ can then be fluid-scheduled by assigning each job of each $\tau_i \in \Gamma$ a fraction of a processor equal to this effective utilization at each instant between its release date and its deadline.

3.2 Global EDF

While the fluid scheduling model is a convenient abstraction for considering multiprocessor scheduling, it is not in general directly implementable. As mentioned above, techniques are known for converting fluid schedules to non-fluid ones under a variety of conditions; however, most such conversions yield schedules with a large number of preemptions and inter-processor migrations. In environments in which there is a considerable overhead associated with each preemption and/ or inter-processor migration, this approach of obtaining a fluid schedule and then converting to a non-fluid one may incur unacceptably high overhead costs.

Review of results for Liu and Layland tasks. The global Earliest Deadline First (EDF) scheduling algorithm has the property that the total number of preemptions and inter-processor migrations in a schedule is bounded from above at the number of jobs in the schedule. (This is easily seen by observing that a job may preempt an already-executing one only upon its arrival, if it happens to have an earlier deadline; such preemption may later lead to an inter-processor migration if the preempted job resumes upon a different processor.) Global EDF may therefore be a more appropriate algorithm to use in environments characterized by significant preemption/ migration overhead costs. Goossens et al. showed [15, Theorem 5] that a system Γ of Liu & Layland tasks is scheduled by global EDF to meet all deadlines upon m unit-speed processors if the following condition holds:

$$\sum_{\tau_i \in \Gamma} U_i \leq m - (m-1) \times \max_{\tau_i \in \Gamma} \{U_i\} \tag{7}$$

(This condition was also shown [15, Theorem 6] to be tight from a utilization-based perspective: there are systems in which $\left(\sum_{\tau_i \in \Gamma} U_i\right)$ is greater than $(m - (m-1) \times \max_{\tau_i \in \Gamma} \{U_i\})$ by an arbitrarily small amount, upon which global EDF misses deadlines.)

Extension to period-elastic tasks. Given a system Γ of period-elastic tasks

$$\Gamma = \left\{ \tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i) \right\}_{i=1}^{n}$$

that is to be scheduled upon an m-processor platform, our objective is to find the smallest value of λ such that the Liu & Layland task system with the following utilizations

$$U_i \leftarrow \left\{ \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)} \right) \right\}_{i=1}^{n} \tag{8}$$

is schedulable using global EDF. We have chosen to solve this problem by iterating through the possible values of λ — see Algorithm 2. This algorithm steps through the range $[0, \Phi]$ with a "granularity" ϵ (Line 1 of Algorithm 2), where Φ is the maximum value among all tasks of the equation $\left(\frac{U_i^{(\max)} - U_i^{(\min)}}{E_i} \right)$. The algorithm seeks the smallest value of λ or which the Liu & Layland task system of Expression 8 above is global EDF-schedulable according to Expression 7. Once this smallest value of λ is determined and returned by Algorithm 2, we can convert the period-elastic task system to a regular Liu & Layland task system by computing the effective utilizations of the tasks according to Expression 3, and then schedule the Liu & Layland task system so obtained by global EDF. Algorithm 2 is $\Theta(\frac{\Phi}{\epsilon} \times n)$.

3.3 Algorithm PriD

It was observed [2] that global EDF tends to under-perform when there is even a single task with high utilization. This is easily explained by examining the

Algorithm 2 Global EDF(Γ,m)

1: $\epsilon \leftarrow 0.05 \times \Phi$ ▷ "Granularity" of the test...
2: **for** $\lambda \leftarrow 0$ **to** Φ **by** ϵ **do**
3: $S \leftarrow 0.0$ ▷ Total utilization of compressed tasks
4: $M \leftarrow 0.0$ ▷ Max. utilization amongst compressed tasks
5: **for** $i \leftarrow 1$ **to** $|\Gamma|$ **do**
6: $tmp \leftarrow \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)}\right)$
7: $S \leftarrow S + tmp$
8: $M \leftarrow \max(M, tmp)$
9: **end for**
10: **if** $(S \leq m - (m-1) \times M)$ **then**
11: ▷ By Eqn. 7, the compressed tasks are global-EDF schedulable,
12: **return** λ
13: **end if**
14: **end for**
15: **return** *(global EDF fails)*

Algorithm PriD (Γ, m)
The Liu & Layland task system $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$ is to be scheduled on m processors
Assume the tasks are indexed according to utilization: $U_i \geq U_{i+1}$ for all i, $1 \leq i < n$
for $i = 1$ **to** m **do**
if $\{\tau_{i+1}, \tau_{i+2}, \ldots, \tau_n\}$ is global-EDF schedulable upon $(m-i)$ processors
then
During run-time $\{\tau_1, \tau_2, \ldots, \tau_i\}$'s jobs will be assigned highest priority
and $\{\tau_{i+1}, \tau_{i+2}, \ldots, \tau_n\}$'s jobs will be assigned EDF-priority
return success
return failure // *Not schedulable by PriD*

Fig. 1 Algorithm PriD priority-assignment rule

utilization-based global-EDF schedulability condition of Inequality 7: observe the presence of the

$$\left((m-1) \times \max_{\tau_i \in \Gamma} \{U_i\}\right)$$

term on the right-hand side. Since this term is *subtracted* from the total computing capacity of the platform (i.e., m), the consequence is that a capacity of $(m-1)$ times the largest individual utilization becomes unavailable due to the presence of this large-utilization task. This phenomenon can be looked upon a consequence of the well-known *Dhall effect* [11,12] which has been widely studied in multiprocessor real-time scheduling theory. Several results have been obtained within the real-time scheduling theory community for dealing with such utilization loss; below we first review some of these results and then seek to extend their applicability to incorporate period-elasticity.

Review of results for Liu and Layland tasks. Recall that one major advantage of EDF-generated schedules over those obtained by converting a fluid-based one is the reduced number of preemptions and inter-processor migrations: the total number of preemptions and migrations in an EDF-generated

is no greater than the number of jobs that are scheduled. It turns out that this property is in fact enjoyed by an entire class of algorithms: all those in which each job is assigned a single fixed priority and at each instant during run-time the highest-priority jobs that are eligible to execute are selected for execution. Algorithms in this class are referred to as *Fixed Job Priority* (FJP) [9] scheduling algorithms. The algorithm fpEDF was proposed [2] as an FJP algorithm that circumvents the utilization loss caused by the Dhall effect. Under the fpEDF run-time scheduling algorithm, jobs of tasks with utilization > 0.5 are statically assigned highest priority while priorities to jobs of the remaining tasks are assigned according to EDF. It has been shown [2, Theorem 4] that a task system Γ is scheduled by fpEDF to meet all deadlines upon m unit-speed processors if the following condition holds:

$$\sum_{\tau_i \in \Gamma} U_i \leq \frac{m+1}{2} \tag{9}$$

A pragmatic improvement to fpEDF, called Algorithm PriD (for "priority driven") was proposed by Goossens et al. [15] — this is the algorithm that we will be adapting below for period-elastic tasks. Algorithm PriD is presented in pseudo-code form in Figure 1. Algorithm PriD, like fpEDF, seeks to circumvent the Dhall effect by assigning greatest priority to jobs of tasks with high utilization; however, while fpEDF designates all tasks with utilization > 0.5 to be "high-utilization" ones, Algorithm PriD determines which tasks are "high-utilization" based on the characteristics of the task system under consideration. It is shown [15] that Algorithm PriD strictly dominates fpEDF: all instances that are deemed schedulable by fpEDF are also deemed schedulable by PriD while the converse of this statement is not true – there are instances deemed schedulable by Algorithm PriD that will not pass the fpEDF schedulability test of Expression 9.

