Elastic Scheduling of Parallel Real-Time Tasks with Discrete Utilizations

James Orr
Washington University in St. Louis
james.orr@wustl.edu

Johnny Condori Uribe
Purdue University
jcondori@purdue.edu

Chris Gill
Washington University in St. Louis
cdgill@wustl.edu

Sanjoy Baruah
Washington University in St. Louis
baruah@wustl.edu

Kunal Agrawal
Washington University in St. Louis
kunal@wustl.edu

Shirley Dyke
Purdue University
sdyke@purdue.edu

Arun Prakash
Purdue University
aprakas@purdue.edu

Iain Bate
University of York
iain.bate@york.ac.uk

Christopher Wong
Brown University
christopher_wong2@brown.edu

Sabina Adhikari
Stephen F. Austin State University
adhikaris3@jacks.sfasu.edu

ABSTRACT

Elastic scheduling allows for online adaptation of real-time tasks' utilizations (via manipulation of each task's computational workload *or* period) in order to maintain system schedulability in case the utilization demand of one or more tasks changes. This is done currently by assigning each task a utilization (and therefore period or workload) from within a continuous range of acceptable values. While this works well for anytime tasks whose quality of service improves with duration or for tasks that can run at any rate within a given range, many computationally-elastic tasks have a specific workload for each distinct mode of operation and therefore cannot perform arbitrary amounts of work. Similarly, some period-elastic tasks must run at specific (e.g. harmonic) rates. Therefore, a discrete set of candidate utilizations per task must be accommodated in such cases.

This paper provides a new elastic task model in which each task has a discrete set of possible utilizations (instead of a continuous range). This allows users to specify only relevant candidate periods and workloads for each task. The discrete nature of this model also allows each task to modify its workload *and/or* its period when changing its mode of operation, instead of adapting in only one dimension of task utilization. Elastic tasks thus can exploit both period elasticity and computational elasticity. This greatly increases both the diversity of adaptations available to each task and the kinds of real-time tasks relevant to elastic scheduling.

We use the real-world example of real-time hybrid simulation as a motivating application domain with discretely computationally-elastic, period-elastic, and combined-elastic parallel real-time tasks under the Federated Scheduling paradigm. We prove the scheduling of these tasks to be NP-hard, and provide a pseudo-polynomial time scheduling algorithm. We then use this scheduling algorithm to implement the first virtual real-time hybrid simulation experiment in which discrete elastic adaptation of platform resource utilizations enables adaptive switching between controllers with differing computational demands. We also study the effects of scheduling tasks with discretized vs. continuous candidate utilizations.

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture**; *Real-time system specification*; Embedded software.

KEYWORDS

real-time scheduling, discrete elastic tasks, real-time hybrid simulation

ACM Reference Format:
James Orr, Johnny Condori Uribe, Chris Gill, Sanjoy Baruah, Kunal Agrawal, Shirley Dyke, Arun Prakash, Iain Bate, Christopher Wong, and Sabina Adhikari. 2020. Elastic Scheduling of Parallel Real-Time Tasks with Discrete Utilizations. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394810.3394824

1 INTRODUCTION

The ***elastic task model***, first introduced by Buttazzo et al. [3], allows for online modification of task periods to maintain schedulability of adaptive ***period-elastic*** tasks without the pessimism required for a static schedule accommodating the worst-case behavior of the most utilization-intensive mode of operation. That model was later extended to include multiprocessor scheduling, tasks with internal parallelism, and tasks that instead can adapt their computational loads (***computational elasticity***). [16–18]

RTNS 2020, June 9–10, 2020, Paris, France

ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394824

In this paper we provide a new elastic task model that expands the state of the art by introducing ***discrete elastic scheduling*** in which each task's assigned utilization is obtained from a finite set of candidate tuples, each of which has an associated period and workload. From one tuple to the next, a task may change its period, its computational workload, or ***both***. The discrete elastic model more accurately describes tasks that have distinct modes of operation, such as a robot with multiple available planning algorithms with varying degrees of computational demand, or a control application that may get better results from running at a higher frequency but needs to maintain harmonic rates with respect to other tasks in the system. Unlike the continuous elastic model, the discrete model allows adaptation of both computational demand and period together, at once (***combined elasticity***).

We use the real-world application domain of ***real-time hybrid simulation (RTHS)***, used by earthquake engineers to understand structural behavior with high fidelity at realistic timescales [9, 10, 19], as a motivating example for discrete elastic scheduling. In RTHS a well-understood portion of a structure is simulated while a portion to be tested or validated is physically built. The combined structure is then connected via sensors and actuators and subjected to external stimuli (such as earthquake ground motions) at fine-grained time scales in order to examine how the relevant portions behave. Different portions of the structure can be simulated at different rates to yield resources to portions of special interest (e.g., those near the physical specimen) that require higher resource utilization. However, to date, resources have been statically assigned in RTHS experiments: each substructure runs at a fixed rate with a fixed set of computational resources, and changes to the system can only be made between successive runs. We exploit discrete elastic scheduling to conduct the first (virtual) real-time hybrid simulation experiment in which resource adaptation enables adaptive switching between controllers with different computational demands. In this experiment, the control algorithm that determines the response to the system's behavior is able to execute in multiple modes of operation, i.e., using a non-linear Kalman filter vs. a more computationally-expensive particle filter. Other tasks in the system (which must run at rates harmonic with that of the control algorithm) are similarly able to adapt their periods, computational loads, or both, accordingly.

This paper is structured as follows. Section 2 provides relevant background information. Section 3 presents the ***discrete elastic scheduling system model***, including a discussion of the implications of ***combined elasticity***, which allows for a task to adapt both its computational workload *and* its period. In Section 3 we also prove the scheduling of parallel tasks using this model under the Federated Scheduling paradigm to be (weakly) NP-hard via a reduction from the Knapsack Problem. We then present a pseudo-polynomial time dynamic-programming algorithm (obtained by reducing our scheduling problem to an instance of the Multiple Choice Knapsack Problem) that can efficiently create an optimal schedule for such tasks. Section 4 describes our adaptive virtual RTHS experiment. Section 5 evaluates the level of pessimism when using discrete elastic scheduling vs. idealized (but often practically unsuitable) continuous elastic scheduling. Section 6 concludes and describes future directions for extending this work.

Although this paper focuses on the discrete elastic scheduling of parallel real-time tasks under federated scheduling, we point out that many of the concepts introduced here are also applicable to sequential tasks; hence, our proposed model should be considered an extension of the elastic task models for sequential *and* parallel workloads.

2 BACKGROUND

In this paper we present the novel concept of *discrete elastic scheduling*, focusing on discretely elastic parallel real-time tasks under the Federated Scheduling paradigm. This section provides background information about *elastic scheduling* and the example application domain that motivates our approach and is used to evaluate it: *real-time hybrid simulation (RTHS)*.

