Scheduling Elastic Applications in Compositional Real-Time Systems

Shaik Mohammed Salman*†, Saad Mubeen†, Filip Marković†, Alessandro V. Papadopoulos†, and Thomas Nolte†
*ABB AB, Västerås, Sweden, shaik.salman@se.abb.com
†Mälardalen University, Västerås, Sweden, {name.surname}@mdh.se

2021 IEEE 26th International Conference on Emerging Technologies and Factory Automation (ETFA) | 978-1-7281-2989-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ETFA45728.2021.9613375

***Abstract*—Many real-time applications have functional behaviour that requires variability in timing properties at run-time. The elastic task model provides a convenient mechanism to specify and encapsulate such variability and enables the modification of an application's periods during run-time to keep the application schedulable. Additionally, reservation-based scheduling techniques were proposed for the same purpose of taming unpredictability of timing variations, but with a different solution, i.e., by providing the spatial and temporal isolation for executing independent applications on the same hardware.**

In this paper, we combine the two approaches by proposing a two-level adaptive scheduling framework which is based on the elastic task model and the compositional framework based on the periodic resource model. The proposed framework minimises the number of requests for bandwidth adaption at the reservation (system) level and primarily enables schedulability by accounting for the application's elasticity by adjusting the periods. The motivation for this design choice is to rather localise the effect of the modifications within the application, without necessarily affecting all the applications at the system level compared to the changes made at the application level. The evaluation results show that the local application changes may often be enough to solve the problem of variability, significantly reducing the number of bandwidth adjustments, and therefore reducing the potential negative impact on all the applications of a system.

***Index Terms*—real-time, elastic task model, reservations, hierarchical scheduling.**

I. Introduction

Many industrial real-time systems such as robot controllers have real-time requirements that are flexible to variability in execution times of the tasks, and the frequency of the task invocations[1]. For instance, Simon et al. [2] provided a feedback-based scheduling algorithm for computed torque control of an industrial arm, where the frequency of the dynamic compensation tasks, such as that of gravity and Coriolis compensation, was regularly adapted to meet both the control objectives as well as the schedulability of the system tasks. Buttazzo et al. [3] proposed the elastic task model to capture such dynamic behaviour of the tasks where the schedulability of the system is managed by adapting the frequencies of the tasks. Recently, modern system architectures based on fog and cloud computing concepts have been proposed to improve the performance of robots [4], [5]. A key idea behind such architectures is to exploit the improved computation capabilities offered by the processors by executing independent applications on the same hardware, e.g., running multiple instances of the robot controller software to control different robots on the same processor. Since the execution of independent applications requires temporal and spatial isolation, the concepts of virtualization and hierarchical scheduling, based on reservations, provide the necessary infrastructure to enable such a requirement. While there exist many solutions to schedule adaptive tasks of independent applications in a hierarchical scheduling approach [6], [7], [8], [9], most of them focus on modifying the reservation parameters according to the application demands, rather than adapting the application behaviour to a fixed reservation bandwidth. A disadvantage of modifying the reservation parameters according to the application demands is that the performance of another independent application co-executing on the same processor may be unnecessarily affected. By making the applications adapt to a fixed reservation bandwidth, we can limit the impact on other applications running on the same processor. However, there may be instances where the local adaptation of the application can fail, e.g., due to insufficient bandwidth, compelling a bandwidth modification. Therefore, to meet the aforementioned requirements, we propose a two-layered adaptive approach to schedule applications specified according to the elastic task model within the compositional real-time framework based on the periodic resource model [10]. Concretely, we address the following questions:

- Q1 Given an application with elastic tasks, what is a feasible bandwidth reservation according to the periodic resource model?
- Q2 Given a fixed bandwidth reservation according to the periodic resource model, how can the elastic application adapt its frequencies to remain schedulable?
- Q3 Given an elastic application, can a schedulable reservation be found if the application requests for a modified bandwidth reservation?

We address Q1 by assuming that an application specifies initial desired frequencies for each of its tasks and then uses those values to identify a feasible bandwidth. We address Q2 by modifying the application task frequencies whenever there is an overload or an application's task requests a different frequency such that the application satisfies the schedulability conditions under the periodic resource model. We address Q3

978-1-7281-2989-1/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

by checking if the system-level schedulability is satisfied for the modified bandwidth reservation.

We provide the system model and discuss the necessary background on elastic tasks and the periodic resource model in Section II, followed by the proposed solution in Section III. We present the evaluation of our approach in Section IV and the related work in Section V. Finally, Section VI concludes the paper.

II. Proposed System Model

This section presents the system model of the two-level compositional scheduling framework for uniprocessor systems. At the application level, we consider a real-time application specified according to the elastic task model with implicit deadlines(See Section. II-A). We assume that each application provides a local scheduler, based on either fixed-priority preemptive scheduling implementing Rate Monotonic (RM) priority assignment or dynamic-priority preemptive scheduling implementing the Earliest Deadline First (EDF) policy. At the system level, we assume that the CPU resource is made available to each application according to the Periodic Resource Model (PRM) [10](See Section. II-B).