Extension to period-elastic tasks. Our adaptation of Algorithm PriD to period-elastic tasks is similar to our adaptation of global EDF: given an instance of periodic-elastic tasks

$$\Gamma = \left\{ \tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i) \right\}_{i=1}^{n}$$

to be scheduled upon m unit-speed processors, we iterate through possible values of λ between 0 and Φ, seeking the smallest value such that the Liu & Layland task system with utilizations

$$U_i \leftarrow \left\{ \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)} \right) \right\}_{i=1}^{n}$$

is deemed schedulable by Algorithm PriD upon m unit-speed processors. (The pseudo-code for this algorithm is very similar to the pseudo-code in Algorithm 2, and hence omitted.)

Algorithm PriD is $\Theta\Big(n \times log(n) + m\Big)$. Therefore, the overall complexity of iterating over λ values for elastic tasks to be scheduled under Algorithm PriD is $\Theta\Big(\frac{\Phi}{\epsilon} \times (n \times log(n) + m)\Big)$

3.4 Global Rate-Monotonic

Both Global EDF and Algorithm PriD consider tasks whose priorities may vary from job to job. Under *Global Rate-Monotonic* (RM) scheduling, all jobs of a task have a single fixed priority – tasks with a shorter period are given higher priority and will always preempt a task with a longer period.

Review of results for Liu and Layland tasks. The best known utilization bound for global (RM) scheduling was established by Bertogna et al. [5] A system Γ of Liu & Layland tasks is scheduled by global EDF to meet all deadlines upon m unit-speed processors if:

$$\sum_{\tau_i \in \Gamma} U_i \leq \frac{m}{2}\left(1 - \max_{\tau_i \in \Gamma}\{U_i\}\right) + \max_{\tau_i \in \Gamma}\{U_i\} \tag{10}$$

Once again the presence of the $\max_{\tau_i \in \Gamma}\{U_i\}$ term demonstrates the presence of the Dhall effect.

Extension to period-elastic tasks. The adaptation of global RM to period-elastic tasks is similar to our adaptation of global EDF and Algorithm PriD discussed earlier: given an instance of periodic-elastic tasks

$$\Gamma = \left\{\tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i)\right\}_{i=1}^{n}$$

to be scheduled upon m unit-speed processors, we iterate through possible values of λ between 0 and Φ, seeking the smallest value such that the Liu & Layland task system with utilizations

$$U_i \leftarrow \left\{\max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)}\right)\right\}_{i=1}^{n}$$

is deemed schedulable by global Rate-Monotonic upon m unit-speed processors. (The pseudo-code for this algorithm is again very similar to the pseudo-code in Algorithm 2, and hence omitted.) The overall complexity is the same as Algorithm 2, $\Theta(\frac{\Phi}{\epsilon} \times n)$.

4 Partitioned Scheduling

The partitioned scheduling of Liu & Layland task systems is known to be equivalent to the bin-packing problem[18,17], and hence NP-hard in the strong sense. Several polynomial-time heuristics have been proposed for solving this problem approximately: most of these heuristic algorithms for partitioning have the following common structure. First, they specify an order in which the tasks are to be considered. Then in considering each task (in the order chosen), they specify the order in which to consider upon which processor to attempt to allocate the task. A task is successfully allocated upon a processor if it is observed to "fit" upon the processor; within the context of the partitioned EDF-scheduling, a task fits on a processor if the task's utilization does not

exceed the processor capacity minus the sum of the utilizations of all tasks previously allocated to the processor. The algorithm declares success if all tasks are successfully allocated; otherwise, it declares failure.

Lopez et al. [22] have extensively compared several widely-used heuristic algorithms that fit this overall structure. They define the concept of a *Reasonable Allocation* (RA) partitioning algorithm: an RA algorithm is one that fails to allocate a task to a multiprocessor platform only when the task does not fit into any processor upon the platform. All the heuristic algorithms considered by Lopez et al. [22] are RA ones — indeed, there seems to be no reason why a system designer would ever consider using a non-RA partitioning algorithm. Within the RA algorithms, Lopez et al. [22] compared heuristics that

1. use three different ways for ordering the tasks to consider: arbitrary, in order of increasing utilization, and in order of decreasing utilization; and
2. also use three different heuristics for ordering the processors to consider: "first fit" (assign a task to the first processor upon which it fits), "worst fit" (assign a task to the processor with the maximum remaining capacity), and and "best fit" (assign a task to the processor with the minimum remaining capacity that exceeds the task's utilization).

Extension to period-elastic tasks. Any of the partitioning heuristics can be adapted for period-elastic tasks in a manner that is very similar in structure to the manner in which global EDF, PriD, and global RM were adapted for elastic tasks. That is, given an instance of periodic-elastic tasks

$$\Gamma = \left\{ \tau_i = (U_i^{(\max)}, U_i^{(\min)}, E_i) \right\}_{i=1}^{n}$$

to be scheduled upon m unit-speed processors, we iterate through possible values of λ between 0 and Φ, seeking the smallest value such that the Liu & Layland task system with utilizations

$$U_i \leftarrow \left\{ \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)} \right) \right\}_{i=1}^{n}$$

is deemed schedulable by upon m unit-speed processors by the partitioning heuristic. (The pseudo-code for doing so is again very similar to the pseudo-code in Algorithm 2, and hence omitted.)

Although the partitioning heuristics are nearly identical, we differentiate between partitioning tasks with fixed-job priority vs tasks with fixed-task priority: the task acceptance criteria onto a processor are very different for fixed-job priority and fixed-task priority algorithms. For fixed-job priority algorithms, it follows from the optimality property of preemptive uniprocessor EDF that each processor may use up to its full capacity. Hence in order to determine whether a task may be assigned to a processor under partitioned fixed-job priority scheduling the procedure Task_Compress(Γ,U_d) of Buttazzo et al. [6] (reproduced here as Algorithm 1) can be applied to each processor with $U_d \leftarrow 1.0$. However for fixed-*task* priority scheduling each processor may suffer a schedulability loss that could be close to 30% of its utilization.

Uniprocessor RM scheduling is known to be an optimal fixed-task priority algorithm (hence our decision to use it for scheduling the individual processors), but utilization-based schedulability test can be overly pessimistic [21]: using *response-time analysis* [19, 1] provides a better admission criterion. In response time analysis, rather than using a utilization-based schedulability test, a task being partitioned computes the *interference* the task will receive, the amount of time each of its jobs will be preempted by higher-priority tasks already assigned to a processor. If the task's required computation time plus the interference it suffers from higher-priority tasks already on a processor is less than or equal to its relative deadline, then the task is able to be partitioned to the processor.

5 Simulation Experiments

We have performed a simulation-based comparison of the various algorithms presented in Sections 3 and 4 for the multiprocessor scheduling of sequential period-elastic tasks; we report on the findings of this comparison below. We describe the setup for these simulation experiments in Section 5.1 and present our findings in Section 5.2; based upon these findings, we draw some high-level conclusions in Section 5.3.

5.1 Experimental Setup

We randomly generate sets of sequential period-elastic tasks and attempt to schedule them upon a given number of processors m using the different scheduling algorithms – fluid, global EDF, PriD, global RM, and partitioned scheduling of both fixed-job (RM) and fixed-task (RM) priorities – described in Sections 3 and 4 above. Specifically,

- We separately consider multiprocessor platforms containing $m = 4, 8$, and 16 identical processors.
- For each of these values for m, we consider task sets with $n = 2 \times m$, $n = 4 \times m$, and $n = 8 \times m$ tasks.
- For each combination of values of m and n, we also vary the maximum utilization value any individual task is allowed to be assigned, denoted α. This value can directly impact schedulability of a task set, particularly when using the global EDF, PriD, and global RM algorithms. We study values of of $\alpha = \{0.6, 0.8, 1.0\}$.
- For each selected combination of values of m and n, we generate task sets in which the maximum utilizations of the tasks (i.e., their $U_i^{(\max)}$ parameters) sum to $1.1 \times m \times \alpha, 1.5 \times m \times \alpha$, and $1.9 \times m \times \alpha$.