2.1 Elastic Scheduling

The (continuous) ***elastic task model*** was first proposed by Buttazzo et al. [3] for scheduling adaptive sequential tasks on uniprocessor systems via period manipulation. The approach is based on a sophisticated analogy between (1) uniprocessor tasks maintaining a collective utilization no greater than a desired utilization U_d (e.g. for schedulability, $U_d = 1.0$ for preemptive EDF scheduling) and (2) a set of springs laid end-to-end being compressed by a collective force until their combined length is at or below a desired maximum length. Just as springs have different maximum and minimum lengths and resistances to compression, elastic tasks have different minimum and maximum period values (and therefore different maximum and minimum utilizations) and resistances to changing their periods [3]. Each task is formally represented as $\tau_i = \left\langle C_i, T_i^{(max)}, T_i^{(min)}, E_i \right\rangle$ where C_i represents the task's constant *worst-case execution time (WCET)* and the closed range $[T_i^{(min)}, T_i^{(max)}]$ spans all acceptable period values for a task, where a lower period (and therefore higher utilization) is always preferred. The *current* period is denoted T_i. A task's *elasticity coefficient* E_i is a measure of how relatively easy or difficult it is to change a task's period, analogous to a spring's stiffness as a measure of its resistance to changing its length: a higher elasticity coefficient indicates a more elastic task.

Buttazzo et al. present an efficient ($\Theta(n^2)$) iterative scheduling algorithm [3] that increases each task's period T_i from $T_i^{(min)}$ proportional to its elasticity coefficient E_i (to a maximum of $T_i^{(max)}$). Recall that a task's utilization $U_i = \frac{C_i}{T_i}$. With a constant C_i, the values $T_i^{(min)}$ and $T_i^{(max)}$ therefore can be expressed equivalently as maximum and minimum utilizations $U_i^{(max)}$ and $U_i^{(min)}$, respectively. The algorithm ends either when tasks successfully have been assigned periods such that their combined utilization is less than U_d, or when each task's period has been stretched to $T_i^{(max)}$ (giving $U_i^{(min)}$) and their combined minimum utilization is still greater than U_d, in which case the taskset is declared unschedulable. Chantem et al. [6, 7] later proved this algorithm to be equivalent to solving the following optimization problem:

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i}(U_i^{(max)} - U_i)^2 \tag{1}$$

such that

$$U_i^{(min)} \leq U_i \leq U_i^{(max)} \text{ for all } \tau_i$$

and

$$\sum_{i=1}^{n} U_i \leq U_d.$$

Uniprocessor elastic scheduling has since been expanded to include constrained deadlines [6], resource sharing [4] and unknown computational loads [5].

The ***Federated Scheduling*** paradigm was first introduced by Li et al. [15] to schedule sporadic parallel tasks represented as ***directed acyclic graphs (DAGs)***, each with a utilization $U_i \geq 1$ that demands more than a single processor. These ***high-utilization tasks*** are each given exclusive use of m_i processors according to the equation

$$m_i = \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil \tag{2}$$

In Equation 2, C_i is the task's cumulative ***work***, or the sum of the worst-case execution times of all nodes in its DAG of precedence-constrained sub-tasks. This is equivalent to the task's worst-case execution time if run on a single processor. Similarly L_i is its ***span*** (or ***critical-path length***), the longest worst-case execution time of any sequential chain of nodes in the DAG. This forms a lower bound on the task's execution time on a theoretically infinite number of processors. T_i is each task's ***minimum inter-arrival time or period***, which also serves as its ***implicit deadline***. It was proved [15] that a taskset composed of high utilization tasks is schedulable if the required number of processors is less than or equal to m, the number of processors available to the system. ***Low-utilization tasks*** with utilization $U_i < 1$ are treated as sequential tasks under Federated Scheduling and are scheduled on the pool of remaining processors.

Later work by Orr et al. [17, 18] extended the elastic task model to include *parallel real-time DAG tasks* under Federated Scheduling. To keep parallel elastic scheduling as semantically equivalent to Buttazzo's original model as possible, the authors present an optimal scheduling algorithm that directly solves a minimization problem similar to that given in Equation 1:

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i}(U_i^{(max)} - U_i)^2 \tag{3}$$

such that:

$$U_i^{(min)} \leq U_i \leq U_i^{(max)} \text{ for all } \tau_i$$

and

$$\sum_{i=1}^{n} m_i \leq m$$

Each task is initially given its minimum number of processors, and the remaining CPUs are allocated in a manner that minimizes the sum in Equation 3. That work also expanded the concept of task elasticity. Noting that task utilization is dependent on *both computational load and period*, it allows for tasks to have a range of acceptable utilizations $[U_i^{(min)}, U_i^{(max)}]$ that can be either a range of acceptable periods $[T_i^{(min)}, T_i^{(max)}]$ as in Buttazzo's model ***or*** a range of acceptable computational loads $[C_i^{(min)}, C_i^{(max)}]$. Tasks that adapt their periods are called ***period-elastic tasks***, while tasks that adapt their workloads are ***computationally-elastic tasks***. This paper extends that elastic task model, (1) allowing for the more realistic scenario of discrete candidate utilization values instead of continuous ranges; and in doing so also (2) allowing for ***combined-elastic tasks*** to adapt both their periods and computational loads at once.

2.2 Motivating Application Domain

Although the adaptive capabilities and discrete workloads enabled by discrete elastic scheduling are relevant to a variety of real-time applications, we focus here on ***real-time hybrid simulation (RTHS)***, which is used by structural engineers to study the dynamic behavior of a structural specimen under loading that potentially results in unknown and highly nonlinear behavior. Traditionally, a new structural concept or a new vibration mitigation device is validated in one of two ways: a physical structure is built and subjected to tests, or a numerical model is tested via computer simulations. However, building physical structures, even if not at full scale, and subjecting them to full physical tests, though robust, can be prohibitively expensive in terms of money and time. On the other hand, running computer simulations such as finite element models is less expensive but may not fully capture nuances of a physical structure: for instance, accurate numerical models may not exist for some types of damage that a physical structure could sustain.

Realy-time hybrid simulation (RTHS) [9, 10] combines the strengths of purely physical and purely numerical approaches. A portion of a structure is physically built to be studied, while the remainder is simulated numerically. The complete structure (composed of both physical and simulated components) is then dynamically subjected to external loads (such as earthquake ground motions) during experimentation, resulting in a feedback control system with numerical models that must be executed on-line. At fine-grained time scales with real-time requirements, the physical components are driven by actuators, and their displacement, velocity, and acceleration are measured by sensors and input back into the computational subsystem. The resulting computation in turn determines the forces the actuators should apply to the physical substructure in the next time step. A widely-used platform for RTHS is MathWorks's Speedgoat/XPC Target that runs in coordination with real-time Simulink. However, such a system is neither parallel nor adaptive, which limits the kinds of experiments that it can run.

The potential for extensive damage to equipment, test specimens, or even people as a result of unintended actuation (e.g., in the case of an unstable control algorithm) necessitates that before full RTHS experimentation can be done safely, as much validation of the proper system setup as possible must be performed. One such validation that always precedes a RTHS is a ***virtual RTHS*** in which the physical component of RTHS is replaced by a simulation, often on an entirely different machine and using the same interface as the physical component. Although the simulated "physical component" in a virtual RTHS cannot fully capture the dynamics of the actual physical specimen under examination in the full RTHS (indeed the partially unknown dynamics of the physical specimen may be the very reason for running the RTHS experiment), a virtual RTHS can effectively validate control algorithms and numerical models that will be used in RTHS experiments. As such, in this paper we present an adaptive virtual RTHS using discrete elastic scheduling

(in Section 4) as a crucial first step towards a full adaptive RTHS using our new discrete elastic scheduling model.

