Subtask-Level Elastic Scheduling

Marion Sudvarg, Daisy Wang, Jeremy Buhler, Chris Gill
Department of Computer Science and Engineering
Washington University in St. Louis
(msudvarg, w.yanwang, jbuhler, cdgill)@wustl.edu

Buttazzo et al.'s elastic scheduling model allows task utilizations to be "compressed" to ensure schedulability atop limited resources. Each task is assigned a range of acceptable utilizations and an "elastic constant" representing the relative adaptability of its utilization. In this paper, we consider federated scheduling, under which each high-utilization parallel task is assigned dedicated processor cores. We propose a new model of elastic workload compression for parallel DAG tasks that assigns each *subtask* its own elastic constant and continuous range of acceptable workloads. We show that the problem can be solved offline as a mixed-integer quadratic program, or online using a pseudo-polynomial dynamic programming algorithm. We also consider joint core allocation and compression of low-utilization sequential tasks and present a mixed-integer linear program for optimal elastic compression of tasks under partitioned EDF scheduling. We show empirical improvements in schedulability over the prior work and present a case study for the Fast Integrated Mobility Spectrometer (FIMS).

I. Introduction

Elastic real-time scheduling models provide a framework in which task utilizations may be reduced to guarantee schedulability despite limited resources. The original model of Buttazzo et al. [1], [2] considers uniprocessor scheduling of implicit-deadline task systems. Each task is assigned a range of allowed utilizations, as well as an additional elasticity parameter that "specifies the flexibility of the task to vary its utilization" [1]. Ideally, each task is allowed to execute at its maximum utilization. However, if this would cause the system to become overloaded, each task's utilization is "compressed" proportionally to its elastic constant until the total utilization no longer exceeds the schedulable bound of the system, or until the task reaches its minimum serviceable utilization.

The growing prevalence of multicore CPUs, even in embedded platforms, has enabled increasingly complex real-time applications to exploit intra-task parallelism. Tasks that individually require parallel execution on more than one processor to meet their deadlines are found in autonomous vehicles [3], computer vision systems [4], mobile robotics [5], hybrid structural simulation [6], [7], and satellite telescopes [8]–[10].

This has inspired extensions of the elastic framework to *federated scheduling* [11] of parallel real-time tasks, under which each high-utilization parallel task (those with $U > 1$) is allocated dedicated processor cores in sufficient number to guarantee schedulability. In prior work by Orr et al. [12], if the number of allocated cores exceeds those available in the system, parallel task utilizations are compressed by decreasing their workloads over a continuous range until the demand for processors can be met. Utilizations thus assigned satisfy a reformulation of the quadratic optimization problem presented by Chantem et al. [13], [14] that is solved by Buttazzo's original elastic scheduling model [1], [2].

Limitations of Prior Work: The proposed approach in [12] has three key limitations. First, it decreases the task's computational demand as a whole, without considering the impact on each *subtask*. The ability of each subtask to vary its utilization — and the resulting impact on quality of outcome (e.g., control performance, prediction accuracy, etc.) — should be considered individually to maximize overall quality within the resource constraints [15]–[19]. Second, it allocates processor cores per the methodology in [11], which considers each parallel task's total workload, deadline, and *span*. As it decreases task workloads, the model in [12] holds the span constant. However, span may also decrease with subtask workloads, allowing schedulability with less overall compression. Third, it only considers core allocation to high-utilization parallel tasks. In fact, under the federated scheduling model in [11], *low-utilization* tasks are scheduled concurrently on any remaining cores not allocated to the high-utilization tasks. Orr et al.'s model [12] compresses parallel tasks given a number of available cores, only suggesting as an aside that low-utilization tasks can be compressed if there are cores remaining. However, jointly compressing *all* tasks may change the numbers of cores separately allocated to high- and low-utilization tasks.

Contributions of This Work: To address these limitations of the current state of the art, we propose a model of *subtask-level* elastic scheduling. To capture the semantics of individually compressing subtask workloads, it assigns to *each* subtask an elasticity parameter and continuous range of acceptable workloads. It adapts the quadratic objective for elastic scheduling from Chantem et al. [13], [14] to this new model.

It also demonstrates solver-based methods for assigning subtask workloads. The model can be expressed and solved as a mixed-integer quadratic program (MIQP), though it is nontrivial to express the schedulability constraint as a function of subtask workloads because their assignment affects the task's span. We propose and analyze two different methods to construct the MIQP, as well as a dynamic programming algorithm for jointly compressing multiple tasks.

Finally, it jointly compresses low-utilization tasks. By considering them in aggregate, the model captures the objective value associated with compressing low-utilization tasks onto different numbers of cores, then folds these values into the dynamic program to allocate cores to both high- and low-utilization tasks. We demonstrate this concretely in the context of fluid [20] and partitioned EDF scheduling [21]. For the latter, we present a novel mixed-integer linear program (MILP) formulation for optimal elastic scheduling of sequential tasks.

Empirical Results: We implement the MIQPs and MILP in Gurobi [22], an off-the-shelf constraint-programming solver. In our evaluation, solving the MIQP for *individual* parallel tasks with up to 50 subtasks took under 42ms. By solving multiple MIQPs offline, our pseudo-polynomial algorithm enabled online core allocation and workload compression in under 60ms for 10 tasks with up to 20 subtasks each. Moreover, for the parallel tasks considered, we demonstrate that our model may achieve schedulability on systems with only 41% of the cores required by the prior model of Orr et al. [12].

We also apply our approach to a real-time atmospheric aerosol monitoring pipeline [23], [24] for the Fast Integrated Mobility Spectrometer (FIMS). We re-implement its image processing task that detects and sizes aerosol particles to parallelize over multiple image segments. We assign elastic constants based on average particle densities within each segment. This allows us to compress the subtask workloads to remain schedulable even on future drone-based deployments where reduced particle resident times require shorter periods.

II. Background

A. Uniprocessor, Implicit-Deadline Elastic Scheduling

Buttazzo's elastic recurrent real-time workload model [1], [2] provides a framework for managing overload by reducing ("compressing") the utilizations of individual tasks until the total no longer exceeds the schedulable bound. It characterizes each task $\tau_i{=}(C_i, U_i^{\min}, U_i^{\max}, U_i, E_i)$ by five non-negative parameters: C_i is the task's worst-case execution time; $U_i^{\max}$ is its maximum utilization, i.e., its nominal value when executing at the desired service level in an uncompressed state; $U_i^{\min}$ is its minimum utilization, i.e., a bound on the amount its service can degrade; U_i is the task's assigned utilization, constrained to $U_i^{\min} \leq U_i \leq U_i^{\max}$; and E_i is an elastic constant, representing "the flexibility of the task to vary its utilization" [1].

A task system $\Gamma = \{\tau_1, \ldots, \tau_n\}$ has a total uncompressed utilization $U_{\text{SUM}}^{\max} = \sum_{i=1}^{n} U_i^{\max}$ and a desired utilization U_D representing the utilization bound given by the scheduling algorithm in use. In the event of system overload, i.e., if $U_{\text{SUM}}^{\max} > U_D$, the elastic model compresses each task's utilization such that it is reduced from its desired maximum proportionally to the task's elasticity parameter, subject to the constraint that it remains no less than the specified minimum. Compression is realized by adjusting each task's period T_i according to its new utilization, i.e., $T_i = C_i/U_i$.

Chantem et al. [13], [14] demonstrated that utilizations thus assigned satisfy the following quadratic optimization problem:

$$\min_{U_i} \quad \sum_{i=1}^{n} \frac{1}{E_i}(U_i^{\max} - U_i)^2 \tag{1a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} U_i \leq U_D \tag{1b}$$

$$\forall_i, \quad U_i^{\min} \leq U_i \leq U_i^{\max}. \tag{1c}$$

This supports elasticity in other task models with schedulability tests that do not rely strictly on a utilization bound, including *federated scheduling* of parallel real-time tasks [11].

B. Elastic Frameworks for Federated Scheduling

The federated scheduling model of Li et al. [11] deals with systems of *parallel* implicit-deadline tasks. Each task τ_i consists of a set of subtasks $\tau_{i,j}$, each characterized by a workload $c_{i,j}$ representing its worst-case execution time. Subtasks may run in parallel, except as constrained by a precedence relation: if $\tau_{i,a} \prec \tau_{i,b}$, then $\tau_{i,a}$ must fully complete its execution before $\tau_{i,b}$ is scheduled. The partial-ordering of precedence over subtask execution that describes task execution gives rise to a standard *directed acyclic graph (DAG)* representation with a collection of vertices $v_{i,j}$ corresponding to subtasks $\tau_{i,j}$. A directed edge from vertex $v_{i,a}$ to $v_{i,b}$ exists if and only if $\tau_{i,a} \prec \tau_{i,b}$ and there is no $\tau_{i,c}$ for which $\tau_{i,a} \prec \tau_{i,c} \prec \tau_{i,b}$, i.e., $\tau_{i,b}$ directly succeeds $\tau_{i,a}$.

Each high-utilization parallel task τ_i is allocated m_i dedicated processor cores, where

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

Here, $C_i = \sum_j c_{i,j}$ represents the task's total workload (DAG volume). L_i is the task's *span*, i.e., the DAG's critical path length (weighted by subtask execution time). D_i is the task's deadline; for implicit-deadline tasks, this equals the period T_i.

In [25], Orr et al. extended the elastic framework to the federated scheduling model. If the total processor cores allocated exceed the number available, each high-utilization parallel task has its utilization compressed until demand is met. Rather than a simple utilization bound, Eqn. 2 implies the following schedulability condition:

$$\sum_{i=1}^{n} \left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil \leq m. \tag{3}$$

where m is the total number of processor cores available. Task utilizations are assigned according to Eqn. 1, with the original schedulability condition (Eqn. 1b) replaced by Eqn. 3.