Hence a total of $3 \times 3 \times 3 \times 3 = \mathbf{81}$ different combinations of values of m, n, α, and $\left(\sum_i U_i^{(\max)}\right)$ are considered. For each such combination, we generate 500 task sets in the following manner. We generate the individual $U^{(\max)}$ values

using the *Randfixedsum* algorithm [13] to provide an unbiased distribution of maximum utilizations. The corresponding individual task minimum utilization values $U_i^{(\min)}$ are uniformly generated over the range $(0, U_i^{(\max)})$. In the case that a task set's $U_i^{(\min)}$ values sum to more than m (i.e. the task set is not schedulable under fluid scheduling, or therefore, any other scheduling algorithm), we repeatedly generated new $U_i^{(\min)}$ values for each task until their sum is sufficiently low. Tasks' elastic coefficients is chosen uniformly randomly over the range [1, 5]. For all algorithms a "granularity" of $\epsilon = \frac{\Phi}{1{,}000}$ was used.

We attempt to schedule each task set generated as described above using the various algorithms discussed in Sections 3 and 4: fluid, global EDF, PriD, global RM, partitioned fixed-job priority (FJP) with uniprocessor EDF, and partitioned fixed-task priority (FTP) with uniprocessor RM. For partitioned algorithms, we first sort the tasks. For FJP we sort in order of decreasing utilization (their $U_i(\lambda)$ parameters) whereas for JTP we sort by increasing period, in order to assign the highest priority tasks to schedulers first and make the response-time analysis simpler. In both cases we then attempt to assign them to the available processors using the the "first-fit," "worst-fit," and "best-fit" heuristics. We return the first λ value that deems the task set schedulable by any of these heuristics. We note that the ability to partition tasks onto processors in an efficient manner is an advantage of partitioned over global scheduling algorithms, since it is infeasible to carry out full simulation of global EDF, PriD, or global RM.

5.2 Observations

In our experiments, we noted (i) the fraction of task-sets that were determined to be schedulable by each of our considered algorithms; and (ii) for those task-sets that were deemed schedulable by all the algorithms, the minimum λ needed to achieve schedulability by each algorithm. Our results are presented in graphical form in Figures 2–19. In these graphs we show results of both the average minimum normalized λ value ($\frac{\lambda}{\Phi}$–this gives a value on the interval [0, 1] and is needed to compare λ values across task sets) needed to achieve schedulability for a given scheduling algorithm, and the percentage of the 500 task sets that each algorithm deemed scheduleable. To ensure a consistent comparison, we only compare lambda values for task sets deemed schedulable by all scheduling algorithms.

As mentioned in Subsection 5.1, there are 81 combinations of m, n, α, and $\left(\sum_i U_i^{(\max)}\right)$ considered in this simulation.

Figures 2–19 show both the λ values and percentage of schedulable task sets for all the considered scheduling algorithms (fluid, global EDF, PriD, and partitioned) for $\alpha = 0.6, 0.8$, and 1.0. Some trends can be noticed across all graphs. We note that fluid scheduling is an idealized optimal scheduling algorithm; not surprisingly, therefore, it schedules the largest percentage of task-sets and returns the smallest λ value. This is seen consistently across all results. It serves

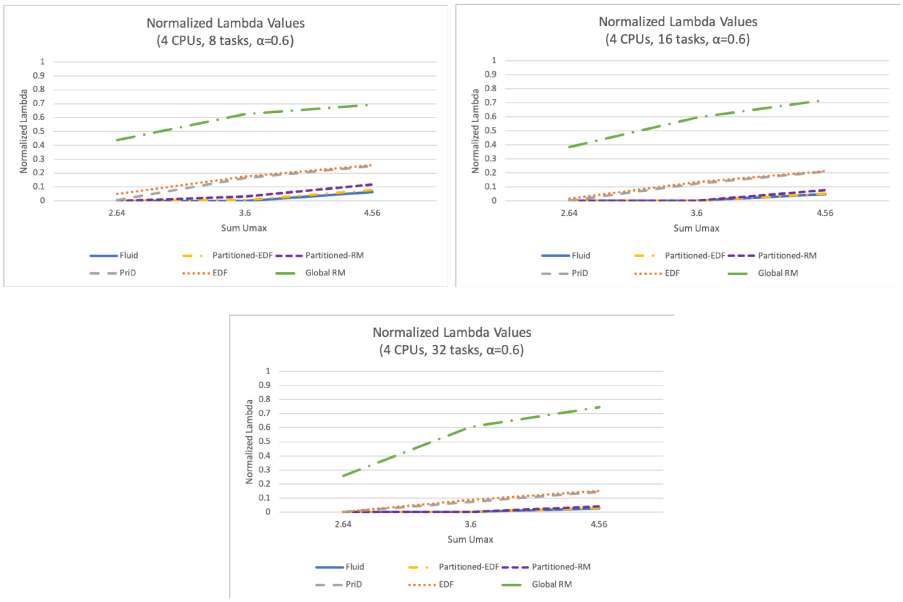

Fig. 2 Lambda Values (m=4,$\alpha = 0.6$)

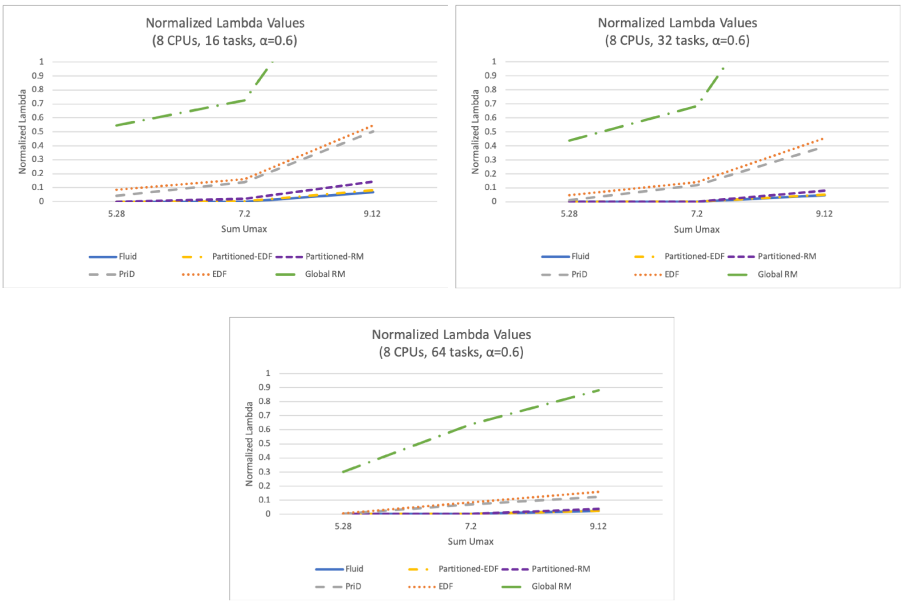

Fig. 3 Lambda Values (m=8,$\alpha = 0.6$)