Multi-time-stepping (MTS) decomposes an RTHS into subsystems (with individual tasks) and runs each task at its own harmonic periodic rate, where for any two subsystems, the periodic rate of one has a ***time-step ratio*** of x times that of the other. Data are exchanged at each iteration of the slower of the tasks to ensure subsystems have a consistent view of the overall system. Multi-time-stepping allows for more precise control over individual subsystems' periods (e.g., one subsystem runs relatively quickly in order to read a vital physical sensor more frequently or another subsystem runs more slowly in order to process more simulation data in each period) than if the entire system were running at a single periodic rate. However, multi-time-stepping alone does not allow for fine-grained control over tasks' computational loads. Nor does it allow for run-time re-allocation of resources (e.g., which would allow for a subsystem's runtime behavior to change with its workload) [2].

The Cybermech platform was developed by Ferry et al. [9] to run parallel RTHS experiments. Although Cybermech supports multi-time-stepping, each subsystem only runs at a fixed periodic rate [2], and thus is only applicable to systems whose control model is linear. In contrast, the discrete elastic scheduling approach introduced in this paper allows for dynamic re-allocation of individual subsystems' periodic rates and/or computational resources to accommodate linear and potentially non-linear behavior which can occur with new experimental devices (e.g., for energy-dampening). We demonstrate such adaptive resource management capabilities and use them to enable adaptive switching between controllers with differing computational demands for the first time in a virtual RTHS as is described in Section 4.

3 DISCRETE ELASTIC SCHEDULING

In this section we present a new discrete elastic task model for parallel real-time systems. We then discuss implications of the *combined-elastic* adaptations enabled by this model. We also prove that scheduling of discrete elastic tasks under Federated Scheduling is NP-Hard in the weak sense, and provide a pseudo-polynomial time algorithm for scheduling them. [1]

3.1 Task Model

Similar to the continuous elastic task model, in the discrete elastic task model, each task τ_i has elasticity coefficient E_i and the assigned utilization U_i of each task can range between $U_i^{(min)}$ and $U_i^{(max)}$. However, in the discrete model, rather than allowing any utilization within the continuous range $[U_i^{(min)}, U_i^{(max)}]$, each parallel task τ_i has exactly k_i discrete modes of operation. Each mode of operation j ($1 \leq j \leq k_i$) for each task has a specific period (and implicit deadline) $T_i^{(j)}$, work $C_i^{(j)}$, and span $L_i^{(j)}$. The candidate utilizations for the task come from the period and work in each of these modes of operation $U_i^{(j)} = C_i^{(j)}/T_i^{(j)}$, and $U_i^{(min)}$ and $U_i^{(max)}$ are the lowest and highest such utilizations, respectively. In strictly period-elastic tasks, all modes have the same work and span values (i.e., $\forall x, y; 1 \leq x \leq k_i, 1 \leq y \leq k_i; C_i^x = C_i^y, L_i^x = L_i^y$). Similarly, all modes of operation in strictly computationally-elastic tasks have the same period (i.e., $\forall x, y; 1 \leq x \leq k_i, 1 \leq y \leq k_i; T_i^x = T_i^y$). We use Equation 2 to determine $m_i^{(j)}$, the number of processors required to schedule τ_i in mode j. [2]

We seek to schedule n tasks on m processors by selecting a mode of operation j for each task τ_i ($1 \leq j \leq k_i$) while minimizing Equation 3. A pseudo-polynomial time algorithm for this is presented in Section 3.4. The tasks may be fully independent, in which case there are no restrictions on the potential modes of operation for each task. In other cases, the potential modes of operation may encode dependencies among tasks (e.g., all tasks must run at rates harmonic to a base control rate). Even with those encoded dependencies, we assume that each task is free to change among its modes independently of the other tasks (e.g. any such rate dependencies are encapsulated by only allowing modes that contain such harmonic rates).

On its face the discrete elastic task model presented in this paper is similar to one used decades ago by Kuo and Mok [14] to model adaptive real-time tasks. However, there are several key differences. Both models have a set of adaptive tasks with candidate modes of operation. However, whereas our model allows for arbitrary C_i and T_i combinations between modes of operation, the model presented by Kuo and Mok scales task periods and workloads under a constant utilization. For instance, $\tau_i = (C_i, T_i)$ may have candidate modes $(2, 4), (2.5, 5), (3, 6)$ in [14] where all modes necessarily have a utilization of 0.5. This is allowed in the discrete elastic model presented here. However, a fourth candidate mode of $(2, 5)$ with utilization 0.4, which is also acceptable in our model, is not allowed in theirs. Furthermore [14] seeks to assign periods in such a way as to maximize harmonic chains and therefore maximize schedulability on a uniprocessor. The period-assignment problem asks whether there is a period assignment such that the maximum harmonic base is at least a certain value. This problem is proven to be strongly NP complete (i.e., no pseudo-polynomial time algorithm exists unless P=NP). The problem considered here is fundamentally different. This model does not (necessarily) care about harmonic chains and uses a pseudo-polynomial time dynamic programming algorithm for utilization selection.

3.2 Discussion

The continuous elastic task model allows for tasks to adapt their periods *or* their workloads to any value over a continuous range depending on the needs of the system. This is useful for many kinds of tasks. Consider, for instance, an anytime algorithm [8] that can return a valid answer at any instant with the quality of the answer potentially improving as the algorithm is allowed to run longer. Such an algorithm can be modeled as a task with an elastic computational requirement that may vary over a continuous range. However, not all algorithms are anytime algorithms: for some tasks, meaningful results are only returned if the algorithm is allowed to execute for certain specific durations. In a similar vein,

[1] In this paper we focus on scheduling high-utilization tasks ($U_i \geq 1$) via Federated Scheduling, although low-utilization tasks ($U_i < 1$) can be (partitioned if necessary and then) scheduled sequentially on a uniprocessor in a fashion similar to that described in this section by focusing on keeping their aggregate system utilization below a desired utilization U_d.

[2] We assume each task receives at least 1 dedicated CPU under Federated Scheduling. Any mode of operation with $U_i^{(j)} <= 1$ will receive a single dedicated processor.

