Subtask-Level Elasticity for Federated Scheduling of Parallel Tasks

Marion Sudvarg*, Chris Gill†, Jeremy Buhler‡
Department of Computer Science & Engineering
Washington University in St. Louis
St. Louis, Missouri
Email: *msudvarg@wustl.edu, †cdgill@wustl.edu, ‡jbuhler@wustl.edu

***Abstract*—Buttazzo et al.'s elastic scheduling model provides a framework in which task utilizations may be "compressed" to guarantee schedulability despite limited resources. Each task is assigned a range of acceptable utilizations and an "elastic constant" representing the relative adaptability of its utilization. Other prior work has extended the model to federated scheduling of computationally elastic parallel tasks, in which each high utilization task is assigned dedicated cores in sufficient number to guarantee schedulability. If the demand for cores exceeds the supply, task execution times are reduced until the system becomes schedulable. This model assumes that each task's span is a constant value; it does not consider that adjusting execution times may change the critical path of the computation. In this work, we present an extension of the model in which *each subtask* is assigned a range of workloads and an elastic constant. By adjusting subtask workloads, both the total execution time and the span of each task may change. We formulate the problem as a mixed-integer quadratic program, and perform a preliminary evaluation of SCIP, a non-commercial off-the-shelf solver, for solving the problem.**

***Index Terms*—real-time systems, elastic scheduling, parallel DAG tasks, mixed integer quadratic programs**

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. Under this model, compression is realized by increasing task periods: a task with workload C_i and newly-assigned utilization U_i would have its period set as $T_i = C_i/U_i$.

Chantem et al. [3], [4] formulated a quadratic optimization problem that assigns utilizations according to Buttazzo's proportional compression. The problem is constrained according to the minimum serviceable and maximum desired utilizations of each task, and by the utilization bound of the system. This allowed extensions of the elastic framework to other task models with schedulability tests that do not rely strictly on a utilization bound, including to federated scheduling of parallel real-time tasks [5] for which periods [6] are adjusted in response to reduced utilization assignments. In [7], the model was extended to allow parallel workloads to be adjusted instead 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 [8].

Under federated scheduling, each high utilization parallel task τ_i is assigned m_i dedicated processor cores according to:

$$m_i = \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil \tag{1}$$

where C_i, L_i, and T_i are respectively the workload, span, and period of the task. This was proven in [5] to be sufficient to guarantee schedulability. Under the elastic model in [7], if the total number of assigned cores exceeds the number available in the system, task utilizations are compressed by reducing workloads C_i until the total core assignment becomes feasible.

This model is fundamentally limited, however, because it only considers the aggregate implication of reducing a parallel task's overall workload, and not the individual implications of reducing the workloads of each *subtask*. Changing the computational workload of each subtask may fundamentally affect quality of outcome (e.g., control performance, prediction accuracy, etc.) in different ways [9], [10]. Furthermore, the model holds the values L_i constant. Because L_i represents the parallel task's critical path of computation, decreasing workload C_i may also decrease span, which further reduces the necessary allocation of processor cores to the task. For this effect to be captured, an elastic model must be cognizant of the DAG structure induced by the precedence constraints among the subtasks composing each parallel task. To these ends, we propose a new model of *subtask-level elasticity for federated scheduling of parallel tasks* in which each subtask is assigned a range of acceptable workloads and its own elastic constant. The elastic scheduling model is thus applied to the complete collection of subtasks in the system to guarantee schedulability according to the resulting task execution times and spans.

This research was supported in part by NSF award CNS-2229290 (CPS) and NASA award 80NSSC21K1741.

In this work, we formulate assignment of workloads to each subtask as a constrained optimization problem. The inverse-elasticity-weighted sum of squared deviations of each subtask's compressed utilization from its nominal value is minimized subject to the constraint that the total joint assignment of processor cores to tasks according to Eqn. 1 does not exceed the number available. We outline how to represent the problem as a mixed-integer quadratic program (MIQP); empirically demonstrate how the number of constraints grows with task complexity; then implement the problem using SCIP [11], a non-commercial, off-the-shelf solver. We demonstrate that even for systems of up to 10 tasks, each with 10 subtasks, the MIQP can often be solved in under a minute using a single-threaded run of the solver.

II. Background and System Model

In this work, we consider a system Γ of n independent, sporadic, parallel, implicit-deadline tasks. Each task τ_i represents a sequence of jobs and is characterized by a *workload* C_i, representing the worst-case execution time of each of its jobs; and by a *period* T_i, representing the minimum inter-arrival time between consecutive jobs. Each task's *deadline* D_i implicitly equals its period, i.e., $D_i = T_i$. The focus of this paper is *federated scheduling*: each task is assigned a dedicated set of cores on which it alone executes (this paradigm has been successfully used in applications such as real-time hybrid simulations for structural engineering [12]).