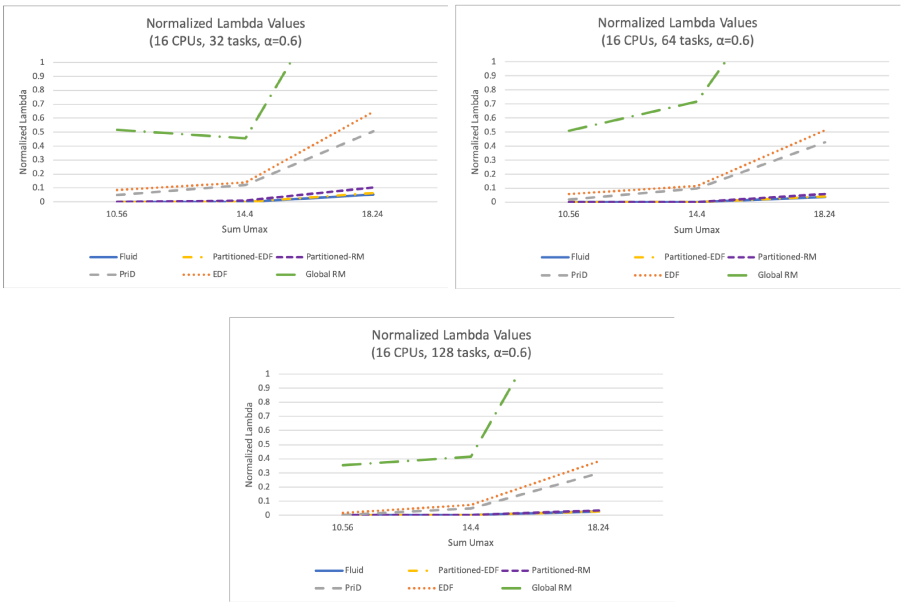

Fig. 4 Lambda Values (m=16,$\alpha = 0.6$)

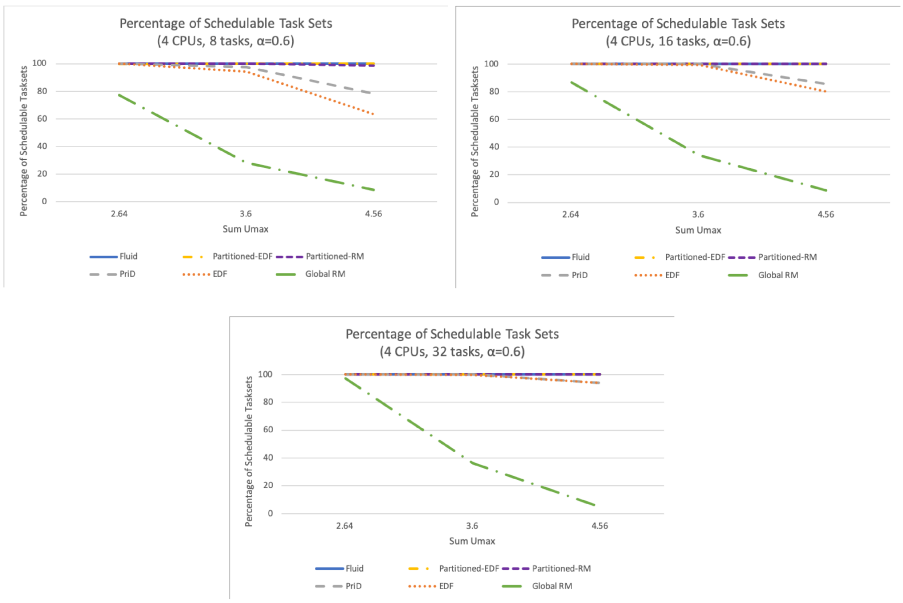

Fig. 5 Schedulability (m=4, $\alpha = 0.6$)

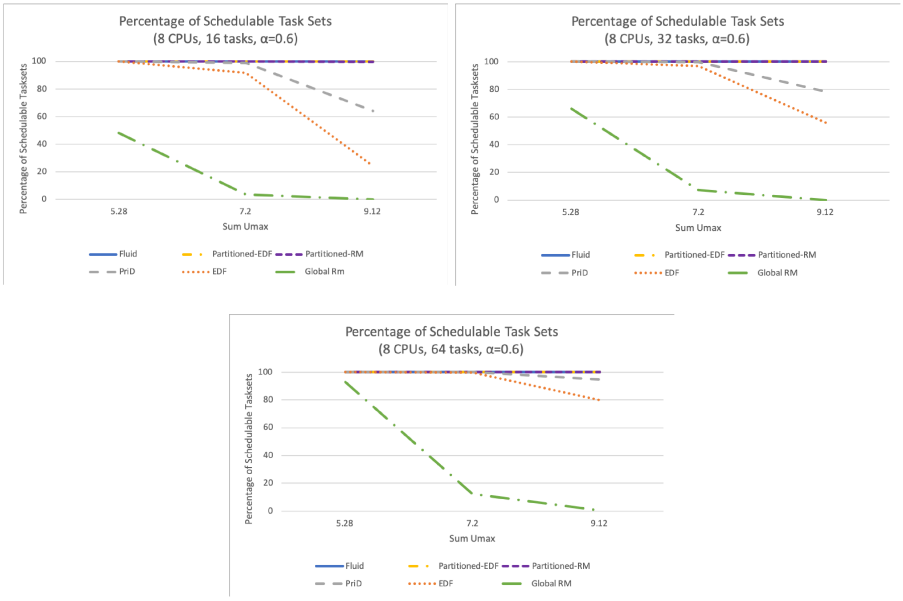

Fig. 6 Schedulability (m=8, $\alpha = 0.6$)

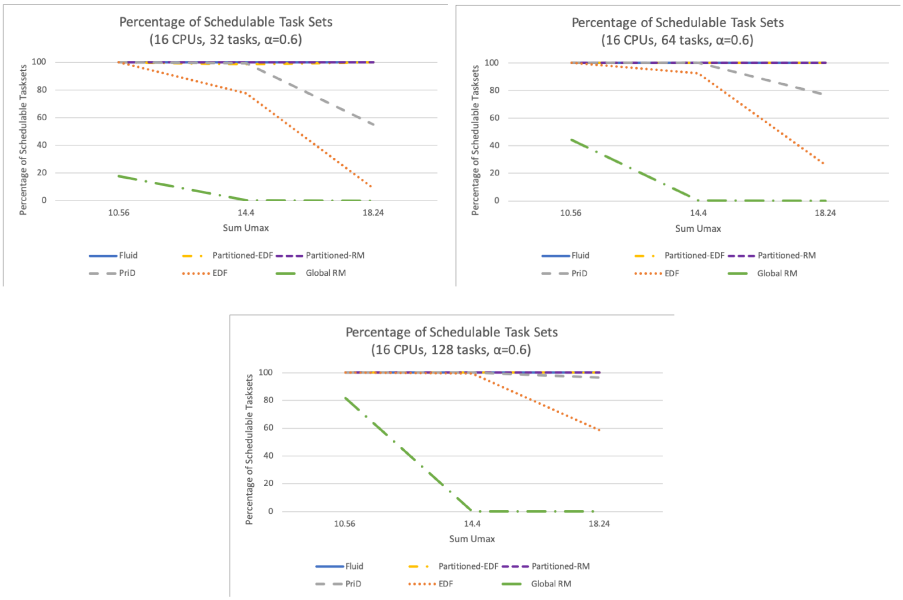

Fig. 7 Schedulability (m=16, $\alpha = 0.6$)