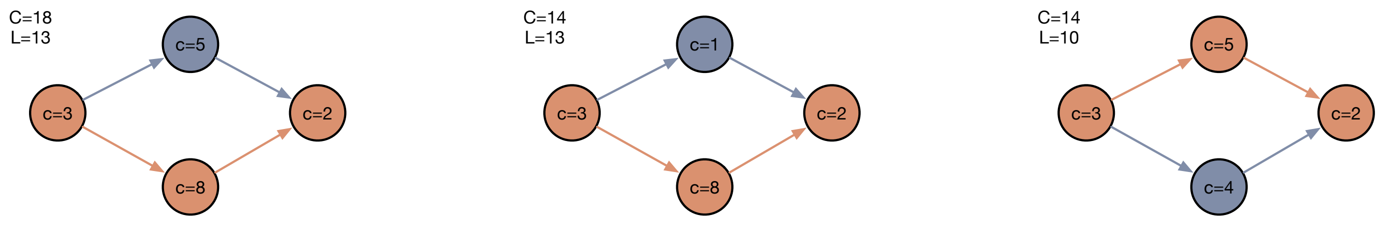
In [12], Orr et al. extended their approach to *computationally-elastic* tasks, allowing parallel workloads to be adjusted over a continuous range: a task with period T_i would have its workload assigned as $C_i = T_i \cdot U_i$. This may be realized, for example, by reducing the quantity of input data to process or by forcing an iterative anytime algorithm to terminate early [26]. The span L_i is held constant.

This work addresses limitations of that model. In particular, we consider how reducing the workload of each individual *subtask* impacts result quality and task span. We also consider the joint compression of low-utilization tasks. The next section details these limitations and motivates our work.

III. Motivation and Problem Statements

A. Subtask-Level Elastic Scheduling

Subtask-Level Workload Compression: In a computationally-elastic task, the workloads of individual *subtasks* may be



(a) Uncompressed workload **(b)** No change to span **(c)** Span compressed

Fig. 1: Critical path may change depending on which subtask workloads are compressed.

able to adapt in different ways. Examples include anytime workloads, which may iteratively refine the result to achieve greater precision. If their execution time budget is exhausted, the current result is used. Others support discrete execution modes that can be selected prior to execution, which may correspond to different algorithmic techniques [19] or varying degrees of numerical precision [17], [27], [28]. At fine enough granularity, discrete modes can be approximated as a continuous state space, e.g., the proportion of input data selected from a large set for processing [19]. Furthermore, as we demonstrate with our case study in §VIII, a subtask may represent some optional execution to improve the result. If the subtask's execution times form a wide distribution, e.g., due to dependence on the number of features present in an image, then selecting an execution time from a continuous range increases the likelihood that it will be able to complete.

Moreover, the workload assigned to each individual subtask may uniquely impact result quality. For example, in autonomous vehicles, AutoE2E [18], [29] adjusts end-to-end task execution to maintain schedulability in open and unpredictable environments. It considers the relative importance of each subtask, both from the perspective of driver preference and control outcome. For LiDAR object detection, fine-grained time and accuracy tradeoffs in the PointPillars [30] encoder pipeline are exploited in [17] to enable adaptive execution in response to dynamic deadlines in an open environment. The authors analyze the execution time and corresponding accuracy associated with different levels of computational precision in the DAG's subtasks. For prompt gamma-ray burst localization, the authors of [19] model the localization pipeline as a highly-parallel fork-join task, then parameterize its workloads along continuous degrees of freedom, characterizing the impact on localization accuracy of compressing each pipeline stage.

Each of these applications enables adaptive real-time execution based on the importance of each subtask. However, there is as yet no model that extends elastic scheduling to consider individual subtasks for parallel DAG tasks in general.

Considering Task Span: The model in [12] for federated scheduling of computationally-elastic tasks holds the span L_i of each task τ_i constant while compressing workloads C_i. Depending on *how* the new workload assignment C_i is to be realized, i.e., which *subtask* workloads $c_{i,j}$ are to be reduced, the value L_i may also decrease, as Fig. 1 illustrates. Without accounting for this, the model may be pessimistic in resource allocation and may over-compress task workloads.

Example 1. *Consider a task with parameters $C_i^{\max} = 10$, $L_i = 4$, and $D_i = 6$ to be scheduled on only 2 processor cores. If L_i is held constant, the task's workload would have to be decreased to $C_i = 8$ to satisfy Eqn. 2. But the workload needs to only be reduced by 1 unit along its critical path ($C_i = 9$ and $L_i = 3$) to be schedulable.*

The Subtask-Level Elastic Workload Model: To fill the above-mentioned gaps, we propose a model of subtask-level computational-elasticity where *each* subtask $\tau_{i,j}$ is assigned a continuous range of allowed execution times $[c_{i,j}^{\min}, c_{i,j}^{\max}]$ and an elastic constant $E_{i,j}$. This elasticity parameter, similarly to the model of Buttazzo et al. [1], [2], represents the adaptability of the subtask's workload, e.g., based on its relative importance to result outcome (a *more important* subtask would be *less elastic*). Subtask workloads $c_{i,j}$ are then selected to satisfy a modified version of the quadratic optimization of Chantem et al. [13], [14] in Eqn. 1 so as to **(4a)** minimize the (weighted) deviations of individual *subtask* utilizations from their desired values, within **(4b)** the schedulability constraints that arise from assigning each parallel task its own dedicated processor cores according to Eqn. 2, and **(4c)** constraints on the range of allowed execution times for each subtask.

$$\min_{\{c_{i,j}\}} \quad \sum_{\tau_{i,j}} \frac{1}{E_{i,j}T_i^2}\left(c_{i,j}^{\max} - c_{i,j}\right)^2 \tag{4a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i(\{c_{i,j}\})}{T_i - L_i(\{c_{i,j}\})} \right\rceil \le m \tag{4b}$$

$$\forall_{i,j}, \quad c_{i,j}^{\min} \le c_{i,j} \le c_{i,j}^{\max}. \tag{4c}$$

B. Joint Compression of Low-Utilization Tasks

Motivation: Prior models for elastic scheduling of parallel tasks in [12], [25], [27] do not address compression of low-utilization (i.e., sequential, non-DAG) elastic tasks. They assume a fixed allocation of processor cores to high-utilization parallel tasks, mentioning as an aside that sequential tasks can be compressed per Buttazzo's original algorithm in [1], [2] to be schedule on any remaining processors. However, the semantics of elastic scheduling suggest that, as the allocation of cores to each task may change, so too may the allocation of cores between high- and low-utilization tasks.

Example 2. *Consider a system with $m = 4$ processor cores on which the following implicit-deadline tasks must be scheduled:*

1) $\tau_1 = (C_1 = 5, T_1 = 10)$, *a sequential task.*
2) $\tau_2 = (C_2 = 3, T_2 = 8)$, *a sequential task.*
3) $\tau_3 = (C_3 = 4, T_3 = 7)$, *a sequential task.*
4) $\tau_4 = (C_4 = 30, L_4 = 10, T_4 = 15)$, *a parallel task.*

The total utilization of the sequential tasks is $\sim$1.45, *and they therefore require* 2 *cores, but task* τ_4 *alone requires* $\left\lceil \frac{30-10}{15-10} \right\rceil = 4$ *cores per Eqn. 2. If we compress its workload by* 5 *units along its span,* τ_4 *is schedulable on the remaining* 2 *cores:* $\left\lceil \frac{25-5}{15-5} \right\rceil = 2$. *However, if we compress the utilizations of the sequential tasks so that they occupy a single core, then the workload and span of* τ_4 *need only be to reduced by* $5/3$.

In §VI of this paper, we present methods for joint compression of both high- and low-utilization tasks that allow dynamic allocation of processor cores to either set.

IV. MIQP for Subtask-Level Elasticity

The optimization problem in Eqn. 4 is naturally expressed as a mixed-integer quadratic program (MIQP), allowing it to be solved using one of many available off-the-shelf solvers.

A. Constructing the MIQP

We demonstrate an approach to constructing the problem for Gurobi [22], a mathematical optimization tool that solves MIQPs. It supports both integer and continuous variables, and constraints and objectives can be linear or quadratic expressions. Though we have chosen to focus on Gurobi, this approach generalizes to other quadratic solvers.

1) Subtask Workloads: For each subtask $\tau_{i,j}$, define continuous variables $c_{i,j}$ representing the workload assigned to the subtask, and constrained as in Eqn. 4c:

$$c_{i,j}^{\min} \leq c_{i,j} \leq c_{i,j}^{\max}. \tag{5}$$

2) Objective: Our goal is to find an assignment of values to each variable $c_{i,j}$ that minimizes Eqn. 4a. Because our objective is to *minimize* this expression, and not to solve it directly, we can simplify by removing constant terms. Thus, our MIQP can be constructed so as to:

$$\textbf{minimize} \sum_{\tau_{i,j}} \left(\frac{1}{E_{i,j} T_i^2} \cdot c_{i,j}^2 - \frac{2 c_{i,j}^{\max}}{E_{i,j} T_i^2} \cdot c_{i,j} \right). \tag{6}$$

3) Span: For each task τ_i, define a non-negative continuous variable L_i representing its span. It is required that L_i does not exceed T_i for τ_i to be schedulable, as $D_i = T_i$. Furthermore, a value of L_i exceeding T_i might result in the LHS of Eqn. 3 taking a negative value, which would be an inconsistent interpretation of the condition, resulting in an invalid solution. To enforce this, we add constraints of the form

$$L_i \leq T_i. \tag{7}$$

Other variables and constraints to enforce the intended interpretation of each variable L_i as the span of τ_i are discussed further in §IV-B and §IV-C.

4) Processor Core Allocations: For each task τ_i, define a non-negative **integer** variable m_i representing the number of cores allocated to the task, which should be sufficient to guarantee schedulability according to Eqn. 2. To enforce this intended interpretation, we add constraints of the form

$$m_i \geq \frac{\sum_j c_{i,j} - L_i}{T_i - L_i}.$$

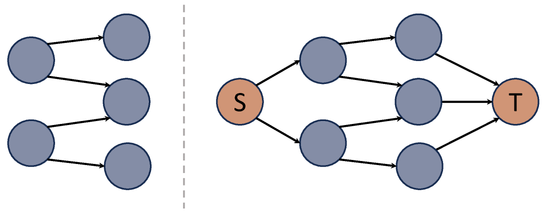


Fig. 2: Left: a DAG with two source vertices and three sink vertices. **Right:** unique source and sink vertices are added; if these both have 0 workload, the corresponding task's execution is unchanged.

Since m_i is specified to be an integer variable, it will respect the ceiling operator that appears in Eqn. 2. Rearranging, this yields quadratic constraints of the form:

$$L_i + T_i \cdot m_i \geq m_i \cdot L_i + \sum_j c_{i,j}. \tag{8}$$

With the above constraint on span (Eqn. 7), this will force L_i to remain strictly less than T_i, since $m_i \to \infty$ as $L_i \to T_i^-$.