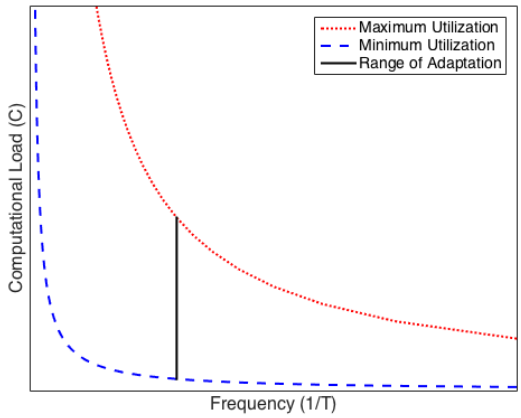

Figure 1: Continuous Computationally-Elastic Task

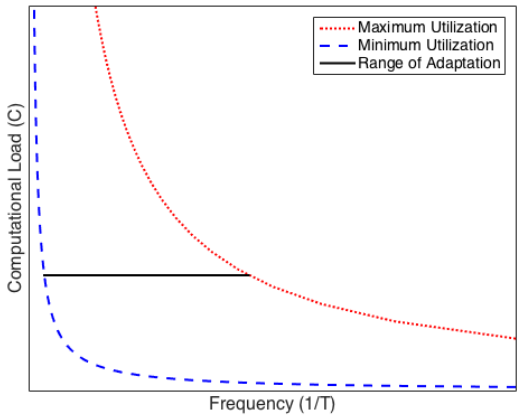

Figure 2: Continuous Period-Elastic Task

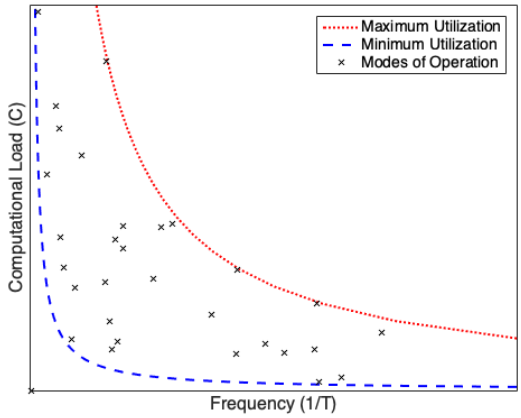

Figure 3: Discrete Combined-Elastic Task

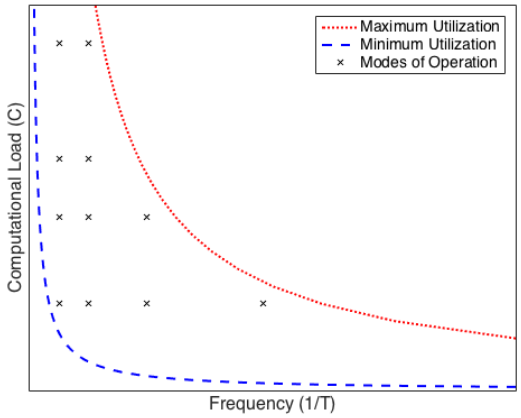

Figure 4: Discrete Workloads and Harmonic Rates

periodic tasks that form part of a control loop may need to execute at frequencies (and hence period values) that are consistent with the remainder of the control loop (e.g., harmonic with respect to the base system frequency), and cannot operate with arbitrary periods.

Therefore, the continuous elastic task model is not appropriate for some important kinds of tasks. This becomes more apparent when one considers that on actual hardware, task execution times are essentially discrete. Processors treat time not as a continuous interval but as a discretized count of cycles. Therefore on a general-purpose CPU, no job can actually run for an arbitrary amount of time, but instead executes for an integer number of CPU cycles.

Under the discrete elastic task model, each task τ_i has k_i unique modes of operation, each of which has an associated period and workload. Varying only a single dimension (i.e., changing only the period or workload as in the continuous model) may allow for more appropriate management of the selected attribute than the continuous elastic model. For instance, the discrete elastic task model allows for the guaranteed selection of harmonic periods among period-elastic tasks.

Perhaps an even greater benefit of the discrete elastic model is its ability to allow exploitation of *both period elasticity and computational elasticity*. This ***combined elasticity*** increases the range of potential modes of operation for a given task. Figures 1 – 4 demonstrate the diversity of adaptations enabled by combined elasticity. Each of the four images shows the same task exploiting different types of elasticity. The y-axis is the task's computational load (C), and the x-axis is its frequency ($1/T$). Any point within the allowed region therefore represents a potential work and period assignment for the task. Constant values $U^{(min)}$ and $U^{(max)}$ are represented by dashed and dotted curves, respectively, so any valid assignment of C and T must therefore fall between these two curves.

Figures 1 and 2 show the potential period and workload values of a computationally-elastic task and a period-elastic task, respectively, under the continuous elastic task model. Although there are infinitely many acceptable period (or workload) values that keep the utilization between $U^{(min)}$ and $U^{(max)}$, the range of adaptation for a single task is relatively narrow.

Contrast this with Figure 3, which demonstrates the potential period and workload values of combined-elastic tasks enabled by the discrete elastic task model. Although there are finitely many modes

of operation, adaptation is allowed in both computational and period dimensions, potentially offering a much broader adaptation space. Any point in the entire region between the minimum and maximum utilization curves may be a candidate mode of operation. Thirty such (randomly-chosen) points are plotted in Figure 3.

Which (and how many) candidate points are available is then a configurable application-specific concern. System designers can select as many or as few potential modes of operation as appropriate. For example, anytime tasks that can perform arbitrary amounts of work for arbitrary time periods have a multitude of possible period and workload combinations. In other cases (such as RTHS), application constraints such as the need to run at harmonic rates and/or have a fixed set of computational completion points restrict or even determine actual modes of operation. Figure 4 shows a sample RTHS task with four potential harmonic periods and four potential workloads. Note that as Figure 4 illustrates, not all workloads can be run at all harmonic periods since the utilization may exceed the maximum utilization curve as the workload increases or the period decreases.

Finally, we note that some loss of utilization may be incurred by discretization. For instance, if the same period-elastic task were scheduled under both the continuous and discrete elastic models, the continuous model may assign a task a feasible period that is between two discrete candidate periods. To maintain schedulability, the task may need to be assigned the longer of the two periods under the discrete model, thereby resulting in a lower utilization than under the continuous model (at the potential cost of some control performance). However, we note that the smaller the gap between candidate periods in the discrete model, the smaller the loss of such system utilization due to discretization is. Anytime tasks can exploit this small loss of utilization by selecting many potential modes of operation that are close together in both dimensions, to approximate continuous elasticity while gaining the benefit of combined elasticity, at a (potentially acceptable) cost of a longer-running scheduling algorithm (see Section 3.4). We discuss and study potential utilization loss due to discretization further, in Section 5.

3.3 Proof of NP-Hardness

We now prove that the Federated Scheduling of parallel discrete elastic tasks is NP-hard, via a reduction of an instance of the Knapsack Problem [12] to an instance of the Discrete Elastic Scheduling Problem.

Theorem 1. *Discrete Elastic Scheduling is NP-hard.*

Proof: Reduce KNAPSACK to Discrete Elastic Scheduling.

An instance of KNAPSACK is specified as follows:[3]

$$I_{\text{KNAPSACK}} = \left\langle \left\{(s_i, v_i)\right\}_{i=1}^{n}, S, V \right\rangle$$

where the objective is to fill a knapsack of capacity S with items chosen from a set of n items, and item i ($i = 1 \ldots n$) has *weight* s_i and *value* v_i, such that the weight of the selected items sum to no more than the knapsack's capacity S and their combined value is maximized, with a total of at least the target value V.

[3] All parameters are assumed to be rational numbers.

Given such a specification, we construct an instance of the Elastic Scheduling problem with n tasks, each of which has 2 modes of operation, to be scheduled on $(n + S)$ processors. All n tasks have the same period in all modes of operation, denoted x (i.e., all tasks are computationally-elastic–we note that though all tasks in this construction are computationally-elastic, the same algorithm also schedules period-elastic and combined-elastic tasks). We construct each task's first mode of operation as follows: Assign $C_i^{(1)} = L_i^{(1)} = x$ for all i. As a consequence these are all sequential zero-slack modes of operation, and $m_i^{(1)} = 1$ (for all i). For each i, define the second mode of operation as $C_i^{(2)} = x \cdot (1 + s_i)$ and $L_i^{(2)} = 0$. These are "embarrassingly parallel" modes of operation. Note that we consequently have $m_i^{(2)} = (1 + s_i)$. Let elastic coefficient $E_i = s_i^2 / v_i$.