A. The Basic Elastic Task Model

Buttazzo et al. [3], [11] proposed the elastic task model for applications whose tasks can have an adaptive temporal behaviour to address overload situations as well as requests for starting new tasks or modifying the task periods. Under this model, whenever there is an overload or a task requests a new period, the utilization of the remaining tasks is adjusted to keep the overall application's utilization under an upper-bound value for a given scheduling algorithm. For example, if the application tasks are scheduled according to EDF, then the application utilization bound is set to 1 and the utilization values of the individual tasks are adjusted accordingly. While the elastic task model can be applied to applications that have computation time variability as well as period variability, in this paper, we will only consider applications with period variability.

Formally, we define an elastic application $\mathbf{A}$ as a set of n elastic tasks $\tau_i = \{C_i, T_i^{min}, T_i^{max}, T_i^d, e_i\}$, where C_i is the Worst-Case Execution Time (WCET) of the task τ_i. T_i^{min} and T_i^{max} specify the minimum and the maximum inter-arrival time between consecutive jobs of τ_i. T_i^d represents the desired period of τ_i. The elastic co-efficient e_i represents the flexibility of τ_i to change. For instance, e_i can be defined to be in the range $[0, 1]$, where $e_i = 0$ indicates that the $T_i^{min} = T_i^d = T_i^{max}$ and that this task's period cannot be modified, and $e_i = 1$ indicates that the task's period can be modified to take up values upto its maximum period. We use T_i (without any postscript) to indicate the current inter-arrival time of τ_i. An example of an elastic taskset is shown in the Table I. while the task τ_1 can execute at its maximum period, the task τ_5 can only execute at its desired period.

The utilization of a task τ_i is given by $U_i = \frac{C_i}{T_i}$. Further, the minimum and maximum utilization of each task is represented by $U_i^{min} = \frac{C_i}{T_i^{max}}$ and $U_i^{max} = \frac{C_i}{T_i^{min}}$ respectively. At run-time, the utilization of a task is kept as close as possible to a desired utilization $U_i^d = \frac{C_i}{T_i^d}$. The desired application utilization is given by $U^d = \sum_{i=1}^{n} U_i^d$. Similarly, the minimum and maximum application utilization is given by $U^{min} = \sum_{i=1}^{n} U_i^{min}$ and $U^{max} = \sum_{i=1}^{n} U_i^{max}$. If a task requests for a change in its current period, its desired utilization U_i^d is updated. An elastic application is said to be schedulable if $U^d \leq U^{ub}$, where U^{ub} is the utilization upper-bound for a given scheduling algorithm. It is assumed that the deadline is elastic-implicit. i.e., the relative deadline of each job of an elastic task is equal to its current period at runtime.

TABLE I
An elastic Task set

Task ID	WCET	T_i^{min}	T_i^d	T_i^{max}	e_i
τ_1	4	40	120	240	1
τ_2	7	40	80	360	0.75
τ_3	10	240	240	480	0.5
τ_4	9	200	240	600	0.25
τ_5	8	40	40	40	0

Elastic Compression Algorithm: At runtime, if a task exceeds its execution time or requests for a change in its period, the application is made schedulable by modifying the periods of the application's tasks to accommodate the new values and ensuring that the total utilization of the application's tasks is below the schedulable utilization bound. This is done according to the original task compression algorithm proposed by Buttazzo et al. [3] and is reproduced here as Algorithm 1. It takes as input the elastic application and the maximum schedulable utilization bound. It computes the minimum utilization of the taskset and compares it to the schedulable utilization bound. If the minimum utilization of the elastic application exceeds the schedulable utilization bound, it immediately exits and returns a failure. Otherwise, it iterates through each task of the application and depending on the elastic coefficients and the current period T_i of each task, it separates the tasks into two disjoint sets $\mathbf{A_f}$ and $\mathbf{A_v}$. The set $\mathbf{A_f}$ contains all the tasks whose utilization values are fixed, i.e., the tasks with elastic coefficients set to 0 and tasks executing with their maximum period values. The set $\mathbf{A_v}$ contains the remaining tasks whose utilization can be varied. Further, U_f represents the sum of the utilization of the tasks in $\mathbf{A_f}$, while E_v represents the sum of the elastic coefficients of tasks in $\mathbf{A_v}$. For each task in $\mathbf{A_v}$, its utilization value is scaled according to the ratio of the elastic coefficient and the sum of all the coefficients in $\mathbf{A_v}$ (Line 21). A new task period T_i is then assigned to the task. If the new task period exceeds the T_i^{max} value, it is set equal to T_i^{max}. If this happens, the task τ_i is added to the set $\mathbf{A_f}$ and the process is repeated. The algorithm returns a feasible taskset if either all the tasks have reached their maximum period or if all the tasks' periods have been updated such that they are schedulable. We use this algorithm

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

as a part of our proposed solution (Section III).