5) Total Processor Cores: The total allocation of cores must not exceed m, the number available. To enforce this, we add the additional constraint

$$\sum_i m_i \leq m. \tag{9}$$

B. Task Span: A Constraint for Each Path

We now return to the problem of representing a DAG task's span in our MIQP. For a task τ_i, the variable L_i represents its span, which can be expressed as the maximum workload

$$L_i = \max_{p_{i,k}} \left\{ \sum_{v_{i,j} \in p_{i,k}} c_{i,j} \right\}$$

over the set of paths $\{p_{i,k}\}$ between pairs of vertices in the task's representative DAG. To simplify this, we consider tasks with DAGs having a single source vertex s and sink vertex t. Any task DAG , even those that are not weakly-connected, can be represented as a weakly-connected DAG with a single source and sink with the following construction. ① Add a vertex $v_{i,s}$ with execution time $c_{i,s} = 0$ and connect it with edges to all vertices in the DAG that do not already have incoming edges. Similarly, ② add a 0-workload vertex $v_{i,t}$, connected with edges from all vertices that do not already have outgoing edges. This is illustrated in Fig. 2.

Because of the restriction that $v_{i,a}$ is connected by an edge to $v_{i,b}$ only if $\tau_{i,b}$ directly succeeds $\tau_{i,a}$, every path from s to t might form the critical path, depending on the assignment of subtask execution times. Therefore, for each path $p_{i,k}$ from s to t, we add constraints of the form

$$L_i \geq \sum_{v_{i,j} \in p_{i,k}} c_{i,j}. \tag{10}$$

Number of Constraints: The inequality in Eqn. 10 represents a constraint for every path from each task DAG's source vertex to its sink. Therefore the number of these constraints can be expressed as the number of *maximal paths* through the DAG.

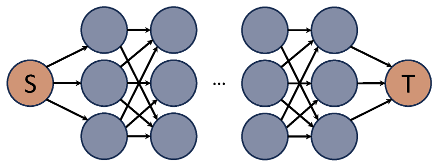


Fig. 3: A DAG with n vertices and $3^{(n-2)/3}$ maximal paths, each of which might be the critical path.

It is shown in [31] that for a DAG without shortcuts[1] having $n = 3k$ vertices for $k \in \mathbb{N}$, the maximum number of maximal paths is 3^k. For our construction that adds a unique source and sink vertex, there thus can be up to $3^{(n-2)/3}$ constraints for the DAG shown in Fig. 3. We analyze the number of these constraints for randomly-generated task DAGs in §VII-A.

C. Task Span: A Polynomial Number of Constraints

To provide a smaller bound on the number of constraints in our MIQP, we propose an alternative method for enforcing the intended interpretation of the L_i variables. Though giving rise to a different representation of the problem, this method still semantically captures the optimization problem (4). To do so, we introduce a term $l_{i,j}$ representing the *subtask span* of $\tau_{i,j}$.

Definition 1 (Subtask Span [32])**.** *The span $l_{i,j}$ of subtask $\tau_{i,j}$ in task τ_i represents the length of the longest path — weighted by the execution time of each subtask along the path — originating at the corresponding vertex $v_{i,j}$ of the task's DAG representation and including $v_{i,j}$ itself. This can be expressed by the following recurrence:*

$$l_{i,j} = c_{i,j} + \max_{v_{i,k}:\ v_{i,j} \prec v_{i,k}} \{l_{i,k}\} \tag{11}$$

For a task τ_i with a single source vertex s, this implies that $L_i = l_{i,s}$. From the above recurrence in Eqn. 11, we can enforce the intended interpretation of each span variable L_i by the following construction. For each subtask $\tau_{i,j}$ that does not correspond to a sink vertex in the task DAG, add a variable $l_{i,j}$ representing its span. The span of the source vertex s is already represented by the variable L_i. Then for each such variable $l_{i,j}$, add the following constraint for every non-sink subtask $\tau_{i,k}$ that directly succeeds $\tau_{i,j}$:

$$l_{i,j} \geq c_{i,j} + l_{i,k}. \tag{12}$$

For every subtask $\tau_{i,k}$ that directly succeeds $\tau_{i,j}$ and that *does* correspond to a sink vertex in the task DAG, add the following constraint instead:

$$l_{i,j} \geq c_{i,j} + c_{i,k}. \tag{13}$$

Number of Constraints: With this method, our MIQP has an additional linear constraint with 3 terms for every edge of the DAG. The maximum number of edges for a DAG with k vertices is $\left\lfloor \frac{k^2}{4} \right\rfloor$, which follows from Turán's theorem [33].

[1]An edge is a shortcut if the vertices it connects are connected by an alternate path. Task DAGs have no shortcuts, since vertices are only connected by an edge if one directly succeeds the other.

V. Joint Compression with Dynamic Programming

We now present an alternative approach to the problem of workload compression using an MIQP. Rather than constructing a joint problem over all tasks, the idea is to construct an MIQP for each task *individually*, then solve to find the optimal assignment of subtask workloads (and the corresponding objective value) for each possible core allocation. This defines a set of discrete states for each task corresponding to different core allocations; the optimal assignment *overall* can then be determined using dynamic programming.

A. Motivation

Compressing task workloads individually, then solving the joint problem with dynamic programming (DP) has three advantages over the single joint MIQP presented in §IV.

Faster Solution Search: Though we cannot make theoretical guarantees about improved complexity, we demonstrate empirically in §VII-C that the time to solve a single joint MIQP increases rapidly with the number of tasks. Solving multiple MIQPs for each individual task, then constructing a dynamic program to allocate cores optimally to all tasks, may be faster.

Efficient Admission Control: Buttazzo's elastic scheduling model in [1], [2] is not simply intended for adjusting a predefined set of tasks to be schedulable on a resource-constrained system. Its primary use-case is in dynamic and open systems where the set of active tasks may change, and therefore an efficient approach to admission control is desirable.

Given that task parameters (control-flow DAGs, execution times, deadlines, etc.) are assumed to be characterized offline, it is also reasonable that discrete states corresponding to optimal subtask-level workload compression for different core allocations also could be computed offline. Then, when configuring the system, or during admission control, only the pseudo-polynomial DP problem needs to be solved.

Joint Compression of Low-Utilization Tasks: Finally, as we show in the next section, a DP-based approach also allows us to address low-utilization sequential tasks to which cores must be allocated concurrently with high-utilization parallel tasks.

B. Method

Our DP-based approach is realised in two steps: ① Construct and solve an MIQP for each task individually over every possible core allocation. Then ② construct an instance of a multiple-choice knapsack problem to allocate cores to each task such that **(a)** the objective in Eqn. 4a is minimized while **(b)** the total allocation of cores does not exceed the number available. This approach is outlined in Alg. 1, which takes a set Γ of n tasks to be scheduled on m processor cores.

1) Constructing and Solving MIQPs: For each individual task τ_i, we compute the minimum $m_i^{\min}$ and maximum $m_i^{\max}$ number of cores that it can be allocated. For any allocation less than the minimum, τ_i is not guaranteed to be schedulable; any allocation greater than the maximum is wasted capacity. Then for each possible core allocation m^* in the range $[m_i^{\min}, m_i^{\max}-1]$, we construct and solve an MIQP according to the procedure in §IV for just the *individual* task.

The MIQP may be simplified by removing the variable m_i that represents the number of cores assigned to task τ_i and replacing it instead with a constant $m = m^*$. In doing so, the constraint taking the form of Eqn. 8 becomes linear instead of quadratic, and the constraint of Eqn. 9 is removed.

Algorithm 1: COMPRESS-DP(Γ,m)

```
1  Input: A set $\Gamma$ of $n$ high-utilization parallel tasks, $m$ available
     processor cores
2  Output: A set $\{c_{i,j}\}$ of subtask workload assignments
3  ▷ Find optimal state for each core allocation
4  forall $\tau_i \in \Gamma$ do
5      $C_i^{\min} \leftarrow \sum_j c_{i,j}^{\min}$, $C_i^{\max} \leftarrow \sum_j c_{i,j}^{\max}$
6      $L_i^{\min} \leftarrow$ Compute span according to $c_{i,j}^{\min}$ values
7      $L_i^{\max} \leftarrow$ Compute span according to $c_{i,j}^{\max}$ values
8      $m_i^{\min} \leftarrow \left\lceil \frac{C_i^{\min} - L_i^{\min}}{T_i - L_i^{\min}} \right\rceil$, $m_i^{\max} \leftarrow \left\lceil \frac{C_i^{\max} - L_i^{\max}}{T_i - L_i^{\max}} \right\rceil$
9      forall $m_{i,k} \leftarrow m_i^{\min}..(m_i^{\max}-1)$ do
10         Construct and solve an MIQP to obtain optimal subtask
             workloads and corresponding objective value $O_{i,k}$ to
             compress the *single* task $\tau_i$ to execute on $m_{i,k}$ cores.

11 ▷ Find optimal joint state for $m$ cores
12 if $\sum_i m_i^{\max} \leq m$ then return *No compression needed*
13 if $\sum_i m_i^{\min} > m$ then return *Not schedulable*
14 ▷ Adapted multiple-choice knapsack
15 $DP[0..m][0..n]$▷ Table to track optimal solution.
16 $DP[0][*].O \leftarrow \infty$, $DP[*][0].O \leftarrow \infty$
17 for $m^* \leftarrow 1..m$ do
18     for $i \leftarrow 1..n$ do
19         MIN $\leftarrow \infty$ ▷ Minimum objective so far.
20         ALLOC $\leftarrow -1$ ▷ Core allocation to $\tau_i$.
21         ▷ Possible compression states for $\tau_i$,
             corresponding to valid core assignments
22         for $m_{i,k} \leftarrow m_i^{\min}..\min(m_i^{\max}, m^*)$ do
23             if $i = 1$ then
24                 ▷ First task, allocate cores.
25                 MIN $\leftarrow O_{i,k}$
26                 ALLOC $\leftarrow m_{i,k}$
27             else if $DP[m^* - m_{i,k}][i-1].O + O_{i,k} <$ MIN then
28                 ▷ Re-assigning $m_{i,k}$ cores to $\tau_i$
                     reduces objective function.
29                 MIN $\leftarrow DP[m^* - m_{i,k}][i-1].O + O_{i,k}$
30                 ALLOC $\leftarrow m_{i,k}$

31         if ALLOC $> -1$ then
32             ▷ Update based on re-allocation.
33             $DP[m^*][i].M \leftarrow DP[m^* -$ ALLOC$][i-1].M$
34             Insert ALLOC into $DP[m^*][i].M$
35             $DP[m^*][i].O =$ MIN
36         else
37             ▷ Otherwise, use previous allocation.
38             $DP[m^*][i] = DP[m^* - 1][i]$

39 return $DP[m][n]$
```

Solving for each value of m^* in this way gives us a set of optimal subtask workload assignments and objective function values for each allocation; for $m^* = m_i^{\max}$, every subtask workload is assigned as $c_{i,j} = c_{i,j}^{\max}$ and the task's contribution to the objective function in Eqn. 4a is 0.