Note that choosing the second mode of the i'th task requires an additional s_i processors (since the first mode requires 1 processor). Let Γ_1 and Γ_2, respectively, denote the tasks for which the first mode and second mode, respectively, are selected. Recall that in Elastic Scheduling, we seek to minimize $\sum_i \frac{1}{E_i}\left(U_i^{(\max)} - U_i\right)^2$. Therefore:

$$\begin{aligned}
&\sum_i \frac{1}{E_i}\left(U_i^{(\max)} - U_i\right)^2 \\
\equiv &\sum_i \frac{1}{E_i}\left(\frac{C_i^{(2)}}{x} - U_i\right)^2 \\
\equiv &\sum_i \frac{1}{E_i}\left(\frac{x \cdot (1 + s_i)}{x} - U_i\right)^2 \\
\equiv &\sum_{i \in \Gamma_1} \frac{1}{E_i}\left((1 + s_i) - U_i\right)^2 + \sum_{i \in \Gamma_2} \frac{1}{E_i}\left((1 + s_i) - U_i\right)^2 \\
\equiv &\sum_{i \in \Gamma_1} \frac{1}{E_i}\left((1 + s_i) - 1\right)^2 + \sum_{i \in \Gamma_2} \frac{1}{E_i}\left((1 + s_i) - (1 + s_i)\right)^2 \\
\equiv &\sum_{i \in \Gamma_1} \frac{s_i^2}{E_i} \\
\equiv &\sum_{i \in \Gamma_1} v_i
\end{aligned}$$

We thereby conclude that a solution to the Discrete Elastic Scheduling Problem in which the function in Equation 3 takes on a value at most

$$\left(\sum_i v_i\right) - V$$

exists if and only if $I_{\text{KNAPSACK}} \in$ KNAPSACK. ■

3.4 Pseudo-Polynomial Time Scheduling Algorithm

MULTIPLE-CHOICE KNAPSACK [20] is similar to KNAPSACK, but rather than selecting items from a single set, there are multiple mutually-exclusive sets, and exactly one item must be chosen from each set in such a way as to maximize profit and ensure a total weight below the knapsack's capacity. We now provide a pseudo-polynomial time algorithm for Discrete Elastic Scheduling by reducing an instance of it to an instance of MULTIPLE-CHOICE KNAPSACK.

A pseudo-polynomial time algorithm. We define the following reduction from Discrete Elastic Scheduling to MULTIPLE-CHOICE

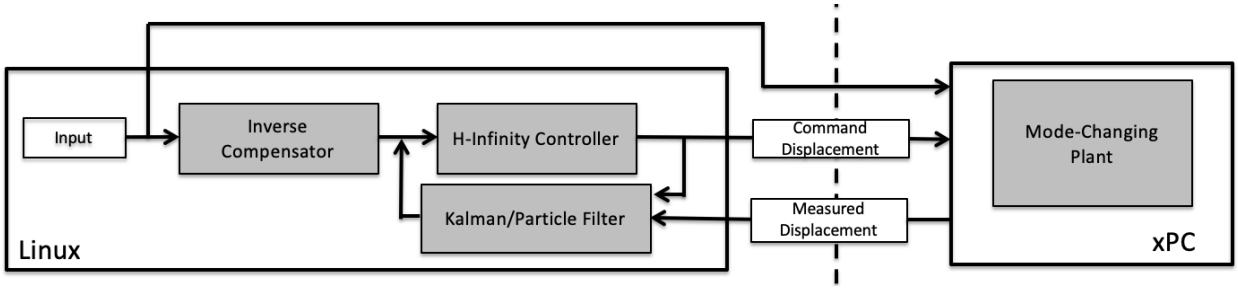

Figure 5: Virtual RTHS Details

Knapsack: each of the n tasks with k_i modes of operation becomes one of n mutually exclusive sets with k_i distinct items. Each task in Discrete Elastic Scheduling needs a mode to be selected, and each set from Multiple-Choice Knapsack needs one item to be selected. Task τ_i operating in mode j becomes an item in the corresponding set with profit $\frac{1}{E_i}\left(U_i^{(\max)} - U_i^j\right)^2$ and weight $m_i^{(j)}$. The knapsack has capacity m. By giving each item weight $m_i^{(j)}$, we ensure that if they fit in a knapsack of capacity m, then the corresponding tasks in the selected modes are schedulable on m processors. Although traditional Multiple-Choice Knapsack seeks to maximize the value of selected items, we instead attempt to minimize the value in Equation 3, which is exactly the profit assigned to each item. We thus use a min() function in place of a max() function that would otherwise be used, which has no bearing on the correctness or complexity of the algorithm.

We note that by successfully selecting one item from each mutually-exclusive set for the knapsack while keeping their combined weight within the knapsack's capacity m, we also select a mode of operation for each task on at most m processors. We therefore have a valid parameterization of the Discrete Elastic Scheduling instance.

A pseudo-polynomial dynamic programming algorithm presented in [13] finds an optimal solution to Multiple-Choice Knapsack by considering the maximum value achievable when considering the first l mutually exclusive sets and reduced knapsack capacity d, in our case, $1 \leq l \leq n$ and $1 \leq d \leq m$. We reproduce a slightly modified version of this algorithm in Algorithm 1: rather than finding the maximum "value" of items in a knapsack, we seek to minimize $\sum_{i=1}^{n} \frac{1}{E_i}(U_i^{(max)} - U_i)^2$.

In Algorithm 1 we build a two-dimensional table *MCKES* where *MCKES[d][l]* gives the optimal solution after considering the first l tasks on d processors. We begin by assigning a score of infinity (since we are minimizing) to both the impossible case of scheduling l tasks on 0 processors (Line 1) and the trivial case of scheduling 0 task on d processors (Line 2). The *for loop* beginning on Line 3 considers scheduling tasks on d CPUs. The inner *for loop* beginning on Line 4 similarly considers the first l tasks on the d processors available. While iterating we assign each task a mode of operation, with the goal of minimizing the objective function in Equation 3. Hence we assign the *MIN* score of each task an initial score of infinity (Line 5) and consider each mode of operation j in turn (Lines 6-14). Line 7 makes sure there are enough unallocated processors to select mode j. If not, we disregard mode j. Otherwise, we consider

Algorithm 1 Multiple Choice Knapsack Elastic Scheduling (MCKES)

```
1:  MCKES[0][l] ← ∞
2:  MCKES[d][0] ← ∞
3:  for d ← 1...m do
4:      for l ← 1...n do
5:          MIN ← ∞
6:          for j ← 1...k_l do
7:              if d − m_l^(j) ≥ 0 then
8:                  if l == 1 and
                    (1/E_l)(U_l^(max) − U_l^(j))^2 < MIN then
9:                      MIN ← (1/E_l)(U_l^(max) − U_l^(j))^2
10:                 else if MCKES[d − m_l^(j)][l − 1]+
                    (1/E_l)(U_l^(max) − U_l^(j))^2 < MIN then
11:                     MIN ← MCKES[d − m_l^(j)][l − 1]+
                        (1/E_l)(U_l^(max) − U_l^(j))^2
12:                 end if
13:             end if
14:         end for
15:         MCKES[d][l] min(MIN,MCKES[d − 1][l])
16:     end for
17: end for
18: return MCKES[m][n]
```

whether selecting mode j decreases the current minimum (Line 10). If so, the new minimum value is stored (Line 11). In the special case that $l == 1$ (this is the first task scheduled), the *MIN* score simply becomes $\frac{1}{E_l}(U_l^{(max)} - U_l^{(j)})^2$ (Lines 8-9). After considering all potential modes of operation, we assign *MCKES[d][l]* the minimum of *MIN* and *MCKES[d − 1][l]* (Line 15). The final optimal value is found at *MCKES[m][n]*. One can keep track of which mode is selected at each iteration for task τ_l, and the set of modes that give the value in *MCKES[m][n]* are then assigned to their respective tasks.