Algorithm 1 Task_Compress()

```
1: function TASK_COMPRESS(A, U^ub)
2:     U^d = Σ_{i=1}^{n} C_i / T_i^d
3:     U^min = Σ_{i=1}^{n} C_i / T_i^max
4:     if U^ub < U^min then
5:         return Infeasible
6:     end if
7:     OK = 0
8:     while OK= 0 do
9:         U_f = 0
10:        E_v = 0
11:        for each τ_i in A do
12:            if e_i == 0 or T_i == T_i^max then
13:                U_f = U_f + U_i
14:            else
15:                E_v = E_v + e_i
16:            end if
17:        end for
18:        OK = 1
19:        for each τ_i in A_v do
20:            if E_i > 0 and T_i < T_i^max then
21:                U_i = U_i^d − (U^d − U^ub + U_f) * e_i / E_v
22:                T_i = C_i / U_i
23:                if T_i > T_i^max then
24:                    T_i == T_i^max
25:                    OK = 0
26:                end if
27:            end if
28:        end for
29:    end while
30:    return Feasible
31: end function
```

B. Periodic Resource Model

Lee et al. [10] proposed the compositional scheduling framework based on the periodic resource model to support the development of component-based hierarchical software systems. In this framework, the computational resource is described as a periodic resource model $\Gamma(\Theta, \Pi)$, where Θ is the periodic resource allocation time and Π is the resource period. Essentially, the periodic resource Γ provides an application $\mathbf{A}$ with Θ time units of CPU time every Π time units. The worst case resource supply of the periodic resource model is shown in Fig. 1. The utilization of the resource supply is defined as $U_\Gamma = \frac{\Theta}{\Pi}$.

Generating the Resource Supply Parameters: we use the method described in Section. 6 of [10] to generate the resource supply parameters Θ and Π, such that an application $\mathbf{A}$, modeled as a set of n periodic tasks with implicit deadlines, where each task modeled as $\tau_i = \{C_i, T_i\}$ is schedulable (under EDF or RM scheduling policy). According to the compositional framework, Given the smallest period of the application, T^{min}, the application is schedulable under RM scheduling policy if the resource supply utilization satisfies Eq. (1) (Eq. 23 in [10]).

$$U_{\Gamma,RM}(k) = \frac{U_{\mathbf{A}}}{\log\left(\frac{2k+2(1-U_{\mathbf{A}})}{k+2(1-U_{\mathbf{A}}}\right)}, \tag{1}$$

where,

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta < T^{min}\} \tag{2}$$

Similarly, the application is schedulable under EDF scheduling policy if the resource supply utilization satisfies Eq. (3) (Eq. 21 in [10]).

$$U_{\Gamma,EDF}(k) = \frac{(k+2).U_{\mathbf{A}}}{k+2(U_{\mathbf{A}})}, \tag{3}$$

where,

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta - \frac{k\Theta}{k+2} < T^{min}\} \tag{4}$$

Schedulable Utilization Bounds: Since the main goal of the our solution is to minimize the modifications of the resource supply parameters once they have been defined, we rely on the schedulable utilization bounds defined in the Section 5 of [10] to keep the application schedulable by changing the application utilization rather than changing the resource supply utilization. Accordingly, an application is schedulable under RM policy, if the application utilization $U_{\mathbf{A}}$ is less than or equal to the utilization bound $\mathbf{UB}_{RM}(n, T^{min})$ as defined in Eq. (5) (Eq. 16 in [10]).

$$\mathbf{UB}_{RM}(n, T^{min}) = U_\Gamma.n\left[\left(\frac{2k+2(1-U_\Gamma)}{k+2(1-U_\Gamma)}\right)^{\frac{1}{n}} - 1\right], \tag{5}$$

where

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta < T^{min}\}$$

Similarly, an application is schedulable under EDF policy, if the application utilization $U_{\mathbf{A}}$ is less than or equal to the utilization bound $\mathbf{UB}_{EDF}(T^{min})$ as defined in Eq. (6)(Eq. 13 in [10]).

$$\mathbf{UB}_{EDF}(T^{min}) = \frac{kU_\Gamma}{k+2(1-U_\Gamma)}, \tag{6}$$

where

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta - \frac{k\Theta}{k+2} < T^{min}\}$$

III. Proposed Solution

To schedule an elastic application in a hierarchical scheduling framework based on the periodic resource model, we propose a two-layered adaptive scheduling mechanism that first attempts to adapt the utilization of the tasks at the application level and if this adaptation fails, it attempts to reallocate available spare resource capacity at the system level. The different components of the proposed framework along

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

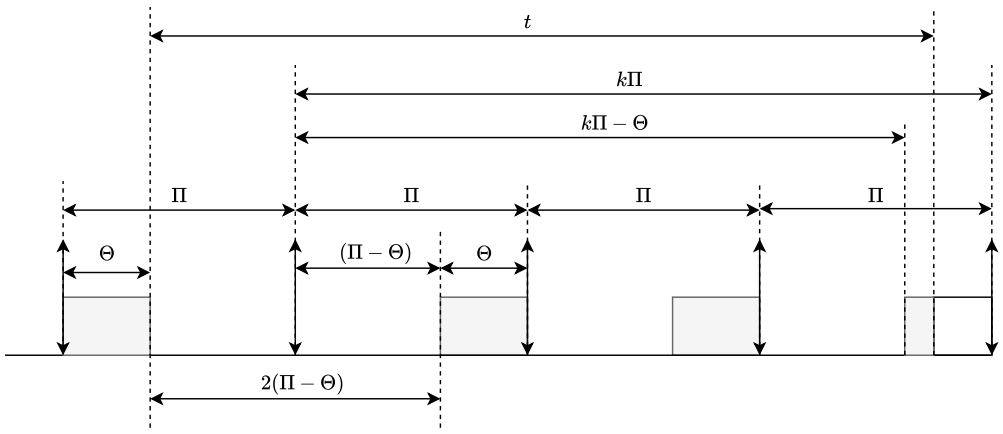

Fig. 1. Resource supply of a Periodic Resource Model.