2) Joint Allocation as a Multiple-Choice Knapsack Problem: Lines 4–10 of Alg. 1 give us, for each task τ_i, a group of pairs of *weight* (processor core allocation, $m_{i,k}$) and *cost* (the minimum value taken by Eqn. 4a, $O_{i,k}$) values for each $m_{i,k} \in [m_i^{\min}, m_i^{\max}]$. The goal is to select a pair from each group to minimize total cost, while preventing the total weight from exceeding the number of available cores m.

The above problem is similar to *multiple-choice knapsack*, in which is given a set of disjoint groups of items with weights and profits, and the problem is to select exactly one item from each group to maximize total profit without exceeding a given total weight bound. It is shown in [27] that the pseudo-polynomial DP-based algorithm presented in [34] for multiple-choice knapsack can be adapted instead to *minimize* total item *cost*, and is therefore applicable to our allocation problem.

Our implementation of this algorithm builds a two-dimensional table DP where $DP[m^*][i]$ gives the optimal solution after considering the first $i{\leq}n$ tasks on $m^*{\leq}m$ cores. Each entry in the table is a pair $\langle M, O\rangle$ where M is a set that tracks the number of cores allocated to those i tasks, and O is the corresponding minimum objective function value. Entries satisfy the following recurrence:

$$DP[m^*][i].O = \min\bigl(DP[m^*-1][i].O, \\ \min_k\{DP[m^* - m_{i,k}][i-1].O + O_{i,k}\}\bigr)$$

The first term is the entry corresponding to assigning the first i tasks on m^*-1 cores, i.e., the case where adding an additional core does not decrease the cost. The second term represents the minimum cumulative cost of assigning $m_{i,k}$ cores to task τ_i and the remaining $m^* - m_{i,k}$ cores to the previous tasks.

Lines 14–38 of Alg. 1 use dynamic programming to iteratively construct the table so $DP[m^*-1][i]$ and all entries $DP[m^*-m_{i,k}][i-1]$ are already populated when $DP[m^*][i]$ is computed. The procedure iterates over cores, then tasks, considering scheduling the first i tasks on m^* CPUs. For each possible assignment of cores $m_{i,k} \leq m^*$ to τ_i (bounded by the minimum and maximum core assignments $m_i^{\min}$ and $m_i^{\max}$ due to the constraints on the subtask workloads of τ_i), the algorithm checks whether re-allocating $m_{i,k}$ cores to task τ_i improves the result (i.e., decreases the tracking variable MIN). If an improved allocation is found, then the entry of the DP table is updated with the objective (cost) and corresponding core allocation. Otherwise, it is updated to match the best allocation over the previously-considered $m^* - 1$ cores.

Runtime Complexity and Admission Control: While we cannot make guarantees about the time to solve each MIQP, the DP portion of Alg. 1 is pseudopolynomial in n and m. There are m CPUs to allocate (line 17) to n tasks (line 18). For each task τ_i, we consider allocations from $m_i^{\min}$ to $m_i^{\max}$, stopping if the currently-considered allocation m^* is reached (line 19); this bounds the number of iterations of the inner **for** loop to m, since $m^*{\leq}m$. The total worst-case running time is therefore $\boldsymbol{\mathcal{O}(n{\cdot}m^2)}$. As justified in §V-A, if the optimal set of task workloads for each core allocation is obtained *offline* when a task's other parameters are characterized, then admission of a new task to an already-compressed system can be achieved by executing lines 14–32 of the algorithm, enabling admission control in pseudo-polynomial time. We evaluate this in the context of synthetically-generated parallel tasks in §VII-C.

VI. Jointly Compressing Low-Utilization Tasks

Our DP-based approach of the previous section also enables joint scheduling of low-utilization sequential tasks. We characterize such tasks according to Buttazzo's original elastic scheduling model, with utilizations compressed proportionally to their elastic constants [1], [2]. The key idea is that we can consider $m_{\text{LO}}^{\min}$ and $m_{\text{LO}}^{\max}$ as the number of cores necessary to schedule the complete set Γ_{LO} of low-utilization tasks when fully compressed and uncompressed, respectively. For every $m^* \in [m_{\text{LO}}^{\min}, m_{\text{LO}}^{\max} - 1]$, we can quantify the amount of compression necessary to achieve schedulability on m^* cores. By then solving for the corresponding objective function value in Eqn. 4a for the compressed Γ_{LO}, we obtain a set of discrete core assignments and costs. This allows the complete set Γ_{LO} to be integrated into the DP-based algorithm as if it were a single high-utilization parallel task.

Obtaining values $m_{\text{LO}}^{\min}$ and $m_{\text{LO}}^{\max}$ — and the amount of compression needed to schedule on m^* cores — depends on the scheduling algorithm. Though complete coverage of multiprocessor scheduling is out of scope, we outline approaches for fluid and partitioned EDF scheduling, extending the approaches in [35], [36] for multiprocessor elastic scheduling.

A. Fluid Scheduling

Under fluid scheduling, each individual task τ_i is assigned a fraction f_i of a processor at each instant in time. This is a convenient abstraction that considers a task set Γ to be schedulable on m cores so long as **(a)** the total utilization $\sum_i U_i$ of Γ does not exceed m, and **(b)** the individual utilization U_i of each task τ_i does not exceed 1 [20].

For low-utilization tasks, condition **(b)** is automatically satisfied. We can therefore obtain $m_{\text{LO}}^{\min}$ and $m_{\text{LO}}^{\max}$ as

$$m_{\text{LO}}^{\min} = \left\lceil \sum_i U_i^{\min} \right\rceil \quad m_{\text{LO}}^{\max} = \left\lceil \sum_i U_i^{\max} \right\rceil, \tag{14}$$

where $U_i^{\min} = C_i^{\min}/T_i$ or $C_i/T_i^{\max}$ (and similarly for $U_i^{\max}$), depending on whether τ_i is computationally-elastic or rate-elastic. Then for $m^* \in [m_{\text{LO}}^{\min}, m_{\text{LO}}^{\max} - 1]$, we assign values U_i to each task τ_i that satisfy Buttazzo's elastic model [1], [2] with the desired utilization U_D equal to m^*.

The total time — beyond the $\mathcal{O}((n_{\text{HI}}+1)\cdot m^2)$ to solve the DP problem jointly with n_{HI} high-utilization parallel tasks — can be kept to a minimum by using the algorithm of Sudvarg et al. in [36], [37]. Computing $m_{\text{LO}}^{\min}$ and $m_{\text{LO}}^{\max}$ can be done in time linear in n_{LO}, the number of tasks in Γ_{LO}. Compressing to m^* cores can be done in time $\mathcal{O}(n_{\text{LO}} \cdot \log(n_{\text{LO}}))$. However, the quasilinear time is due to an initial step of obtaining a sorted list of tasks; the sort order does not depend on the desired utilization U_D. Therefore, this only needs to be done once. The remainder of [37, Alg. 1] takes time $\mathcal{O}(n_{\text{LO}})$. Thus, the worst-case running time is

$$\boldsymbol{\mathcal{O}(n_{\text{LO}} \cdot \log(n_{\text{LO}}) + m \cdot n_{\text{LO}})}$$

which accounts for the initial sort, followed by at most m linear-time invocations of [37, Alg. 1] and computations of Eqn. 4a, since we can stop when m^* exceeds m.

B. Partitioned EDF Scheduling

While fluid scheduling is a convenient abstraction, and implementations exist to approximate it [38], it often remains impractical in real systems [35]. A more applicable paradigm is partitioned scheduling, where tasks are distributed to processors a priori, then scheduled with other tasks on that processor according to a common approach (e.g., fixed-priority or EDF). An optimal distribution for partitioned EDF is equivalent to bin-packing, and is therefore NP-hard in the strong sense, but approximation algorithms exist that provide guaranteed schedulability if a utilization bound is not exceeded [21]. For example, a set of low-utilization tasks is EDF-schedulable on m^* cores using first-fit or best-fit partitioning if their total utilization does not exceed $(m^*+1)/2$. It is therefore straightforward to adapt the method proposed for fluid scheduling to partitioned EDF by changing the utilization bound.

However, this bound tends to be pessimistic [36]. An alternative heuristic-based method is explored in [35], [36]. We instead propose an exact solution based on a mixed-integer linear program (MILP). The MILP is solved for each possible core assignment $m^* \in [m_{\text{LO}}^{\min}, m_{\text{LO}}^{\max} - 1]$, where $m_{\text{LO}}^{\min}$ is computed as per fluid scheduling (representing the absolute lower bound on the number of cores for which a feasible partition might be found under maximum compression) and from the worst-case utilization bound, $m_{\text{LO}}^{\max} = \lceil 2 \sum_i (U_i^{\max}) - 1 \rceil$.

Constructing the MILP: From [39], there is a value λ representing the *amount* of compression applied to the task system. Since every sequential task is compressed proportionally to its elasticity, we can express each task's utilization U_i as

$$U_i(\lambda) = \max\left(U_i^{\max} - \lambda E_i, U_i^{\min}\right). \tag{15}$$

The goal, then, is to find the minimum value of λ for which the set Γ_{LO} of low-utilization tasks is schedulable on m cores.