Runtime complexity. Algorithm 1 has worst-case running time $\Theta(m \times N)$, where $N = \sum_{i=1}^{n} k_i$, as there are m CPUs to allocate (*for loop* beginning on Line 3) and N modes of operation selected for each value of m (*for loops* beginning on Line 4 and on Line 6).

A note about sequential tasks. As alluded to in Footnote 1, Algorithm 1 can be applied to the scheduling of sequential elastic

tasks on a uniprocessor by: (1) assigning each item associated with a candidate mode of operation, a weight equal to the corresponding utilization; and (2) assigning the knapsack a capacity equal to the desired system utilization U_d.

4 ADAPTIVE VIRTUAL REAL-TIME HYBRID SIMULATION EXPERIMENT

To evaluate our discrete elastic scheduling approach and to validate its usefulness in a real-world application, in this section we present a virtual real-time hybrid simulation (RTHS) experiment that (1) has tasks with various discrete work and period values in different modes of operation, (2) can exploit our discrete elastic scheduling approach at run-time to improve experiment accuracy by switching adaptively between modes of operation, and (3) can handle constraints like harmonic rates and discrete workloads effectively. To our knowledge, this is the first time even a virtual RTHS that can adapt its period and/or computational load has been conducted.

This simple experiment is meant as a proof of concept that discrete elastic scheduling and the adaptations thereby enabled are beneficial to real-world applications (namely RTHS). Therefore, we start with a less complicated setup than would be involved with validating a new structural component. This virtual RTHS is a tracking problem, meaning we send a displacement signal to a moving non-linear spring (henceforth referred to as the *plant*), and we attempt to make the plant follow the displacement given in the input signal as closely as possible.

The details of our experiment are shown in Figure 5. The input into the system is a recording of the displacement of a physical specimen that has been excited by forces taken from the El Centro earthquake. This is sent to an inverse compensator, which enhances tracking performance by reducing/smoothing small residual time delays introduced by the control algorithm. The controller itself uses a modified robust integrated actuator control (RIAC) strategy [19], which uses H-infinity optimization [11] to provide a trade-off between performance and robustness. The H-infinity controller uses the smoothed desired displacement passed to it from the inverse compensator and an estimate of the plant's current location to determine a command displacement to send to the plant. This estimate is the output of either a Kalman filter or a particle filter (depending on which mode of operation the task is in), both of which provide an estimate of the plant's current displacement based on noisy data (the last known measured displacement of the plant and the last commanded displacement). Each of these is calculated once per iteration and both inform the behavior of the system in the next iteration. It is assumed that when the desired displacement exceeds a certain threshold (i.e., when the plant is too far from its origin), the plant's behavior becomes more difficult to predict. Therefore, the more computationally-expensive particle filter is used then, while the Kalman filter is used otherwise.

All of the above components except the plant (which is simulated on an xPC target machine[4]) are run within a single parallel real-time task on Linux with the RT-PREEMPT patch. The relative simplicity of this experiment means that multiple tasks are not needed to accomplish the main goal of this virtual RTHS (vRTHS) experiment.

[4]MathWorks's Speedgoat/XPC Target runs in coordination with real-time Simulink. It is a widely-used platform for a variety of cyber-physical systems, including RTHS.

To gauge our approach more fully however, for scenarios where there may be different substructures of a building to simulate (at potentially different rates or detail levels) within a realistic structural validation experiment, we generate additional synthetic discrete elastic tasks to run alongside the vRTHS task, as there would be in a more complex virtual RTHS. These tasks may adapt their system resources (i.e., operate in different modes of execution) in response to the virtual RTHS task whenever it changes modes of operation from using the Kalman filter (which requires 1 processor) to the particle filter (which requires 2 processors), or vice versa. Similar to a structural validation RTHS experiment with multi-time stepping, we constrain each synthetic task to run at a rate that is harmonic with the 2048Hz rate needed by the virtual RTHS. Some of these new tasks are period-elastic, some are computationally elastic, and some are combined-elastic.

To perform this experiment, we extended the parallel (continuous) elastic concurrency platform from [17], which is available as open-source [1]. The underlying system calls, concurrency mechanisms, and synchronization techniques remain unchanged, but we replaced the original scheduling algorithm with Algorithm 1. All tasks were run on a 16 core machine with two Intel E5-2687W processors running at a constant 3092.616 MHz with Hyperthreading disabled. The RTOS used was x86-64 Linux with the RT-PREEMPT patch, and all programs were written in C++ and compiled with GNU G++ 5.2.0.

Figures 6 and 7 show the results of our adaptive virtual RTHS experiment. The solid line shows the curve of the desired plant displacement, while the dotted line shows the estimated displacement output from the particle filter or the Kalman filter. The horizontal lines mark the mode-change criterion. For any desired displacement between the lines, the estimator uses the Kalman filter. The system switches modes and uses the particle filter when the plant's desired displacement is too far from its origin, i.e., outside the lines.

Looking at Figure 6, the two curves appear nearly indistinguishable. However, when we zoom in on the peaks in Figure 7, the difference becomes visible.

As mentioned before, we ran synthetic tasks with the virtual RTHS task that adapted with its mode change, similar to how more elaborate RTHS experiments would do. Figures 8 and 9 show the workload and period of each task in the system during operation of the Kalman filter and particle filter, respectively. Note that the virtual RTHS task and Synthetic Task 3 adapt their workloads (between 488μ sec and 812 μ sec and between 2000μ sec and 5000μ sec, respectively); Synthetic Task 1 adapts its period (between 1953 μ sec and 976 μ sec), and Synthetic Task 2 adapts both its period (between 976 μ sec and 1953 μ sec) and workload (between 8000 μ sec and 5500 μ sec). Also note that there are only 3 period values used–2048Hz$\approx$ 488μ sec, 1024Hz$\approx$ 977μ sec, and 512Hz$\approx$ 1952μ sec. This is because the estimator must run at a constant 2048Hz and substructure tasks in more complicated RTHS experiments must run at harmonic rates with respect to the main feedback control loop.

A normalized root mean squared error (nRMSE) of approximately 0.5% is considered acceptable in the RTHS community. The nRMSE between the estimated and desired displacement shown in Figure 6 is 0.267%. Therefore, the virtual RTHS not only successfully transitions modes, but also performs well.