with the data flow between them are shown in Fig. 2. At the application level, it consists of an independent application defined according to the elastic task model, an elastic manager that implements the task compression algorithm of Buttazzo et al. [3]. and a local scheduler implementing either the RM scheduling policy or the EDF scheduling policy. At the system level, the Global Compositional Scheduling Resource (GCSR) manager provides the necessary interface for communicating with the different applications and the functional support for serving requests of new bandwidth resource allocations from the individual applications. The functional behaviour of the GCSR manager is supported by the OS or the hypervisor kernel. At the application level, whenever there is an overload situation or an elastic task requests for a new period, the elastic manager will try to modify and update the periods of the rest of the tasks to keep the application schedulable using the Algorithm 1. If the resource supply is insufficient for the current demand, the elastic manager generates a new sufficient resource supply interface and requests the GCSR manager for updating the resource supply parameters. The GCSR manager will accept the request and responds successfully(i.e., assign new resource supply parameters) if the global system schedulability is preserved with the updated parameters. The elastic manager will then re-adjust the periods based on the updated resource supply parameters.

A. Initial Desired Resource Supply

In the proposed framework, we first find the suitable resource supply $\Gamma(\Theta, \Pi)$ for the application $\mathbf{A}$ considering the parameters $\tau_i(C_i, T_i)$. We choose as T_i, the desired periods for each task. For example, in Table I, The values under the column T_i^d represent the initial desired periods of the application tasks. We assume that such a taskset is feasible. Next, depending on the scheduling algorithm, we find the resource supply utilization bound necessary to keep the application tasks schedulable according to Eq. (1) and Eq. (3). While there exist fully polynomial time solutions to find approximate bandwidth allocations for the periodic resource model, e.g., [12], we use the approach proposed by Lee et al. [10] in this paper. We assume that $\Gamma(\Theta, \Pi)$ is schedulable at the system level. Note that if $\Gamma(\Theta, \Pi)$ is not schedulable at the system level, then the application will have to modify its initial desired periods or the resource supply of the other co-running applications will have to be modified. In our approach, we reject an application if the initial resource supply is not schedulable.

B. Runtime Adaptation

Under worst-case conditions, the resource supply provided according to PRM can result in a no supply interval of duration $2(\Pi - \Theta)$ (see Fig. 1). Therefore, once the application is executing, whenever a task requests for a new period T_i^{new}, we need to consider two different scenarios depending on the value of T_i^{new}. If T_i^{new} is greater than the no supply duration, we can adapt the tasks utilization at the application level without changing the resource supply parameters. If T_i^{new} is less than or equal to the no supply duration, we need to adapt the resource supply at the system level.

a) Application level Adaptation: From Eq. (5) and Eq. (6), it is easy to see that the utilization bound to keep the application tasks schedulable remains constant as long as T^{min} remains unchanged. When the requested period T_i^{new} is greater than or equal to T^{min}, it implies that the resource supply parameters do not have to be changed since the current T^{min} remains unchanged. As a consequence, and based on the sustainability property of the utilization tests [13], the elastic manager can find a schedulable period reassignment by ensuring that the modified application utilization $U_{\mathbf{A}}^{new}$ remains below the schedulable utilization bound as shown in Eq. (7) or Eq. (8). Note that the periodic supply resource utilization U_Γ remains constant under application level adaptation.

$$U_{\mathbf{A}}^{new} \leq \mathbf{UB}_{EDF}(T^{min}) \tag{7}$$

$$U_{\mathbf{A}}^{new} \leq \mathbf{UB}_{RM}(n, T^{min}) \tag{8}$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

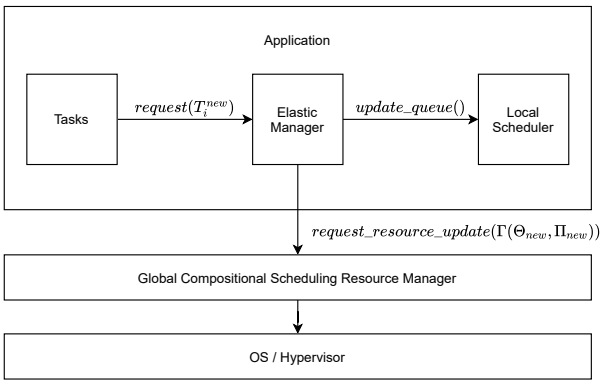

Fig. 2. Data Flow Between The Adaptive Scheduling Framework Components.