1) Task Utilizations: For each low-utilization task τ_i, define a continuous variable U_i representing its utilization:

$$U_i^{\min} \leq U_i \leq U_i^{\max}. \tag{16}$$

2) Compression: We define a real-valued variable λ interpreted as above. To enforce this, we specify the constraint:

$$0 \leq \lambda \leq \lambda^{\max}. \tag{17}$$

Here, $\lambda^{\max}$ represents the value of λ for which every task's utilization U_i reaches its minimum $U_i^{\min}$. This is the maximum compression that may be applied, after which utilizations no longer change. From Eqn. 15, this can be computed as:

$$\lambda^{\max} = \max_{\tau_i \in \Gamma_{\text{LO}}} \left(\frac{U_i^{\max} - U_i^{\min}}{E_i} \right).$$

To enforce the utilization and compression relationship in Eqn. 15, for each task τ_i we add a constraint of the form

$$U_i \geq U_i^{\max} - E_i \lambda. \tag{18}$$

Then the objective is to

$$\textbf{minimize } \lambda. \tag{19}$$

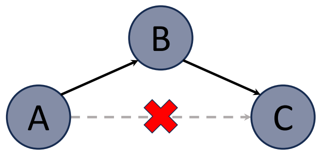


Fig. 4: Removing shortcut edges.

3) Schedulability: A set of tasks is *partitioned EDF schedulable* on m cores if and only if there exists a partition of tasks into m sets such that the total utilization of tasks in each set does not exceed 1. For each task τ_i and each core $k \in 1..m$, we define zero-one variables $x_{i,k}$ with the intended interpretation that $x_{i,k}$ takes the value 1 if task τ_i executes on core k, and 0 otherwise. So that each task τ_i is assigned to exactly one core, we add constraints of the form

$$\sum_{k=1}^{m} x_{i,k} = 1. \tag{20}$$

To enforce the schedulability condition, we require that for every core k, $\sum_i U_i \cdot x_{i,k} \leq 1$. To avoid quadratic constraints, we define a large constant value M and for each variable $x_{i,k}$ we define a corresponding variable $z_{i,k}$ constrained as

$$z_{i,k} \geq 0, \tag{21}$$

$$z_{i,k} \geq U_i + M \cdot (x_{i,k} - 1). \tag{22}$$

This way, if $x_{i,k}$ takes the value 1, the term $M \cdot (x_{i,k} - 1)$ will evaluate to 0 and so $z_{i,k}$ will be forced to take the value U_i; if $x_{i,k}$ instead takes the value 0, the term $M \cdot (x_{i,k} - 1)$ will take the value $-M$. If $M > \max_i\{U_i^{\max}\}$, the expression $U_i + M \cdot (x_{i,k} - 1)$ evaluates to a negative value, and thus, $z_{i,k}$ will be forced to 0. To enforce schedulability, we can therefore add a constraint of the following form for each core k:

$$\sum_{\tau_i \in \Gamma_{\text{LO}}} z_{i,k} \leq 1. \tag{23}$$

VII. Evaluation

A. Analysis of Span Constraints

We begin by analyzing the number of constraints necessary to enforce the intended interpretation of the span variables in the MIQP discussed in §IV-B and §IV-C. Though we have already provided theoretical upper bounds, we would like to now empirically quantify a range of realistic problem sizes associated with sets of synthetically-generated DAG tasks.

Experimental Setup: We generate DAGs according to a modified version of the Erdős-Rényi method [40]:

1) Select a number of vertices k for the DAG G (we iterate over values of k from 5–50).
2) For each pair of vertices in $\{v_2, ..., v_{k-1}\}$, add a connecting edge with probability p (we iterate over values of p from 0.05–0.95 in steps of 0.05). So the graph remains acyclic, we direct the edge from the smaller to larger vertex index.
3) Vertex v_1 is the source vertex: direct an edge from it to all remaining vertices (except v_k) with no incoming vertices. Similarly, vertex v_k is the sink: direct an edge to it from all vertices with no outgoing vertices. This guarantees that the DAG is weakly connected.
4) For every edge E connecting vertex v_a to v_b, if there exists a path from v_a to v_b in $G \backslash E$, then E is a shortcut and is removed as illustrated in Fig. 4. This guarantees that no path from source to sink is a subset of another path, so every path might form the critical path, depending on its vertex weights (i.e., the corresponding subtask execution times).

For each combination (k, p), we generate 10 000 graphs.

Counting Maximal Paths: For each DAG, we count the number of paths from the source to the sink vertex; these are exactly the set of maximal paths and correspond to constraints in the form of Eqn. 10. We plot the mean and maximum count for each pair (k, p) in Fig. 5.

We observe that an edge probability of 0.5 is expected to produce the largest number of maximal paths. For tasks with 50 subtasks and $p = 0.5$, 8465 constraints of the form of Eqn. 10 will be added on average with a maximum observed of 106 560. However, an edge probability of 0.55 gives the maximum observed overall at 133 632 such paths. In comparison, the maximum possible for the pathological case illustrated in Fig. 3 is $3^{(50-2)/3}$, which is over 43 million.

Counting Edges: For each DAG, we also count the number of edges remaining after removal of shortcut edges. Each such edge corresponds to a constraint in the form of Eqn. 12 or Eqn. 13. Results are plotted in Fig. 6.

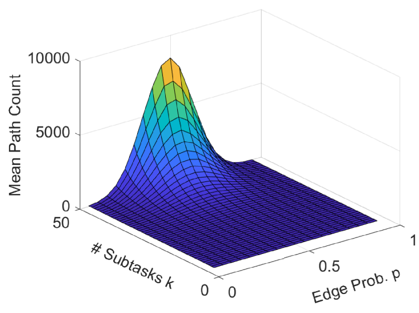
We observe that for smaller numbers of subtasks, an edge probability of 0.2 is expected to produce the largest number of edges, as edge shortcuts are removed after the initial set of edges are generated. As the number of subtasks k approaches 50, $p = 0.15$ is expected to result in the most edges: 106 on average. The maximum observed overall was 136. Most importantly, as the number of subtasks increases, the number of paths rapidly overtakes the number of edges. This suggests that the method in §IV-C for enforcing the intended interpretation of the span variables using a constraint for each edge in the task DAG will tend to scale better with problem size; we confirm this with the following experiments.

B. MIQP Solver Performance

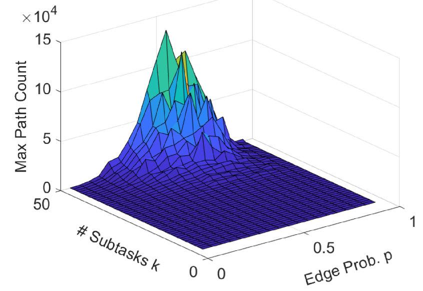
We now evaluate the feasibility of solving the optimization problem listed in Eqn. 4. We use version 10.0.3 [41] of the Gurobi Optimizer [22] to solve the MIQP. We measure execution times on a server with an AMD EPYC 9754 CPU and 128GB of RAM running Linux 5.14.0. Simultaneous Multithreading and CPU throttling are disabled.

Compressing Individual Tasks: We begin by randomly generating tasks according to the modified Erdős-Rényi method outlined above, using an edge probability of $p = 0.5$ since we have observed that this typically induces the greatest number of maximal paths. For each value k (number of subtasks) in 5–50, we generate 1000 such tasks, for a total of 46 000.

Each subtask $\tau_{i,j}$ has its elasticity $E_{i,j}$ randomly selected as an integer from the range 1–100. To assign a range of acceptable execution times to each subtask, we randomly

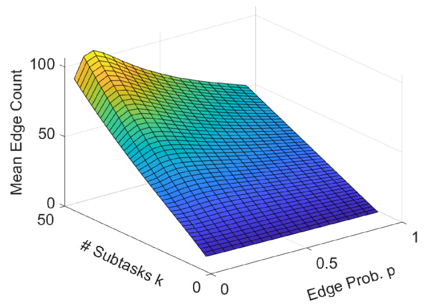


(a) Mean path counts

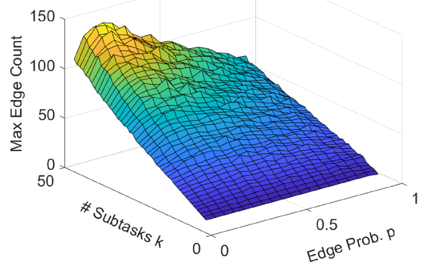


(b) Max path counts

Fig. 5: Maximal path count statistics.

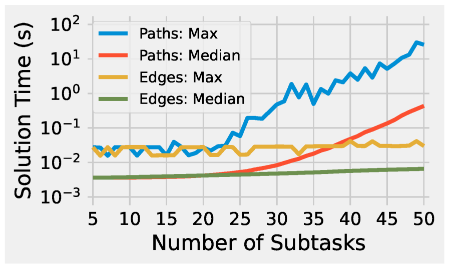


(a) Mean edge counts

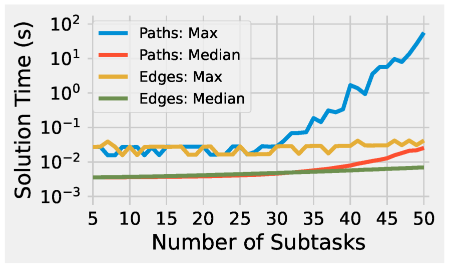


(b) Max edge counts

Fig. 6: Edge count statistics.



(a) Edge probability $p = 0.5$



(b) Edge probability $p = 0.2$

Fig. 7: MIQP times for individual tasks.

select two integer values in the range 1–100. The smaller value is assigned to $c_{i,j}^{\min}$ and the larger to $c_{i,j}^{\max}$. So that the task remains high-utilization even if all subtasks are assigned their minimum execution times, we randomly select D_i as an integer from the range $[L_i^{\max} + 1, C_i^{\min} - 1]$ (if $D_i \leq L_i$, the core assignment in Eqn. 2 becomes invalid). If for some task τ_i, $L_i^{\max} + 1 > C_i^{\min} - 1$, values of $c_{i,j}$ are regenerated.

Generated parameters are used with Eqn. 2 to determine the minimum $m_i^{\min}$ and maximum $m_i^{\max}$ number of cores to guarantee schedulability for each task; the integer value m of total cores available is selected uniformly from the range $[m_i^{\min}, m_i^{\max} - 1]$. We formulate two MIQPs for each task according to the procedure in §IV, using both proposed methods to enforce the intended interpretation of the span variables. We configure Gurobi to execute in a single thread.