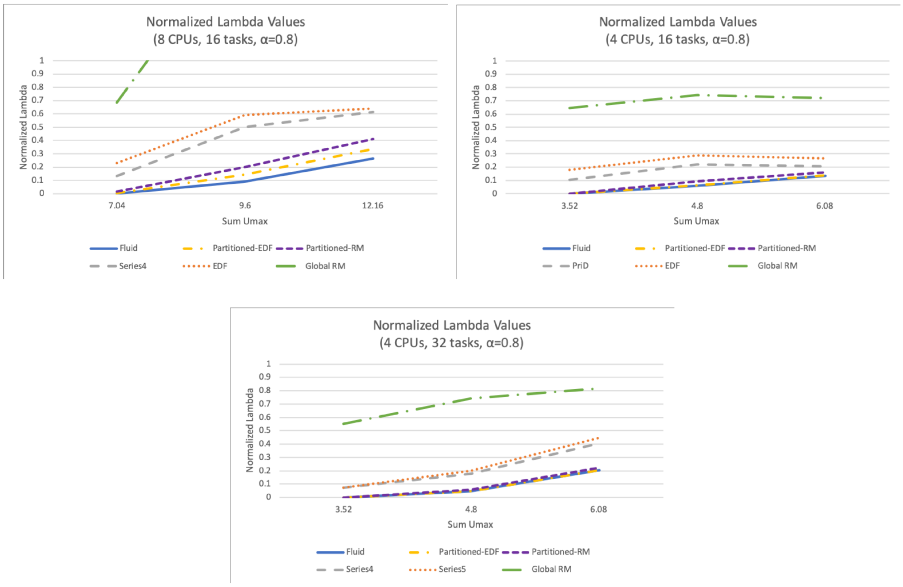

Fig. 8 Lambda Values (m=4,$\alpha = 0.8$)

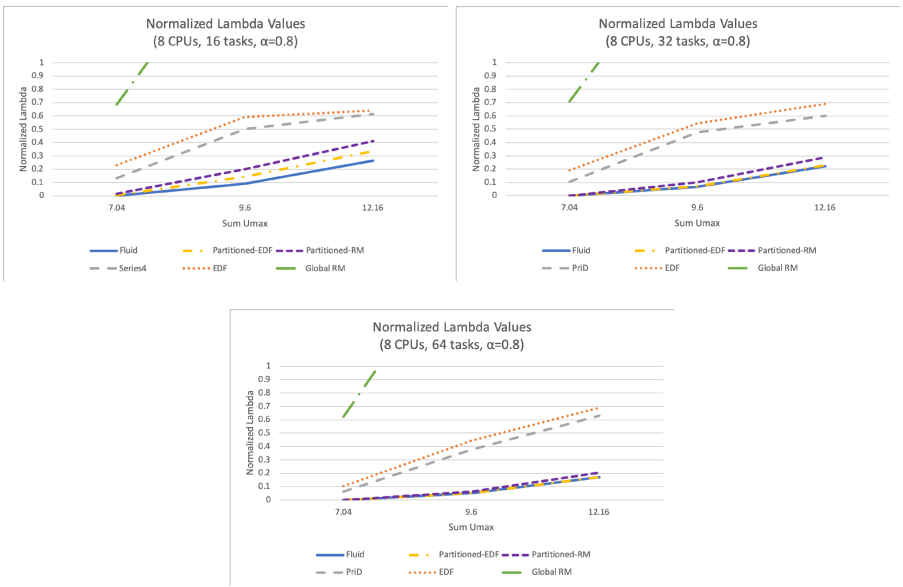

Fig. 9 Lambda Values (m=8,$\alpha = 0.8$)

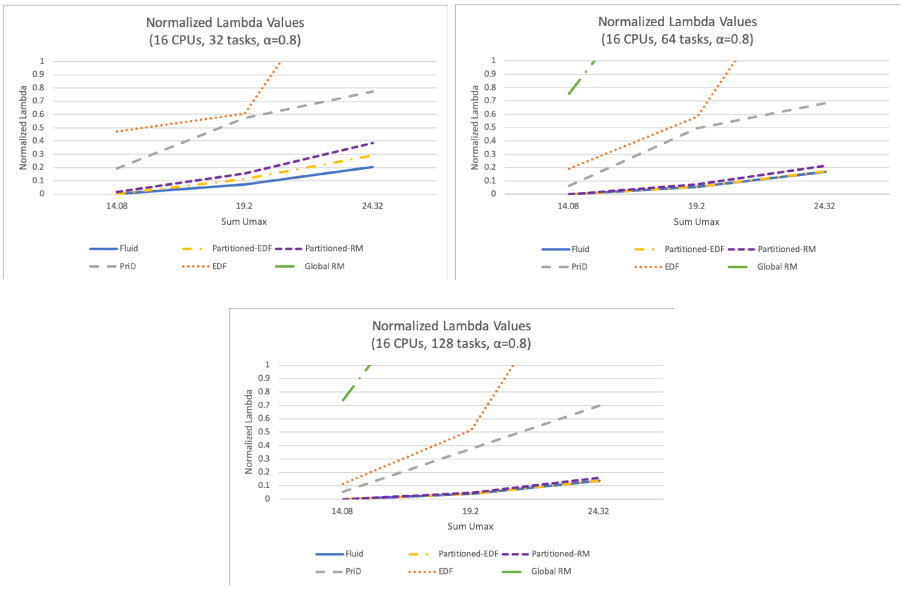

Fig. 10 Lambda Values (m=16,$\alpha = 0.8$)

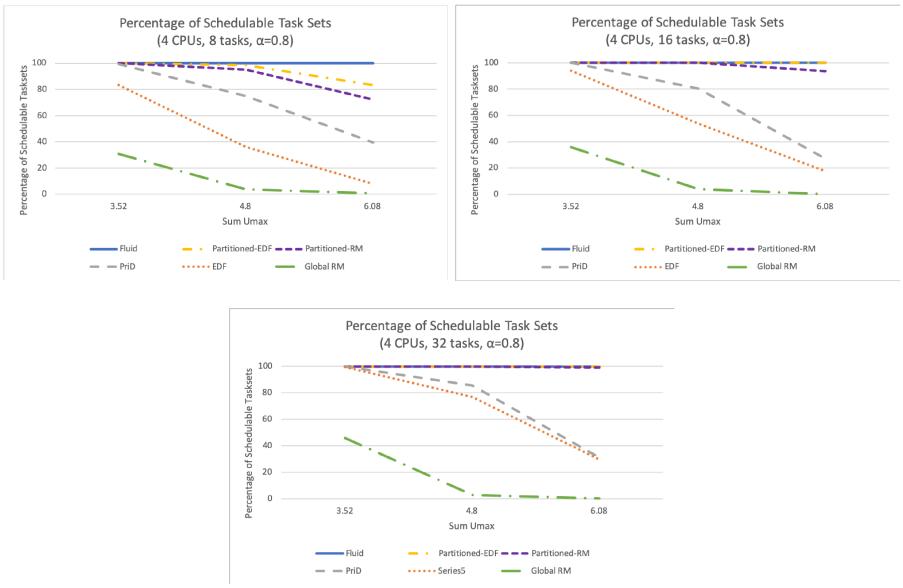

Fig. 11 Schedulability (m=4, $\alpha = 0.8$)