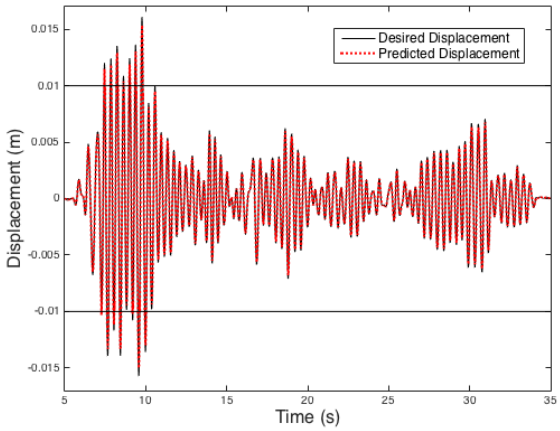

Figure 6: vRTHS Desired vs. Predicted Displacement

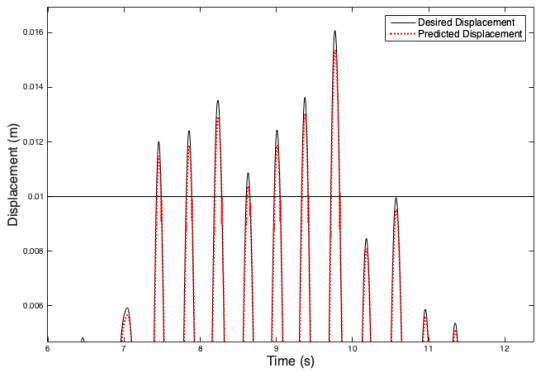

Figure 7: A Closer Look at Desired vs. Predicted Displacement

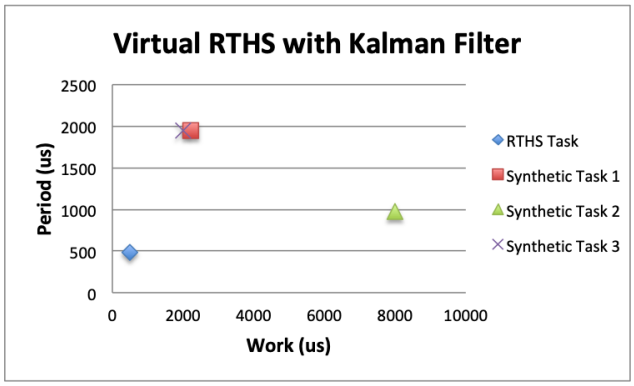

Figure 8: System Overview during Kalman Filter Execution

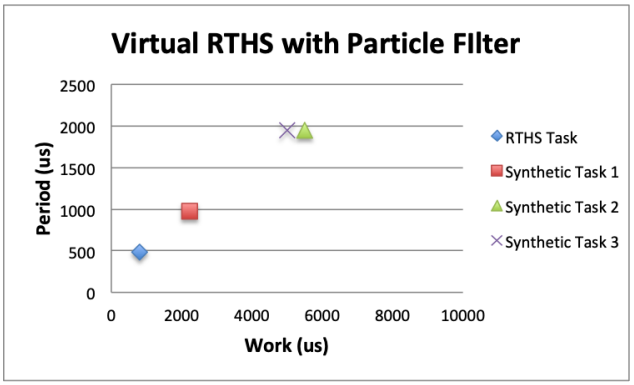

Figure 9: System Overview during Particle Filter Execution

It is important to note that the primary function of this experiment is to validate that discrete elastic scheduling can allow for the on-line adaptation of resources for parallel tasks in a real system. The tracking of a spring via an adaptive controller accomplishes this goal and lays a necessary foundation for using the discrete elastic scheduling technique for larger experimentation for structural validation via RTHS in the future.

5 EFFECTS OF TASKSET DISCRETIZATION

In this section we look at the effect that discretization of tasks' periods and workloads has on schedulability of example tasksets through loss of system-wide processor utilization compared to the continuous version. We begin by randomly generating 10,000 continuous parallel elastic tasks in the manner described in [17]. Each task is either period-elastic or computationally-elastic, and we schedule these continuous tasks according to the optimal algorithm provided in [17, 18], noting the overall system utilization and objective function value. We then create four discretized tasksets from each continuous one by assigning a discretization delta of 0.05, 0.1, 0.2, and 0.5, to each task, meaning we discretize each task in such a way that in the new tasksets, there is a candidate utilization every 5%, 10%, 20%, and 50% of the way between $U^{(min)}$ and $U^{(max)}$, plus the endpoints. For example, a period-elastic task with an $T^{(min)} = 0$ and $T^{(max)} = 100$ would be discretized to have candidate period values of 0, 20, 40, 60, 80, and 100 for $\Delta = 0.2$, and it would have candidate period values of 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 for for $\Delta = 0.1$, etc. We then schedule each of these 40,000 generated discrete elastic tasks using Algorithm 1, again noting the system utilization and objective function value.

Figures 10 through 13 show representative results. Figure 13 shows the average (and standard deviation of the) system utilization for each level of discretization. Without exception, each discretized taskset had a higher (worse) objective function value from Equation 3 than the continuous taskset from which it was derived. Typically, the objective function value increased with the discretization delta, too, as in the examples shown in Figure 10 and Figure 12. The single exception in 10,000 tasksets is depicted in Figure 11. In this case the optimal solution for the taskset obtained from $\Delta = 0.1$ occurs when each task selects the utilization value obtained from the 50th percentile. This is exactly the subset of candidate utilizations used to obtain the taskset derived from $\Delta = 0.5$ and so also gives the optimal solution for that taskset (a subset of the former).

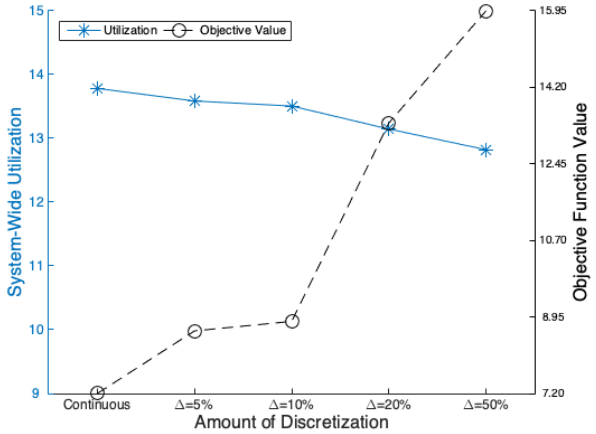

Figure 10: Taskset 1 Utilization and Objective Value

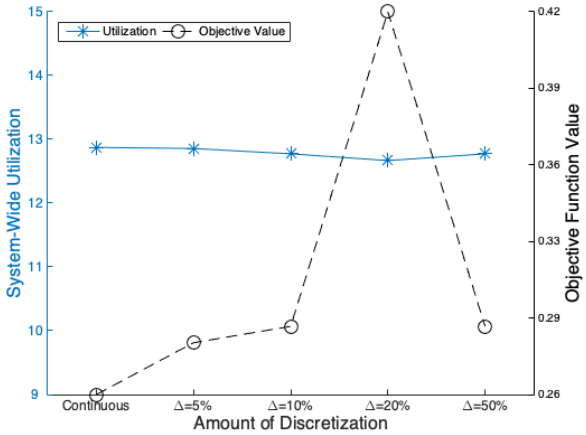

Figure 11: Taskset 2 Utilization and Objective Value

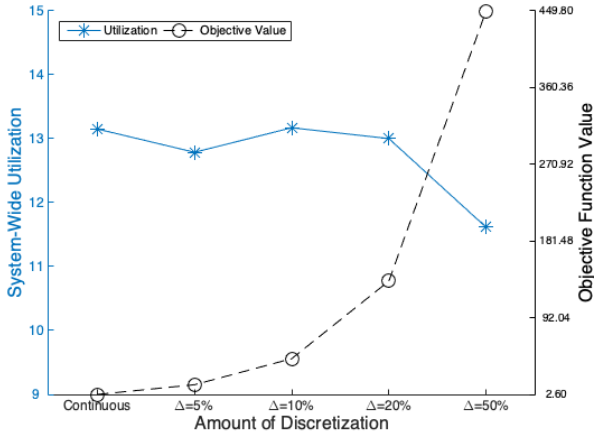

Figure 12: Taskset 3 Utilization and Objective Value

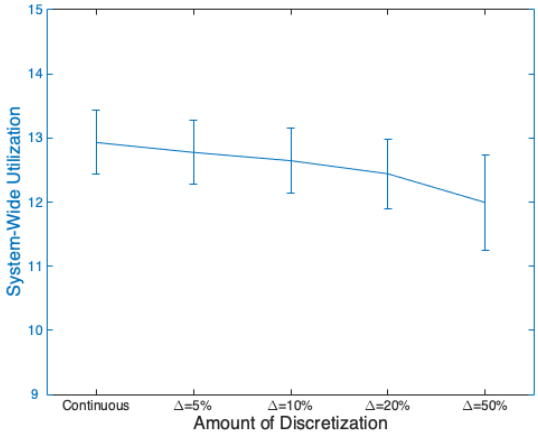

Figure 13: Average Utilization (10K Tasksets)