b) System Level Adaptation : At runtime, if a task requests for a new period T_i^{new} that is less than T^{min}, then it is not guaranteed that the existing resource supply Γ can provide sufficient CPU time for the application tasks to remain schedulable. This is because the schedulable utilization bound (for both EDF and RM scheduling policy) is a function of the minimum period of the taskset and since we are now reducing the minimum period, it may so happen that the T_i^{new} will have its arrival and deadline in the no supply interval of duration $2(\Pi - \Theta)$ in the worst case (see Fig. 1). Therefore, whenever a task makes a request of T_i^{new} less than T^{min}, the elastic manager will generate new resource supply parameters and request the GCSR manager to update the resource supply according to the new parameters so that the application remains schedulable. If the GCSR manager rejects the request, then the elastic manager will not be able to satisfy the application request and it is then up to the application to decide how it needs to proceed. If the resource supply parameters are updated, the next request for a period change will be handled based on the updated resource supply parameters. While it is possible to apply the elastic task compression algorithm at the system level to modify the resource supply utilization to accommodate the requests from the different applications, it requires modifications of the resource supply of the co-running applications which in turn can trigger application level modifications. To avoid this, we require some spare capacity to be made available at the system level so that it can be distributed among the different applications whenever required. Although we do not propose any particular method in this paper for the distribution of the spare capacity, the methods in [14], [15] are particularly well suited for the spare capacity distribution.

c) PRM Elastic Scheduler: We now discuss how the elastic manager and the GCSR manager work together to adapt the application as well as system resources to maintain schedulability. The pseudo-code is presented in Algorithm 2. The functional behaviour is split between the elastic manager and the GCSR manager. The Elastic manager takes as input the request for T_i^{new} and if T_i^{new} is greater than or equal to the T^{min} of the application, it uses the elastic compression

Algorithm 2 PRM Elastic Scheduler

```
1:  function GET_PERIOD_INTERFACE(A, T^min, k)
2:      U_Γ^new ← FIND_UTILIZATION_BOUND(U_A)
3:      Γ(Θ_new, Π_new) ← FIND_SOLUTION(k, A, U_Γ^new)
4:      success ← REQUEST_RESOURCE_UPDATE(Γ(Θ_new, Π_new))
5:      if success == true then
6:          return Γ(Θ_new, Π_new), U_Γ^new
7:      else
8:          return Failure
9:      end if
10: end function
11: function ELASTIC_MANAGER(T_i^new)
12:     T_i^d ← T_i^new
13:     if T_i^new ≥ T^min then
14:         TASK_COMPRESS(A, U_Γ)
15:     else
16:         T^min ← T_i^new
17:         Interface ← GET_PERIOD_INTERFACE(A, T^min, k)
18:         if Interface! = Failure then
19:             TASK_COMPRESS(A, U_Γ^new)
20:         else
21:             handleFailure()
22:         end if
23:     end if
24:     return
25: end function
```

algorithm to find a feasible period reassignment. Before it calls the task compression algorithm, it modifies the period parameter of the task from T_i^d to T_i^{new}. The task compression algorithm then takes as input the updated taskset parameters and the current resource supply utilization U_Γ to adapt the periods of the tasks to keep the application schedulable. If T_i^{new} is less than T^{min} of the application, it sets the T^{min} value equal to T_i^{new}. It then generates the new resource supply parameters via the `GET_RESOURCE_INTERFACE` function. This function takes as input the updated taskset parameters and the value k satisfying Eq. (2) or Eq. (4). It then finds the resource supply utilization bound according to Eq. (5) or Eq. (6). It uses this value to find a solution according to the approach given in [10]. The Elastic manager requests the GCSR manager to modify its resource supply parameters via the `REQUEST_RESOURCE_UPDATE` function. The GCSR manager tries to allocate resources from the spare capacity while maintaining system schedulability. It returns success if the requested resource supply parameters can be accommodated or returns failure along with the maximum resource supply utilization that it can provide. In case of failure, it is up to the application to decide on how to handle this failure.

IV. Evaluation

We evaluate the performance of the proposed framework in the context of EDF scheduling. To demonstrate the advantages of the proposed method, we generated 900 random tasksets with each taskset consisting of 10, 20 or 30 tasks. For each

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

taskset, we set the initial desired utilization equal to 0.25, 0.5, and 0.75. The utilization for each task was then was derived using the algorithm proposed by Griffin et al. [16]. The initial desired periods were chosen at random from a normal distribution in the range [10,100]. The WCET values were set as $C_i = U_i * T_i$. The minimum and maximum periods for each of the tasks were assigned as a function of the initial desired period, i.e., to fix the minimum period, we subtracted a random percentage in the range [10-50] from the desired period. Similarly, for the maximum period, we added a random percentage in the range [10-50] to the desired period. We assigned random integer values from a normal distribution n the range [0-10] as the elastic coefficients of the tasks. For each taskset, we then derived a periodic resource interface for the initial desired periods according to the algorithm in [10].