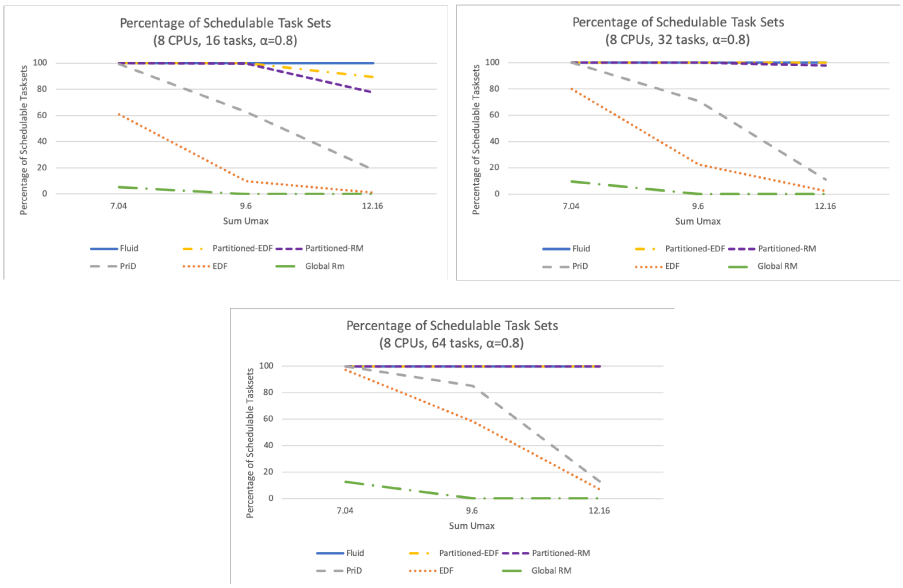

Fig. 12 Schedulability (m=8, $\alpha = 0.8$)

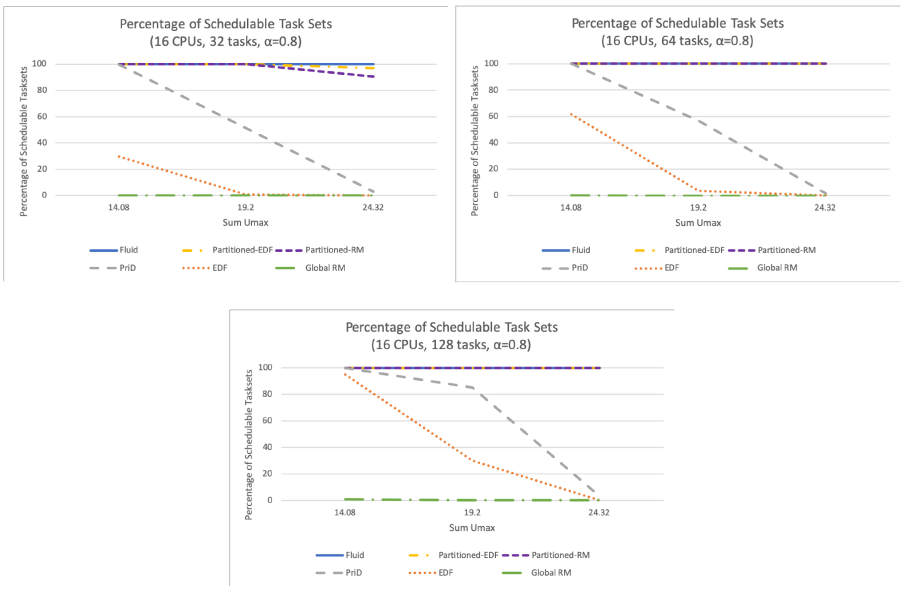

Fig. 13 Schedulability (m=16, $\alpha = 0.8$)

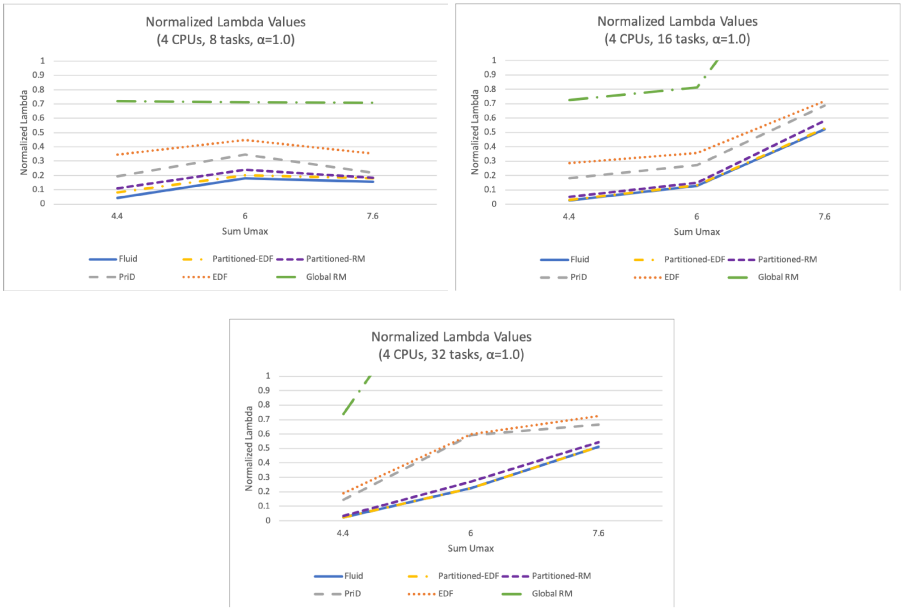

Fig. 14 Lambda Values (m=4,$\alpha = 1.0$)

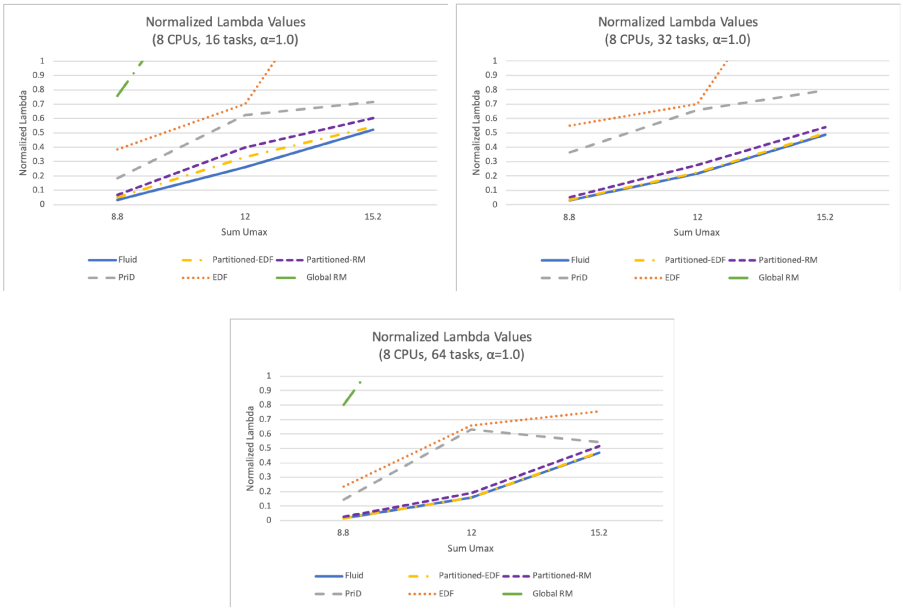

Fig. 15 Lambda Values (m=8,$\alpha = 1.0$)