However, none of those selected periods are in the taskset derived from $\Delta = 0.2$ (a different subset of the former). Therefore, the objective function's value when $\Delta = 0.2$ is necessarily higher. This trend of a (typically) worsening objective function value with an increase of discretization is thus expected. We note that objective function values cannot be compared directly between tasksets as they are dependent on tasks' elastic coefficients and maximum utilizations.

For the majority of tasksets, system utilization also decreased as a taskset became more discretized, as in Figure 10. However, because we make scheduling decisions based on the objective function (weighted task utilization) rather than on system utilization, there are cases when making an inferior objective function decision increases taskset utilization: this occurred in approximately 18% of tasksets (consider Figure 12 where $\Delta = 0.1$ gives a higher system utilization than even the continuous version of the taskset).

6 CONCLUSION

In this paper, we have presented a new elastic task model with discrete sets of possible utilizations for each task. This model allows each task to modify its workload *and/or* its period when changing modes of operation, instead of adapting in only one of those dimensions. This in turn allows a wider range of parallel real-time tasks to exploit elastic scheduling techniques, and also offers a greater diversity of potential adaptations of each task, over a larger region of potential periods and workloads. It is also better aligned with task execution times on realistic hardware.

We have shown how this model can support new real-time hybrid simulations with discretely computationally-elastic, period-elastic, and combined-elastic parallel real-time tasks under the Federated Scheduling paradigm, via a pseudo-polynomial time scheduling algorithm. We used this scheduling algorithm to implement, for the first time, adaptive resource management to enable adaptive switching between controllers with different computational demands in a virtual real-time hybrid simulation (vRTHS), and examined the effects of scheduling tasks having discretized vs. continuous candidate utilizations in terms of both system utilization and objective function value.

The results presented in this paper motivate further expansion of this research as future work. Of particular interest is to begin using these tecniques for full-scale RTHS structural validation.

REFERENCES

[1] [n. d.]. Real-Time Scheduling of Parallel Tasks: Theory and Practice. http://prt.wustl.edu/

[2] Gregory B Bunting. 2016. *Parallel Real-Time Hybrid Simulation of structures using multi-scale models.* Ph.D. Dissertation. Purdue University.

[3] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. 1998. Elastic Task Model for Adaptive Rate Control. In *IEEE Real-Time Systems Symposium (RTSS).*

[4] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. 2002. Elastic Scheduling for Flexible Workload Management. *IEEE Trans. Comput.* 51, 3 (March 2002), 289–302. https://doi.org/10.1109/12.990127

[5] M. Caccamo, G. Buttazzo, and Lui Sha. 2000. Elastic feedback control. In *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000.* 121–128. https://doi.org/10.1109/EMRTS.2000.853999

[6] T. Chantem, X. Hu, and M. Lemmon. 2009. Generalized Elastic Scheduling for Real-Time Tasks. *IEEE Trans. Comput.* 58, 4 (April 2009), 480–495. https://doi.org/10.1109/TC.2008.175

[7] T. Chantem, X. S. Hu, and M. D. Lemmon. 2006. Generalized Elastic Scheduling. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06).* 236–245.

[8] Thomas Dean and Mark Boddy. 1988. An Analysis of Time-dependent Planning. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence (AAAI'88).* AAAI Press, 49–54. http://dl.acm.org/citation.cfm?id=2887965.2887974

[9] D. Ferry, G. Bunting, A. Maqhareh, A. Prakash, S. Dyke, K. Aqrawal, C. Gill, and C. Lu. 2014. Real-time system support for hybrid structural simulation. In *2014 International Conference on Embedded Software (EMSOFT).* 1–10. https://doi.org/10.1145/2656045.2656067

[10] David Ferry, Amin Maghareh, Gregory Bunting, Arun Prakash, Kunal Agrawal, Chris Gill, Chenyang Lu, and Shirley Dyke. 2014. On the performance of a highly parallelizable concurrency platform for real-time hybrid simulation. In *The Sixth World Conference on Structural Control and Monitoring.*

[11] J. William Helton. 1978. *Orbit structure of the Mobius transformation semigroup action on H-infinity (broadband matching).* 129–198.

[12] R. Karp. 1972. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher (Eds.). Plenum Press, New York, 85–103.

[13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *The Multiple-Choice Knapsack Problem.* Springer Berlin Heidelberg, Berlin, Heidelberg, 317–347. https://doi.org/10.1007/978-3-540-24777-7{_}11

[14] Tei-Wei Kuo and Aloysius K. Mok. 1991. Load Adjustment in Adaptive Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium.* 160–171.

[15] Jing Li, Abusayeed Saifullah, Kunal Agrawal, Christopher Gill, and Chenyang Lu. 2014. Analysis Of Federated And Global Scheduling For Parallel Real-Time Tasks. In *Proceedings of the 2012 26th Euromicro Conference on Real-Time Systems (ECRTS '14).* IEEE Computer Society Press, Madrid (Spain).

[16] James Orr and Sanjoy Baruah. 2019. Multiprocessor Scheduling of Elastic Tasks. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems (RTNS '19).* ACM, New York, NY, USA, 133–142. https://doi.org/10.1145/3356401.3356403

[17] J. Orr, C. Gill, K. Agrawal, S. Baruah, C. Cianfarani, P. Ang, and C. Wong. 2018. 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.* ACM Press.

[18] J. Orr, C. Gill, K. Agrawal, J. Li, and S. Baruah. [n. d.]. Elastic scheduling for parallel real-time systems. *Leibniz Transactions on Embedded Systems* ([n. d.]). https://www.cse.wustl.edu/~cdgill/publications/LITES19_ElasticParallel.pdf

[19] Ge Ou, Ali Irmak Ozdagli, Shirley J. Dyke, and Bin Wu. [n. d.]. Robust integrated actuator control: experimental verification and real-time hybrid-simulation implementation. *Earthquake Engineering & Structural Dynamics* 44, 3 ([n. d.]), 441–460. https://doi.org/10.1002/eqe.2479 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/eqe.2479

[20] P Sinha and A. A. Zoltners. 1979. The Multiple Choice Knapsack Problem. *Operations Research* 27 (1979), 503–515.

ACKNOWLEDGMENTS

This research was supported in part by NSF grants CCF-1337218 titled "XPS: FP: Real-Time Scheduling of Parallel Tasks", CNS-1136073/1136075 titled "CyberMech, a Novel Run-Time Substrate for Cyber-Mechanical Systems", and CMMI-1661621 titled "RCN: Research Network in Hybrid Simulation for Multi-Hazard Engineering".