We set up the experiments according to the different configurations of the number of tasks N and the total desired utilization U, i.e., each configuration was defined as a pair(N,U). For each configuration, 100 random tasksets were generated. For each configuration and a random taskset, we requested a change in the desired period 100 times. For each new period request, we assigned the new period values chosen from a uniform distribution within their defined period ranges. For each configuration, we counted the number of times the elastic manager was able to modify the utilization such that the application remains schedulable. If the elastic manager failed to find a feasible solution, it would adapt the interface bandwidth[1]. The task requesting for a new period was chosen at random for each of the new period request.

Fig. 3 shows the distribution of requests between the elastic manager and the GCSR manager for 300 different configurations with N equal to 10 and the total utilization set to 0.25, 0.5, and 0.75. We can observe that the elastic manager was able to successfully handle a significantly large percentage of the new period requests locally. For lower utilization values, the percentage of requests handled locally was less than the percentage for the higher utilization. Fig. 4 shows the distribution of the requests between the elastic manager and the GCSR manager for 300 configurations with N equal to 20 and the total utilization set to 0.25, 0.5, and 0.75. Similar to the previous observations, the elastic manager was able to successfully handle a large percentage of the requests locally, while for lower utilization values, the percentage was less than the percentage for higher utilization. For the remaining 300 configurations, we set N equal to 30 and the total utilization was set to 0.25, 0.5, and 0.75. In this case, the elastic manager was able to successfully handle a higher percentage of requests at lower utilization values when compared to higher utilization tasksets. This is shown in Fig. 5. Here, even when compared to the lower number of tasks with the same total utilization, there are more requests for system-level bandwidth adaptation.

In another experiment, we modified the range of the minimum and maximum periods of the taskset from the initial [10,50] percent values to [10,100] percent. When the difference between the initial desired period and the minimum and maximum periods is increased, fewer requests were handled at the application level compared to the system level. As shown in Fig. 6, for the configuration of 10 tasks and utilization set to 0.25, more than 70% of the requests are for bandwidth adaptation when the difference between the minimum and maximum periods was changed to [10,100] percent from [10-50] percent. Another significant observation is that for certain configurations and tasksets, all of the 100 new period requests could either be handled locally by the elastic manager or handled only at the system level by the GCSR manager. This can be observed in Fig. 7. This indicates that the approach based on setting the initial resource supply according to the initial desired periods can have a considerable impact on the number of requests that need bandwidth adaptations.

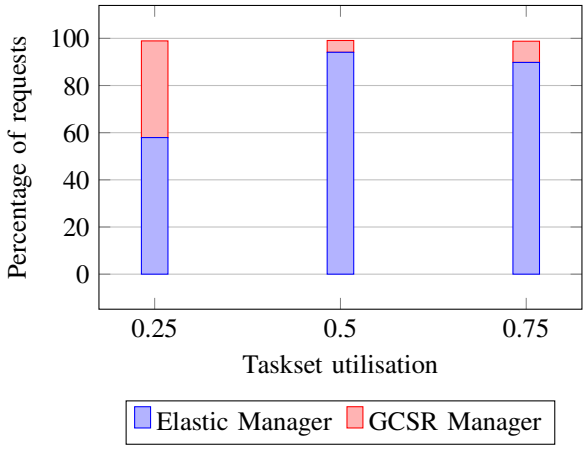

Fig. 3. Percentage of requests handled by Elastic and GCSR manager for taskset size 10.

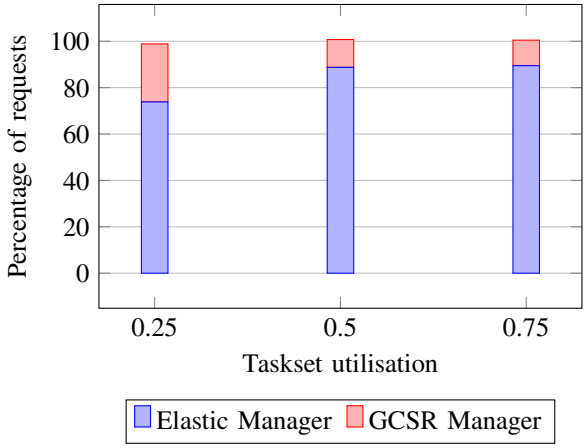

Fig. 4. Percentage of requests handled by Elastic and GCSR manager for taskset size 20.

V. Related Work

The concept of elastic tasks was introduced by Buttazzo et al. [3] to model applications whose computational demands

[1]Note that for this evaluation we did not check for system schedulability since a failure of system schedulability test could only mean that the application's resource demands were not feasible.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

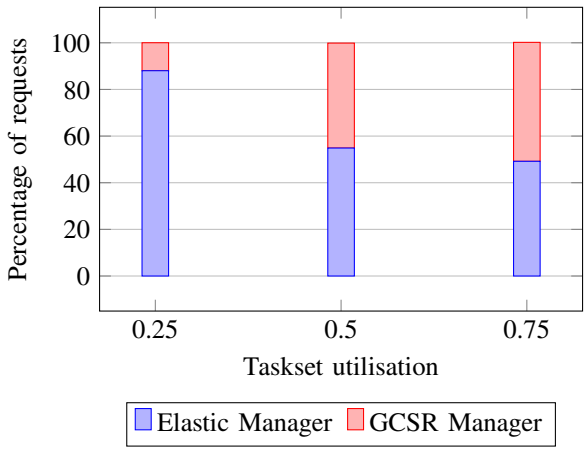

Fig. 5. Percentage of requests handled by Elastic and GCSR manager for taskset size 30.