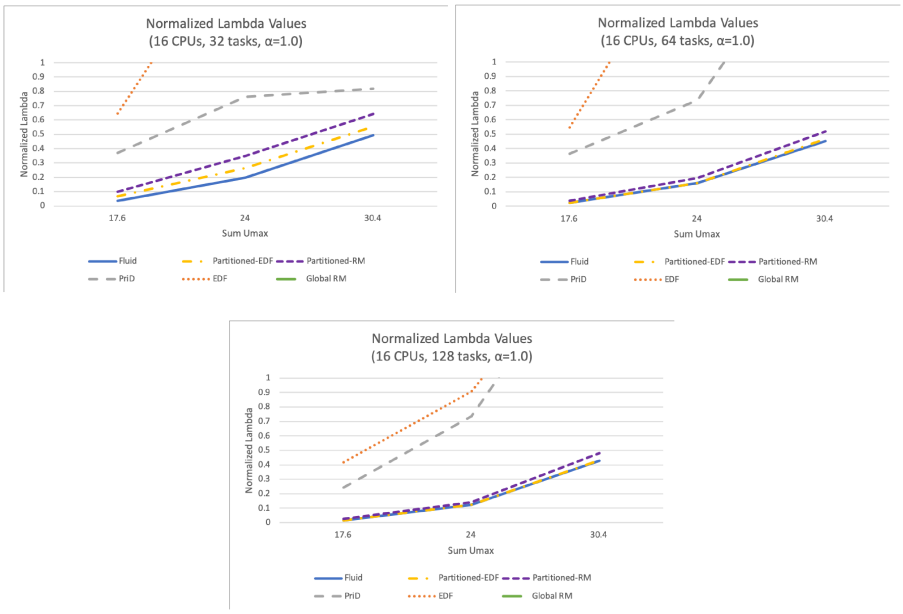

Fig. 16 Lambda Values (m=16,$\alpha = 1.0$)

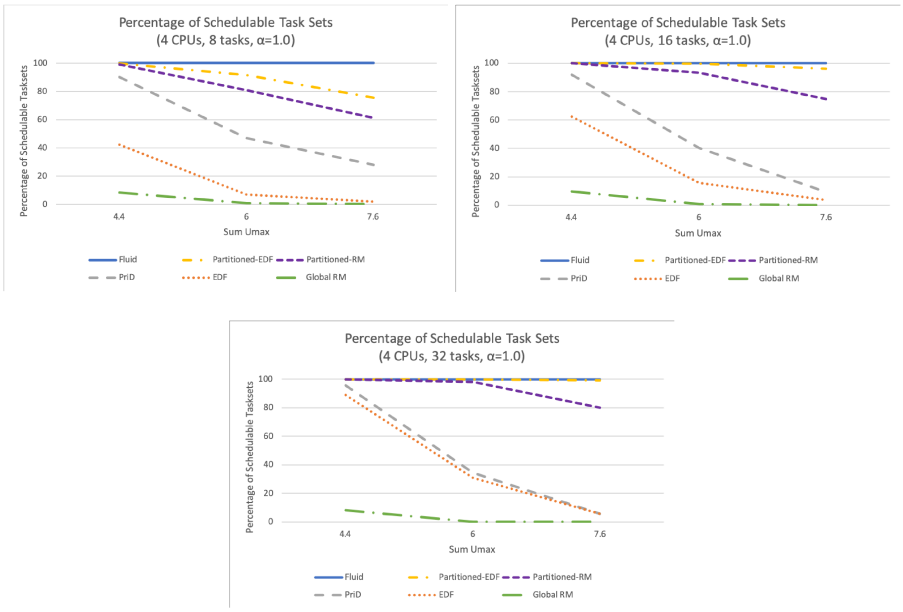

Fig. 17 Schedulability (m=4, $\alpha = 1.0$)

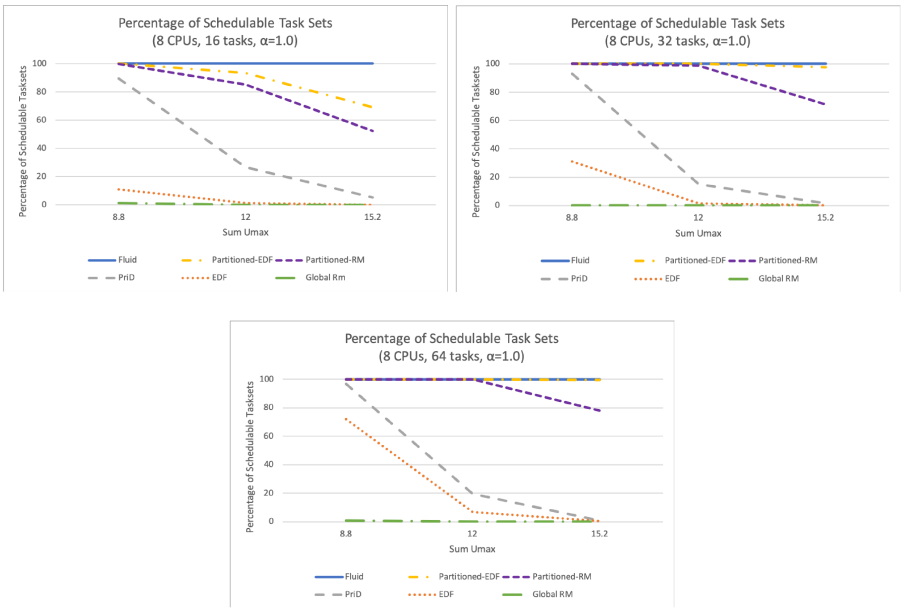

Fig. 18 Schedulability (m=8, $\alpha = 1.0$)

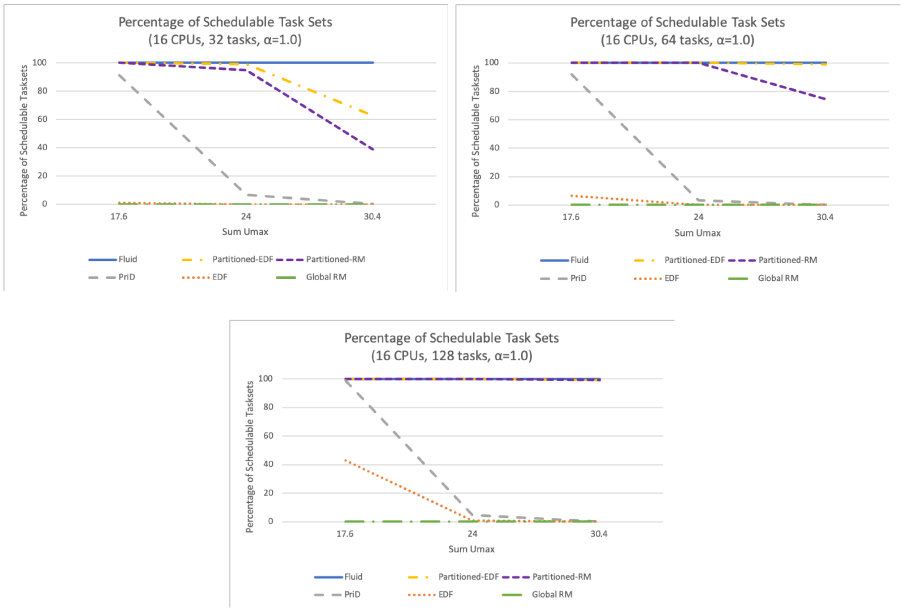

Fig. 19 Schedulability (m=16, $\alpha = 1.0$)

as an upper bound for achievable simulation results for the other scheduling algorithms. We also note that partitioned scheduling algorithms consistently dominate global algorithm PriD, global EDF, and global RM in both λ value and in percentage of schedulable task sets. This is consistent with prior observations [4] regarding global versus partitioned multiprocessor scheduling; in essence, this is likely a reflection of the fact that while global scheduling algorithms like PriD apply schedulability tests that are utilization-based and incorporate considerable pessimism since they must consider "worst-case" task-sets with the same utilization parameters as running full simulations is infeasible, partitioned schedulability tests actually attempt to perform a partition and hence do not necessarily pay the price in terms of such analysis-based pessimism.

Note that global RM scheduling always requires the highest λ value, and that the percentage of task sets deemed schedulable under global RM decreases as $\left(\sum_i U_i^{(\max)}\right)$ increases. Global EDF consistently requires the second highest λ and is the second most difficult to schedule. This is a manifestation of the Dhall effect, and our experiments revealed that this worsens as the number of processors and tasks increase: for some combinations of m, n, and $\left(\sum_i U_i^{(\max)}\right)$ that we considered, global RM, global EDF, and more rarely PriD fail to schedule a single task set out of 500. In such cases the reported λ is off the chart. The Dhall effect can be observed to worsen as α increases.