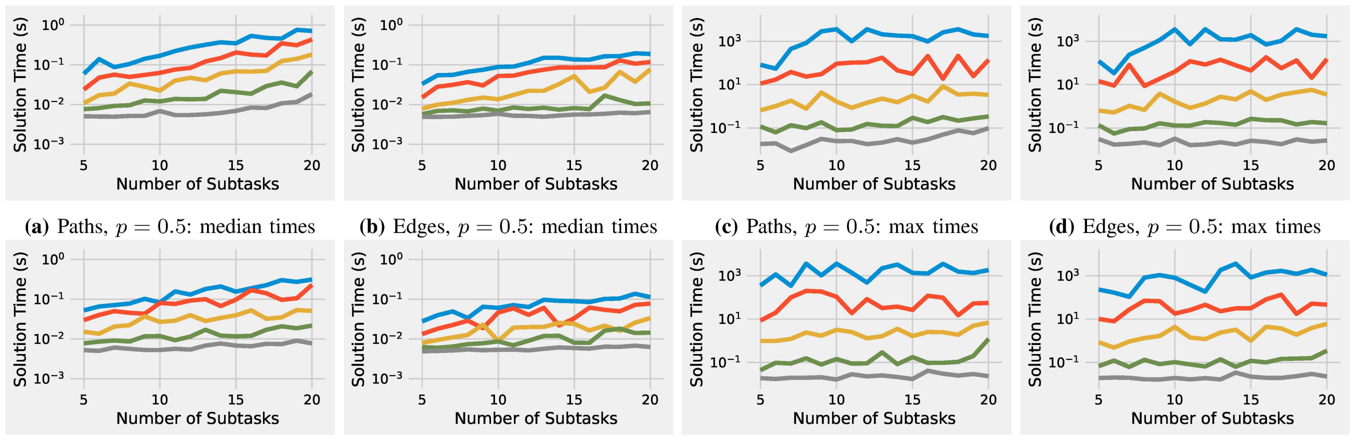
Results are plotted in Fig. 7a. We observe that for smaller numbers of subtasks (i.e., up to around 20), the selection of the method for enforcing the intended interpretation of the span variables does not have a significant impact on execution time. However, as the number of subtasks increases further, a constraint per edge becomes significantly faster. With a constraint per edge, a solution was reached in under 42ms in the worst case, with a median under 6.6ms for 50 subtasks. But with a constraint per path, solution search took up to 30.2 s in the worst case (726× slower), and for tasks with 50 subtasks, the median time was 434ms (66.7× slower).

We rerun the experiment, this time generating task DAGs using an edge probability of $p = 0.2$, since this typically induces the greatest number of edges. Nonetheless, Fig. 7b illustrates that the execution time behavior remains very similar when solving the MIQP that uses a constraint per edge. However, since the number of maximal paths decreases significantly, the median execution time of the MIQP that uses a constraint per path grows more slowly, remaining roughly equivalent to using a constraint per edge for tasks with up to about 30 subtasks. For tasks with 50 subtasks, the median time to reach a solution using a constraint per edge was under 7.0ms, compared to 25.5ms for a constraint per path (only a 3.67× difference). Despite the smaller average-case difference that arises from inducing more edges and fewer paths, using a constraint per edge remains the preferred approach for single tasks.

Joint Task Compression: We next consider the joint compression of multiple parallel tasks. We randomly generate task sets of size n from 2–10 in steps of 2. Every task in a task set is assigned the same number k of subtasks; we consider values of k from 5–20. For each pair (n, k), we generate 100 task sets using the above methodology with an edge probability of $p = 0.5$, and another 100 with an edge probability of $p = 0.2$, for a total of 16 000. Eqn. 3 determines the minimum $m_{\text{SUM}}^{\min}$ and maximum $m_{\text{SUM}}^{\max}$ number of cores to guarantee schedulability for each task system; the value m of total cores available is selected uniformly from $[m_{\text{SUM}}^{\min}, m_{\text{SUM}}^{\max} - 1]$.

We again formulate two MIQPs for each task according to the procedures in §IV, then solve with Gurobi using a single thread. This time, we force the solver to terminate if no solution is found after one hour. Results are plotted in Fig. 8.

We observe slight differences in maximum execution times between the two methods for representing span, and between DAGs generated with different edge probabilities. For $p = 0.5$ and a span constraint per path, the maximum observed execution time for 2 tasks was 97.8ms, 3.05× slower than the maximum 32.0ms observed with a constraint per edge (this is not visually obvious due to the logarithmic scale in the y-axis). For $p = 0.2$, the maximum observed execution time for 2 tasks was 41.4ms with a constraint per path, and 33.8ms with a constraint per edge: with fewer paths and more edges, the maximum time when using path constraints was cut



(a) Paths, $p = 0.5$: median times **(b)** Edges, $p = 0.5$: median times **(c)** Paths, $p = 0.5$: max times **(d)** Edges, $p = 0.5$: max times

(e) Paths, $p = 0.2$: median times **(f)** Edges, $p = 0.2$: median times **(g)** Paths, $p = 0.2$: max times **(h)** Edges, $p = 0.2$: max times

Fig. 8: Joint MIQP solver times. Series in each plot, from bottom to top, are for sets of 2, 4, 6, 8, and 10 tasks.

by $2.36\times$, whereas the maximum time associated with edge constraints only increased by 5%. Moreover, for more than 2 tasks, the median execution times associated with maximal path constraints are much higher than those associated with edge constraints, and even higher when there are more paths.

Most notably, execution times increase rapidly as tasks are added. For every 2 additional tasks added beyond the first 4, both the median and maximum execution times increase by about an order of magnitude. With a span constraint per path, 2 task sets timed out at the imposed 1 hour time limit for an edge probability $p = 0.2$ and 4 for $p = 0.5$. With a constraint per edge, only 1 timed out for $p = 0.2$ and 2 for $p = 0.5$. This suggests that neither solution approach that uses a *single* MIQP scales well with the number of tasks.

C. DP-Based Solution Performance

Though solving a single joint MIQP for larger sets of tasks rapidly becomes infeasible, but remains efficient for *individual* tasks. We therefore expect our DP-based approach of §V to outperform the single MIQP for larger sets of tasks. Moreover, we expect that if the MIQPs are solved offline when task parameters are characterized, it will be possible to quickly solve the DP *online* to achieve real-time task adaptation.

To test these hypotheses, we generate new task sets of size n from 2–20 in steps of 2, each assigned the same number k of subtasks from 5–50 in steps of 5. For each pair (n, k), we generate 100 task sets each for edge probabilities of $p = 0.5$ and $p = 0.2$ in the same manner as before, for a total of 20 000.

Measured execution times are plotted in Fig. 9. Results indicate that our DP-based approach remains efficient even for larger numbers of tasks — with an edge probability of $p = 0.5$, total execution time remains less than one minute for 20 tasks with 50 subtasks each. Compared to solving a single joint MIQP, which can take over an hour for 10 tasks with 20 subtasks, this a substantial improvement. Moreover, just solving the DP is even faster, taking under 60ms for up to 20 tasks with 50 subtasks. Where the MIQPs can be solved offline, this can enable real-time online adaptation and re-allocation of cores, e.g., during admission control of new tasks, or when the number of available processors changes.

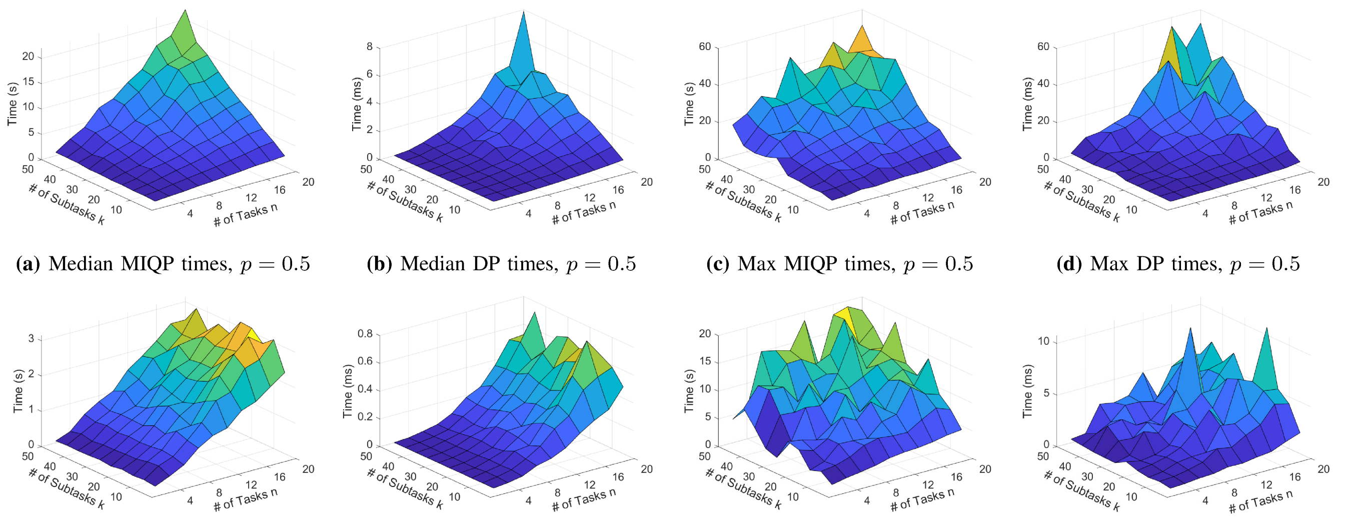
Results are even better for DAG tasks generated using an edge probability of $p = 0.2$. Total execution time remained under 19s, while solving the DP took under 11ms. Despite the number of edges being *higher*, on average, for $p = 0.2$, the DAG tasks thus generated are generally able to occupy fewer cores; thus, the number of MIQPs to solve, and the number of choices for the DP, are lower.

D. Comparison to Prior Work

We next compare our model to the earlier method of Orr et al. that holds task spans constant [12]. We intend to characterize the extent to which, by considering that the span term L_i in Eqn. 2 decreases with the workload C_i, our model can reduce the amount of compression necessary to achieve schedulability, and in doing so reduce the minimum number of cores needed to schedule a compressed task.