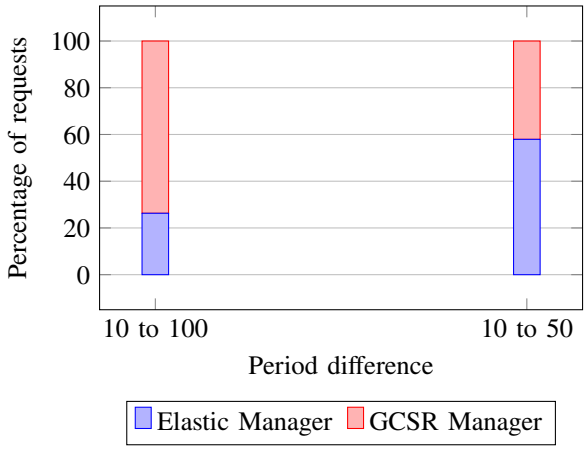

Fig. 6. Percentage of requests handled by Elastic and GCSR manager for different intervals between the minimum and the maximum periods.

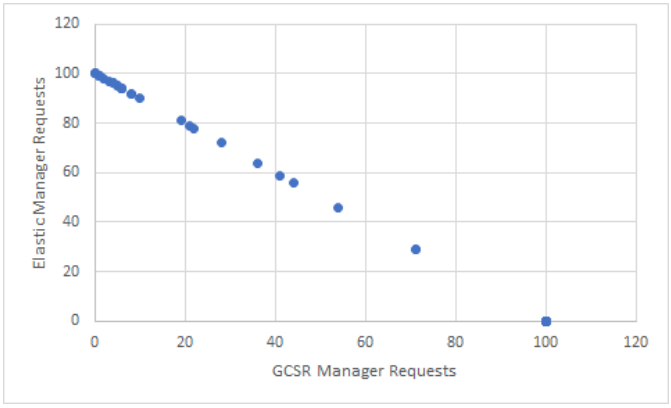

Fig. 7. Requests handled by Elastic and GCSR Manager for 100 tasksets of a single configuration (20,0.25).

can occasionally exceed the available capacity by allowing the application to modify the demand by changing the frequency of its jobs through an elastic coefficient. This was extended to address resource sharing within the elastic task model in [11]. Chantem et al. [17], [18] reformulated the problem as a quadratic optimization problem and showed that the original elastic tasks compression algorithm was indeed a solution to solving a quadratic problem. Tian et al. [19] extended the modified problem to include a "Quality-of-Control" metric as a part of the objective function of the quadratic optimization problem. More recently, Orr et al. [20], [21] provided algorithms to schedule sequential elastic tasks on multiprocessor systems and further extended the concept of the elastic task to federated DAG-based parallel task systems in [22], [23]. Beccari et al. [24], [25] provided alternative algorithms to schedule similar applications by expressing the task period ranges in a linear programming formulation.

Hierarchical scheduling of applications was encapsulated in a compositional real-time scheduling framework by Lee et al. [10]. In this framework, the computational demand of an application was abstracted with a single demand interface as a pair of capacity and period and the resource supply server was abstracted as a periodic resource model where each server was guaranteed a reserved capacity Θ every Π time units. Easwaran et al. [26] extended the periodic resource model to include the deadline parameter, where each server was guaranteed a reserved capacity Θ within D time units, in every time interval Π. Dewan et al. [27], [12] provided algorithms to find an approximate allocation of bandwidth for a set of periodic and sporadic tasks under the periodic resource model. Khalilzad et al. [6] proposed an adaptive hierarchical scheduling model to accommodate the adaptive behaviour of the periodic and sporadic tasks by changing the bandwidth allocation. In contrast, this paper assumes that a bandwidth allocated for a server under the periodic resource model remains constant and that the workload within the server can be adapted according to the elastic task model. However, if the elastic assignment fails, a request for a new bandwidth allocation will be made. We note that the proposed solution does not take into account possible bandwidth reclamation or mixed-criticality-based approaches to assign new bandwidths if no schedulable allocation can be made. Instead, we leave it to the individual application to handle such failures.

VI. CONCLUSION

Many real-time applications designed to accommodate their behaviour at run-time depend upon user configuration or the physical environment in which they operate. Further, for open real-time systems, the applications can be developed independently and can be run on the same hardware. To accommodate such adaptive behaviour and minimize the impact of an individual application's variability in its timing and resource demands, we proposed a two-level scheduling framework based on the periodic resource model and the elastic task model. We provided a mechanism based on the utilization tests to enable the execution of the elastic applications in a compositional real-time system. Further, by combining the application-level adaptation along with the system-level bandwidth modifications, we have shown that a large percentage

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.

of task modifications can be handled by the framework at the application level. If the local adaptation fails, the system-level reallocation provides an additional mechanism to support the scheduling of elastic applications. However, for some cases, if both the levels fail to find a schedulable modification, it is up to the application to handle such failures. Overall, our combined approach improves the number of application-level variations that can be handled without affecting the property of independence of other co-running applications of the same processor. In future work, we intend to investigate techniques related to mixed-criticality and compositional real-time systems to reallocate resources at the system level.