Our experiments also reveal that it becomes more difficult to schedule tasks (in terms of both λ value and schedulability percentage) for all the scheduling algorithms as $\left(\sum_i U_i^{(\max)}\right)$ increases. The same is true as the number of processors increases but the ratio of processors to tasks remains the same. On a constant number of processors, fluid and partitioned scheduling can return a lower λ value with more tasks in the task set, and a higher percentage of task sets are deemed schedulable under partitioned scheduling (while PriD and global EDF seem less affected). We believe this improvement seen to be a reduction in the Dhall effect: as more tasks are introduced into the system the largest single task is more likely to decrease. Naturally fluid scheduling always deems 100% of tasks to be schedulable.

5.3 Recommendation

Based on our observations in the previous subsection and the graphs in Figures 2–19, we recommend that in the absence of specific knowledge regarding task characteristics that may advocate in favor of global RM, global EDF, or PriD, partitioned scheduling be used for the scheduling of sequential period-elastic tasks on uniform multiprocessor systems, particularly in systems with a large number of tasks. Among the realistic (i.e., excluding the idealized fluid scheduling algorithm) scheduling algorithms considered in this paper, partitioned scheduling 1) consistently returns the lowest value of λ (i.e., compresses tasks the least); and 2) schedules the highest percentage of task sets. While

running uniprocessor EDF on each partitioned processor yields the best results in our simulation, we note that uniprocessor RM has been noted to have several implementation advantages in practice; as such, it may be worth the slight scheduling inefficiency.

6 Related Work

Buttazzo et al. first introduced the elastic task model for sequential tasks on a preemptive uniprocessor [6]. The sequential model was later extended to include resource sharing [7] and unknown computational loads [8]. Chantem et al. proved Buttazzo's initial scheduling algorithm to be equivalent to solving a quadratic optimization problem and introduced a period-based optimization problem scheme for period selection [10]; they further extended the model to include constrained deadlines [10]. Our prior work introduced elastic scheduling of tasks with internal parallelism under the federated scheduling paradigm [26] and the concept of computational elasticity [25]. Recent work by Gill et al., has applied elastic scheduling to mixed-criticality systems [14]. We leave the sequential multi-core scheduling extensions to all of these problems as future work.

7 Conclusion

In this paper we have introduced elastic scheduling for sequential tasks on multiprocessor systems. We have introduced algorithms for scheduling such tasks under both global (in a variety of manners) and partitioned scheduling paradigms. We ran an extensive simulation to compare these methods and conclude that partitioned scheduling should be used if possible.

References

1. Audsley, N., Burns, A., Richardson, M., Tindell, K., Wellings, A.: Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal **8**(5), 285–292 (1993)
2. Baruah, S.: Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. IEEE Transactions on Computers **53**(6) (2004)
3. Baruah, S., Cohen, N., Plaxton, G., Varvel, D.: Proportionate progress: A notion of fairness in resource allocation. Algorithmica **15**(6), 600–625 (1996)
4. Bastoni, A., Brandenburg, B., Anderson, J.: An empirical comparison of global, partitioned, and clustered multiprocessor real-time schedulers. In: Proceedings of the Real-Time Systems Symposium, pp. 14–24. IEEE Computer Society Press, San Diego, CA (2010)
5. Bertogna, M., Cirinei, M., Lipari, G.: New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In: Proceedings of the 9th International Conference on Principles of Distributed Systems. IEEE Computer Society Press, Pisa, Italy (2005)
6. Buttazzo, G.C., Lipari, G., Abeni, L.: Elastic task model for adaptive rate control. In: IEEE Real-Time Systems Symposium (RTSS) (1998)

7. Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L.: Elastic scheduling for flexible workload management. IEEE Trans. Comput. **51**(3), 289–302 (2002). DOI 10.1109/12.990127. URL http://dx.doi.org/10.1109/12.990127
8. Caccamo, M., Buttazzo, G., Sha, L.: Elastic feedback control. In: Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000, pp. 121–128 (2000). DOI 10.1109/EMRTS.2000.853999
9. Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., Baruah, S.: A categorization of real-time multiprocessor scheduling problems and algorithms. In: J.Y.T. Leung (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press LLC (2003)
10. Chantem, T., Hu, X., Lemmon, M.: Generalized elastic scheduling for real-time tasks. IEEE Transactions on Computers **58**(4), 480–495 (2009). DOI 10.1109/TC.2008.175
11. Dhall, S.: Scheduling periodic time-critical jobs on single processor and multiprocessor systems. Ph.D. thesis, Department of Computer Science, The University of Illinois at Urbana-Champaign (1977)
12. Dhall, S.K., Liu, C.L.: On a real-time scheduling problem. Operations Research **26**, 127–140 (1978)
13. Emberson, P., Stafford, R., Davis, R.: Techniques for the synthesis of multiprocessor tasksets. WATERS'10 (2010)
14. Gill, C., Orr, J., Harris, S.: Supporting graceful degradation through elasticity in mixed-criticality federated scheduling. In: Proceedings of the 6th International Workshop on Mixed Criticality Systems (WMC) (2018)
15. Goossens, J., Funk, S., Baruah, S.: Priority-driven scheduling of periodic task systems on multiprocessors. Real Time Systems **25**(2–3), 187–205 (2003)
16. Horn, W.: Some simple scheduling algorithms. Naval Research Logistics Quarterly **21**, 177–185 (1974)
17. Johnson, D.: Fast algorithms for bin packing. Journal of Computer and Systems Science **8**(3), 272–314 (1974)
18. Johnson, D.S.: Near-optimal bin packing algorithms. Ph.D. thesis, Department of Mathematics, Massachusetts Institute of Technology (1973)
19. Joseph, M., Pandya, P.: Finding response times in a real-time system. The Computer Journal **29**(5), 390–395 (1986)
20. Lee, J., Phan, K.M., Gu, X., Lee, J., Easwaran, A., Shin, I., Lee, I.: MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In: Real-Time Systems Symposium (RTSS), 2014 IEEE, pp. 41–52 (2014)
21. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM **20**(1), 46–61 (1973)
22. Lopez, J.M., Diaz, J.L., Garcia, D.F.: Utilization bounds for EDF scheduling on real-time multiprocessor systems. Real-Time Systems: The International Journal of Time-Critical Computing **28**(1), 39–68 (2004)
23. McNaughton, R.: Scheduling with deadlines and loss functions. Management Science **6**, 1–12 (1959)
24. Orr, J., Baruah, S.: Multiprocessor scheduling of elastic tasks. In: Proceedings of the 27th International Conference on Real-Time Networks and Systems, RTNS '19, p. 133–142. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3356401.3356403. URL https://doi.org/10.1145/3356401.3356403
25. Orr, J., Gill, C., Agrawal, K., Baruah, S., Cianfarani, C., Ang, P., Wong, C.: Elasticity of workloads and periods of parallel real-time tasks. In: Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS '18, pp. 61–71. ACM, New York, NY, USA (2018). DOI 10.1145/3273905.3273915. URL http://doi.acm.org/10.1145/3273905.3273915
26. Orr, J., Gill, C., Agrawal, K., Li, J., Baruah, S.: Elastic scheduling for parallel real-time systems. Leibniz Transactions on Embedded Systems **6**(1), 05–1–05:14 (2019). DOI 10.4230/LITES-v006-i001-a005. URL https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v006-i001-a005