To do so, we consider the 92 000 individual tasks with numbers k of subtasks from 5–50 already generated for §VII-B. For each task τ_i, we compute the minimum and maximum total execution times $C_i^{\min}$, $C_i^{\max}$ and spans $L_i^{\min}$, $L_i^{\max}$. We then use these values with Eqn. 2 to compute the maximum number of cores $m_i^{\max}$ needed to schedule the task. We also calculate the minimum $m_i^{\min}$ to achieve schedulability under our model (using $L_i^{\min}$) and the minimum $m_i^{\min *}$ that arises from the model of Orr et al. due to keeping the span constant (using $L_i^{\max}$). For each number of cores in $[m_i^{\min *}, m_i^{\max} - 1]$, we determine the workload C_i to achieve schedulability according to Eqn. 2 under the model of Orr et al., again keeping the span fixed at $L_i^{\max}$. We compare this to the total workload $\sum_j c_{i,j}$ achieved by our MIQP-based model.

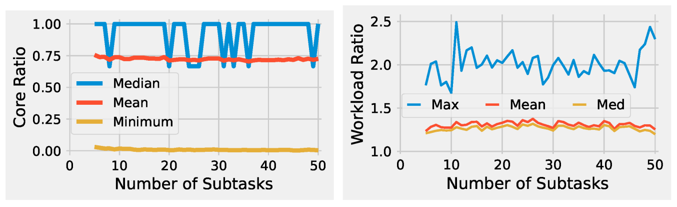
Comparison of Minimum Core Allocations: We begin by considering the ratio $m_i^{\min}/m_i^{\min *}$ of the minimum number of cores allowed by each model. These are plotted in Fig. 10a. We observe that the ratio does not appear to be highly dependent on the number of subtasks. However, it does illustrate that by also compressing task span, the minimum number of cores on which a task can be scheduled is decreased. On average, the value $m_i^{\min}$ achieved by our subtask-aware model is $0.72\times$ the value $m_i^{\min *}$. We also measure the value $\sum_i m_i^{\min} / \sum_i m_i^{\min *}$, aggregating the minimum demand for



(a) Median MIQP times, $p = 0.5$ **(b)** Median DP times, $p = 0.5$ **(c)** Max MIQP times, $p = 0.5$ **(d)** Max DP times, $p = 0.5$

(e) Median MIQP times, $p = 0.2$ **(f)** Median DP times, $p = 0.2$ **(g)** Max MIQP times, $p = 0.2$ **(h)** Max DP times, $p = 0.2$

Fig. 9: Execution time statistics for DP-based approach.



(a) Ratio of minimum allowed cores **(b)** Ratio of compressed workloads

Fig. 10: Comparison to the approach in [12].

cores under both models across all 46 000 sets of tasks. The result suggests that, on average, our model may be able to schedule sets of tasks on systems with 36% the number of cores needed by the earlier method in [12]. Moreover, in cases where most of a task's workload lies on its critical path, adjusting the span L_i during workload compression enabled schedulability on systems with fewer than 1% the number of cores needed when holding span constant.

Comparison of Total Workloads: We next compare the amount of computational workload that remains available to tasks when compressed under each model. For every number of cores on which each task can be compressed by both models, we compare the values C_i achieved by our subtask-level elastic scheduling model, to the values C_i^* achieved by the earlier model of Orr et al. [12]. Fig. 10b shows the ratio C_i/C_i^* of the two values. Once again, we do not see a significant relationship between the ratio and the number of subtasks. We also observe that by compressing task span, a task's total workload does not have to be compressed as much to be schedulable on a given number of cores. The median of the ratio C_i/C_i^* is 1.24, and at best, our subtask-level model achieves a workload 2.49× that of the original model in [12]. This suggests that our model allows workload-elastic tasks to complete more work, which should improve outcomes (e.g., result accuracy).

E. MILP for Partitioned EDF

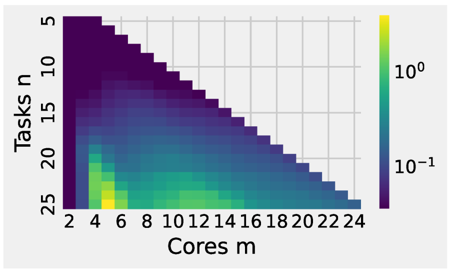
Finally, we evaluate the feasibility of solving the MILP for elastic scheduling under partitioned EDF. We randomly generate sets of sequential tasks of size n from 5–25. For each n we generate 1000 tasks, for a total of 26 000. Every task τ_i has a maximum utilization $U_i^{\max}$ of 1. Using the Dirichlet Rescale (DRS) algorithm [42], we assign minimum utilizations $U_i^{\min}$ uniformly from the space of selections satisfying the conditions that *(i)* the total minimum utilization $\sum_i U_i^{\min}$ is 1 and *(ii)* $U_i^{\min} \leq U_i^{\max}$ for every task τ_i. Elasticities E_i are selected uniformly from the integers 1–100. Since the MILP in §VI-B only assigns utilizations, we do not generate period or workload values for these tasks.

Our chosen utilization ranges make it *feasible* but *nontrivial* to find a compressed state for every core assignment in $[2, n-1]$. We do so for each taskset by constructing an MILP according to the procedure in §VI-B. As before, we solve in a single thread using Gurobi and measure the execution times.

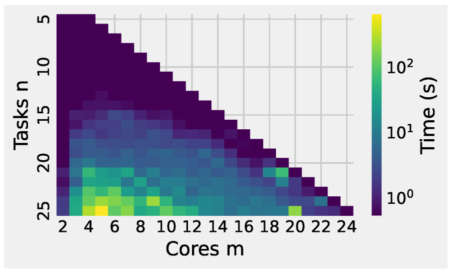
Results are plotted in Fig. 11. Observed times were typically sub-second, suggesting that our proposed MILP is feasible for offline characterization. Combined with our DP-based approach, low-utilization tasks can be jointly compressed with high-utilization tasks when available resources change. In multi-mode systems, where the collection of low-utilization tasks may change, the MILP may be run offline (and corresponding solutions stored online) for each discrete mode. However, the solver occasionally took several minutes (up to 11) to complete. Where new low-utilization tasks can be arbitrarily admitted to a running system, the pessimistic (but efficient) utilization-bound approach outlined in §VI-B might be necessary for online adaptation.

VIII. Case Study: FIMS

The Fast Integrated Mobility Spectrometer (FIMS) [43], [44] is a novel flown instrument for measuring and reporting atmospheric particle size distributions. Its computational pipeline [23], [24] *(i)* images particles in an aerosol chamber, *(ii)* processes these images to identify and bin particles in time windows (as FIMS responses), and *(iii)* inverts the responses to report size distributions. Accurate image processing is computationally intensive, using 10 intensity masks to isolate overlapping particles. Future drone-based deployments may



(a) Median times



(b) Max times

Fig. 11: Execution time statistics for partitioned EDF MILP.

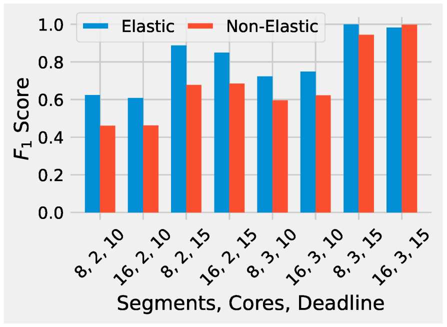


Fig. 12: Elastic vs Non-Elastic F_1 Scores.

also require shorter periods due to reduced instrument size and particle residence time in the chamber.

To improve latency, we segment each image for parallel processing. The image processing task is thus represented as a DAG with an initial subtask to read the image, parallel subtasks to process each segment, and a final subtask to aggregate and de-duplicate identified particles. Parallel execution raises challenges on embedded platforms where size, weight, and power (SWaP) constraints limit the number of cores. On quad-core platforms, image processing might be allocated just 2 or 3 dedicated cores, since the FIMS housekeeping data and response inversion tasks, as well as mission-critical drone control, must be scheduled concurrently.

Toward supporting these requirements, we apply our subtask-level elastic scheduling model to FIMS image processing. The parallel subtasks that process each region are elastic, with maximum workloads representing the worst-case time to apply all 10 masks, but at a minimum, only a single mask must be applied. Though the numbers of masks encode *discrete* computational modes, we consider workloads to be *continuously*-elastic, since more execution time increases the likelihood of more masks completing. We assign each subtask an elastic constant equal to the inverse of the average particle density: more masks are applied to denser regions.

To quantify these parameters, we run the FIMS pipeline in simulation on a Raspberry Pi 4B, with a 1.5GHz, 4-core Cortex-A72 CPU and 4GB of RAM. We use 12 000 images from the FIMS aerosol chamber collected over a 20 minute instrument run. We segment the images into 8 or 16 regions, and measure (after completion of all 10 image masks) the number of particles in each. With CPU throttling disabled, we profile the initial and final subtasks and the time to complete each mask for each segment. The resulting execution time distributions let us define minimum and maximum workloads for each subtask in each configuration.

To evaluate our model, we constrain image processing to 2 or 3 cores with 10 or 15ms deadlines, using 8 or 16 image segments. For each parameter combination, we use Gurobi to solve the resulting quadratic program, assigning execution time budgets to each image segment's corresponding subtask. We then consider the same sequence of test images; for each image segment, we randomly sample from the obtained execution time distributions to determine how many masks can complete within the given budget, then compare the resulting set of identified particles to the baseline where all complete.

Fig. 12 reports the F_1 score [45] of the comparison for each combination. This is the harmonic mean of the precision and recall, measuring the algorithm's ability to accurately identify particles that appear in the baseline while balancing the trade-off between false positives and false negatives. We compare to a non-elastic version where every region is given equal computational time within the deadline. Although the stochastic nature of the execution times allowed the non-elastic version occasionally to perform slightly better when little compression was necessary (see, e.g., the result for 3 cores with 16 segments and a 15ms), in general, by considering the individual importance of each subtask, our subtask-level elastic scheduling model enables better result accuracy.

IX. Conclusions and Future Work

This paper presents a new model of subtask-level elasticity for federated scheduling of parallel tasks, which considers the joint impact of compressing workloads of *each* subtask in the task system, including changes to task spans. We propose and evaluate two ways to formulate the problem as an MIQP that can be solved efficiently with Gurobi to make offline scheduling decisions. With proper offline characterization, a dynamic-programming (DP) algorithm enables pseudo-polynomial compression online to handle task admission or changes in resource availability. It also enables federated core allocation to elastic sequential tasks scheduled using fluid or partitioned EDF. For partitioned EDF, we present an MILP to find the minimum compression amount, and corresponding partition, for a given number of cores. We validate our model with the FIMS image processing task, where it improves accuracy of atmospheric aerosol particle detection.