VII. Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation, and by the Swedish Knowledge Foundation (KKS) under the FIESTA project.

References

[1] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "Rate modulation of soft real-time tasks in autonomous robot control systems," in *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, 1999, pp. 21–28.

[2] D. Simon, D. Robert, and O. Sename, "Robust control/scheduling co-design: application to robot control," in *11th IEEE Real Time and Embedded Technology and Applications Symposium*, 2005, pp. 118–127.

[3] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings - Real-Time Systems Symposium*. IEEE, 1998, pp. 286–295.

[4] S. M. Salman, V. Struhar, A. V. Papadopoulos, M. Behnam, and T. Nolte, "Fogification of industrial robotic systems: Research challenges," in *Proceedings of the Workshop on Fog Computing and the IoT*, 2019, pp. 41–45.

[5] M. S. Shaik, V. Struhár, Z. Bakhshi, V.-L. Dao, N. Desai, A. V. Papadopoulos, T. Nolte, V. Karagiannis, S. Schulte, A. Venito *et al.*, "Enabling fog-based industrial robotics systems," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 61–68.

[6] N. M. Khalilzad, T. Nolte, M. Behnam, and M. Åsberg, "Towards adaptive hierarchical scheduling of real-time systems," in *ETFA2011*, 2011, pp. 1–8.

[7] N. M. Khalilzad, M. Behnam, and T. Nolte, "Multi-level adaptive hierarchical scheduling framework for composing real-time systems," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, pp. 320–329.

[8] N. Khalilzad, M. Ashjaei, L. Almeida, M. Behnam, and T. Nolte, "Towards adaptive resource reservations for component-based distributed real-time systems," *SIGBED Rev.*, vol. 12, no. 3, p. 24–27, Aug. 2015.

[9] N. Khalilzad, F. Kong, X. Liu, M. Behnam, and T. Nolte, "A feedback scheduling framework for component-based soft real-time systems," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 182–193.

[10] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 30:1–30:39, 2008.

[11] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, 2002.

[12] F. Dewan and N. Fisher, "Bandwidth allocation for fixed-priority-scheduled compositional real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4, pp. 91:1–91:29, 2014.

[13] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006, pp. 159–168.

[14] R. Marau, K. Lakshmanan, P. Pedreiras, L. Almeida, and R. Rajkumar, "Efficient elastic resource management for dynamic embedded systems," *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*, pp. 981–990, 2011.

[15] S. Groesbrink, L. Almeida, M. De Sousa, and S. M. Petters, "Towards certifiable adaptive reservations for hypervisor-based virtualization," *Real-Time Technology and Applications - Proceedings*, vol. 2014-October, no. October, pp. 13–24, 2014.

[16] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 76–88.

[17] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling," *Proceedings - Real-Time Systems Symposium*, pp. 236–245, 2006.

[18] ——, "Generalized elastic scheduling for real-time tasks," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 480–495, 2009.

[19] Y. C. Tian and L. Gui, "QoC elastic scheduling for real-time control systems," *Real-Time Systems*, vol. 47, no. 6, pp. 534–561, 2011.

[20] J. Orr, C. Gill, K. Agrawal, J. Li, and S. Baruah, "Elastic Scheduling for Parallel Real-Time Systems," *ACM Subject Classification*, vol. 6, no. 1, pp. 5:1–5:14, 2019.

[21] J. Orr and S. Baruah, *Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation*. Springer US, 2021, vol. 57, no. 1.

[22] J. Orr, C. D. Gill, K. Agrawal, S. K. Baruah, C. Cianfarani, P. Ang, and C. Wong, "Elasticity of workloads and periods of parallel real-time tasks," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*. ACM, 2018, pp. 61–71.

[23] J. Orr, J. C. Uribe, C. D. Gill, S. K. Baruah, K. Agrawal, S. Dyke, A. Prakash, I. Bate, C. Wong, and S. Adhikari, "Elastic scheduling of parallel real-time tasks with discrete utilizations," in *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, L. Cucu-Grosjean, R. Medina, S. Altmeyer, and J. Scharbarg, Eds. ACM, 2020, pp. 117–127.

[24] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "Rate modulation of soft real-time tasks in autonomous robot control systems," in *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*. IEEE Computer Society, 1999, pp. 21–28.

[25] G. Beccari, S. Caselli, and F. Zanichelli, "A technique for adaptive scheduling of soft real-time tasks," *Real Time Syst.*, vol. 30, no. 3, pp. 187–215, 2005.

[26] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 129–138.

[27] N. Fisher and F. Dewan, "A bandwidth allocation scheme for compositional real-time systems with periodic resources," *Real Time Syst.*, vol. 48, no. 3, pp. 223–263, 2012.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on January 18,2022 at 15:33:52 UTC from IEEE Xplore. Restrictions apply.