Future directions include *(i)* addressing hybrid-utilization tasks, which may switch from high- to low-utilization during workload compression; *(ii)* considering interdependent subtask workloads; and *(iii)* evaluating multicore resource contention, which can be significant [46], [47], and might cause worst-case execution times to depend on core assignments.

Acknowledgment

This work was supported by NSF grants CPS-2229290 and CNS-2141256, NASA award 80NSSC21K1741, a Washington University OVCR seed grant, and a Washington University CSE/EECE seed grant. Thanks to the several attendees of the OPERA 2023 workshop, co-located with RTSS 2023, whose suggestions were invaluable in constructing this paper. We would especially like to thank Tanya Amert, Jian-Jia Chen, Mario Günzel, and Corey Tessler for their insights.

References

[1] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proc. of IEEE Real-Time Systems Symposium*, 1998. [Online]. Available: https://doi.org/10.1109/REAL.1998.739754

[2] 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, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/12.990127

[3] J. Kim, H. Kim, K. Lakshmanan, and R. Rajkumar, "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car," in *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2013, pp. 31–40.

[4] G. A. Elliott, K. Yang, and J. H. Anderson, "Supporting real-time computer vision workloads using openvx on multicore+gpu platforms," in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 273–284.

[5] S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli, "Data flow orb-slam for real-time performance on embedded gpu boards," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5370–5375.

[6] D. Ferry, A. Maghareh, G. Bunting, A. Prakash, K. Agrawal, C. Gill, C. Lu, and S. Dyke, "On the performance of a highly parallelizable concurrency platform for real-time hybrid simulation," in *The Sixth World Conference on Structural Control and Monitoring*, 2014.

[7] D. Ferry, G. Bunting, A. Maghareh, A. Prakash, S. Dyke, K. Agrawal, C. Gill, and C. Lu, "Real-time system support for hybrid structural simulation," in *Proceedings of the 14th International Conference on Embedded Software*, ser. EMSOFT '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2656045.2656067

[8] M. Sudvarg, J. Buhler, J. H. Buckley, W. Chen *et al.*, "A Fast GRB Source Localization Pipeline for the Advanced Particle-astrophysics Telescope," *PoS*, vol. ICRC2021, p. 588, 2021.

[9] J. Wheelock, W. Kanu, M. Sudvarg *et al.*, "Supporting multi-messenger astrophysics with fast gamma-ray burst localization," in *Proc. of IEEE/ACM HPC for Urgent Decision Making Workshop*. IEEE, Nov. 2021.

[10] Y. Htet, M. Sudvarg, J. Buhler, R. Chamberlain, W. Chen, J. H. Buckley *et al.*, "Prompt and Accurate GRB Source Localization Aboard the Advanced Particle Astrophysics Telescope (APT) and its Antarctic Demonstrator (ADAPT)," in *Proc. of 38th Int'l Cosmic Ray Conference*, vol. 444. Sissa Medialab, Jul. 2023, pp. 956:1–956:9.

[11] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *Proc. of 26th Euromicro Conference on Real-Time Systems*, 2014, pp. 85–96. [Online]. Available: https://doi.org/10.1109/ECRTS.2014.23

[12] J. Orr, C. Gill, K. Agrawal, S. Baruah *et al.*, "Elasticity of workloads and periods of parallel real-time tasks," in *Proc. of 26th International Conference on Real-Time Networks and Systems*. ACM, 2018, pp. 61–71. [Online]. Available: https://doi.org/10.1145/3273905.3273915

[13] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling," in *Proc. of IEEE International Real-Time Systems Symposium*, 2006, pp. 236–245. [Online]. Available: https://doi.org/10.1109/RTSS.2006.24

[14] ——, "Generalized elastic scheduling for real-time tasks," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 480–495, Apr. 2009. [Online]. Available: https://doi.org/10.1109/TC.2008.175

[15] C. Lu, X. Wang, and X. Koutsoukos, "End-to-end utilization control in distributed real-time systems," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, 2004, pp. 456–466.

[16] ——, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, 2005.

[17] A. Soyyigit, S. Yao, and H. Yun, "Anytime-lidar: Deadline-aware 3d object detection," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2022, pp. 31–40. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/RTCSA55878.2022.00010

[18] Y. Bai, L. Li, Z. Wang, X. Wang, and J. Wang, "Performance optimization of autonomous driving control under end-to-end deadlines," *Real-Time Systems*, vol. 58, no. 4, pp. 509–547, Dec 2022.

[19] M. Sudvarg, J. Buhler, R. Chamberlain, C. Gill, J. Buckley, and W. Chen, "Parameterized workload adaptation for fork-join tasks with dynamic workloads and deadlines," in *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023, pp. 1–10.

[20] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun 1996. [Online]. Available: https://doi.org/10.1007/BF01940883

[21] S. Baruah, "Partitioned edf scheduling: a closer look," *Real-Time Systems*, vol. 49, no. 6, pp. 715–729, Nov 2013. [Online]. Available: https://doi.org/10.1007/s11241-013-9186-0

[22] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[23] D. Wang, J. Zhang, J. Buhler, and J. Wang, "Real-time analysis of aerosol size distributions with the fast integrated mobility spectrometer (FIMS)," in *41st Conference of American Association for Aerosol Research (AAAR)*, Oct. 2023. [Online]. Available: https://aaarabstracts.com/2023/view_abstract.php?pid=752

[24] M. Sudvarg, A. Li, D. Wang, S. Baruah, J. Buhler, C. Gill, N. Zhang, and P. Ekberg, "Elastic scheduling for harmonic task systems," in *2024 Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2024.

[25] J. Orr, C. Gill, K. Agrawal, J. Li, and S. Baruah, "Elastic scheduling for parallel real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 6, no. 1, p. 05:1–05:14, May 2019. [Online]. Available: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v006-i001-a005

[26] M. Sudvarg, J. Buhler, R. D. Chamberlain, C. Gill, J. Buckley, and W. Chen, "Parameterized workload adaptation for fork-join tasks with dynamic workloads and deadlines," in *Proc. of IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2023, pp. 232–242.

[27] J. Orr, J. C. Uribe, C. Gill, S. Baruah *et al.*, "Elastic scheduling of parallel real-time tasks with discrete utilizations," in *Proc. of 28th International Conference on Real-Time Networks and Systems*. ACM, 2020, pp. 117–127. [Online]. Available: https://doi.org/10.1145/3394810.3394824

[28] J. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

[29] Y. Bai, Z. Wang, X. Wang, and J. Wang, "Autoe2e: End-to-end real-time middleware for autonomous driving control," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 1101–1111.

[30] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 689–12 697.

[31] S. Odagiri and H. Goto, "On the greatest number of paths and maximal paths for a class of directed acyclic graphs," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 97, no. 6, pp. 1370–1374, 2014.

[32] M. Sudvarg and C. Gill, "Analysis of federated scheduling for integer-valued workloads," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 12–23. [Online]. Available: https://doi.org/10.1145/3534879.3534892

[33] P. Turán, "On an extremal problem in graph theory," *Matematikai és Fizikai Lapok*, vol. 48, pp. 436–452, 1941.

[34] H. Kellerer, U. Pferschy, and D. Pisinger, *The Multiple-Choice Knapsack Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 317–347. [Online]. Available: https://doi.org/10.1007/978-3-540-24777-7_11

[35] J. Orr and S. Baruah, “Multiprocessor scheduling of elastic tasks,” in *Proc. of 27th International Conference on Real-Time Networks and Systems*. ACM, 2019, pp. 133–142. [Online]. Available: https://doi.org/10.1145/3356401.3356403

[36] M. Sudvarg, C. Gill, and S. Baruah, “Improved implicit-deadline elastic scheduling,” in *Proceedings of the 14th IEEE International Symposium on Industrial Embedded Systems (SIES 2024)*. IEEE, 2024. [Online]. Available: https://sudvarg.com/publications/SIES2024_improved_implicit_elastic.pdf

[37] ——, “Linear-time admission control for elastic scheduling,” *Real-Time Systems*, vol. 57, no. 4, pp. 485–490, Oct 2021. [Online]. Available: https://doi.org/10.1007/s11241-021-09373-4

[38] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, “LITMUSˆRT : A testbed for empirically comparing real-time multiprocessor schedulers,” in *2006 27th IEEE International Real-Time Systems Symposium (RTSS’06)*, 2006, pp. 111–126.

[39] M. Sudvarg, S. Baruah, and C. Gill, “Elastic scheduling for fixed-priority constrained-deadline tasks,” in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 11–20.

[40] L.-C. Canon, M. E. Sayah, and P.-C. Héam, “A comparison of random task graph generation methods for scheduling problems,” in *Euro-Par 2019: Parallel Processing*, R. Yahyapour, Ed. Cham: Springer International Publishing, 2019, pp. 61–73.

[41] S. Mars, “Gurobi 10.0.3 released,” Gurobi Optimization, Technical Report, September 2023. [Online]. Available: https://support.gurobi.com/hc/en-us/articles/18530517319953-Gurobi-10-0-3-released

[42] 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.

[43] J. Wang, M. Pikridas, S. R. Spielman, and T. Pinterich, “A fast integrated mobility spectrometer for rapid measurement of sub-micrometer aerosol size distribution, part i: Design and model evaluation,” *Journal of Aerosol Science*, vol. 108, pp. 44–55, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021850216304426

[44] Y. Wang, T. Pinterich, and J. Wang, “Rapid measurement of sub-micrometer aerosol size distribution using a fast integrated mobility spectrometer,” *Journal of Aerosol Science*, vol. 121, pp. 12–20, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021850217305049

[45] N. Chinchor, “Muc-4 evaluation metrics,” in *Proceedings of the 4th Conference on Message Understanding*, ser. MUC4 ’92. USA: Association for Computational Linguistics, 1992, p. 22–29. [Online]. Available: https://doi.org/10.3115/1072064.1072067

[46] A. Li, M. Sudvarg, H. Liu, Z. Yu, C. Gill, and N. Zhang, “Polyrhythm: Adaptive tuning of a multi-channel attack template for timing interference,” in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 225–239.

[47] A. Li, J. Wang, S. Baruah, B. Sinopoli, and N. Zhang, “An empirical study of performance interference: Timing violation patterns and impacts,” in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 320–333.