Each task τ_i consists of a set of subtasks $\tau_{i,j}$ with a precedence relation $\prec$ over them. Each individual subtask $\tau_{i,j}$ is characterized by a workload $c_{i,j}$, representing its worst-case execution time. Subtasks may run in parallel, except as constrained by the 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 $\tau_{i,a}$. The *span* L_i for the corresponding task τ_i is the length of the critical-path of the DAG, i.e., the longest path, weighted by execution time, among any two vertices in the graph.

The model presented by Orr et al. in [7] additionally assigns each parallel task τ_i an *elastic constant* E_i, representing "the flexibility of the task to vary its utilization" [1]. If the demand for processor cores exceeds the number available, each task's utilization is compressed by decreasing its workload C_i within a continuous range of acceptable values $[C_i^{\min}, C_i^{\max}]$. From these, a corresponding range of utilizations $U_i^{\min} = C_i^{\min}/T_i$ and $U_i^{\max} = C_i^{\max}/T_i$ are derived. On a system with m available processor cores, workloads are assigned to each task according to the quadratic objective function for elastic scheduling proposed by Chantem et al. [3], [4], with the original schedulability condition replaced by Eqn. 1:

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

$$\text{s.t.} \quad \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil \leq m \tag{2b}$$

$$\forall_i, \quad U_i^{\min} \leq U_i \leq U_i^{\max} \tag{2c}$$

This model is limited, however, as it holds each task's span L_i constant. 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 in [7] may be pessimistic in resource allocation and overcompress task workloads. Consider, for example, 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. 1. But the workload needs only to be reduced by 1 unit along its critical path ($C_i = 10$ and $L_i = 3$) to be schedulable.

Furthermore, the workload assigned to each individual subtask may uniquely impact result quality. This was argued in the context of control performance under sequential end-to-end task execution in autonomous vehicles [9], and extends to parallel tasks such as real-time gamma-ray burst localization aboard space-based instruments [10].

To address these limitations, this paper modifies the model in [7] by assigning to *each* subtask $\tau_{i,j}$ a continuous range of execution times $[c_{i,j}^{\min}, c_{i,j}^{\max}]$ and an elasticity $E_{i,j}$. Subtask workloads $c_{i,j}$ are then selected to minimize a modified version of objective (2a) that considers the deviations of individual *subtask* utilizations from their desired values:

$$\min_{c_{i,j}} \quad \sum_{\tau_{i,j}} \frac{1}{E_{i,j}T_i^2}(c_{i,j}^{\max} - c_{i,j})^2 \tag{3a}$$

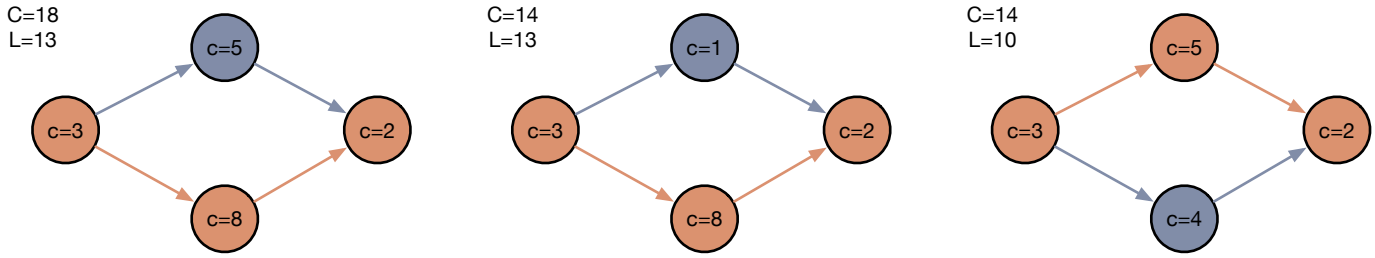
$$\text{s.t.} \quad \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i(\{c_{i,j}\})}{T_i - L_i(\{c_{i,j}\})} \right\rceil \leq m \tag{3b}$$

$$\forall_{i,j}, \quad c_{i,j}^{\min} \leq c_{i,j} \leq c_{i,j}^{\max} \tag{3c}$$

where the span L_i is expressed here as a function of the workloads assigned to each subtask in τ_i.

III. Solution Approach

We propose that the optimization problem (3) expressed in the previous section may be formulated as a mixed-integer quadratic program (MIQP) and solved using one of many off-the-shelf solvers. We demonstrate an approach to constructing the problem for SCIP [11], a non-commercial, open-source solver. SCIP supports both integer and continuous variables, and both linear and quadratic constraints may be defined. However, as of version 8.0 [13], it does not support quadratic *objectives*. As such, the MIQP is formulated as follows:



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

Fig. 1: An elastic DAG task. The span and critical path may change depending on which subtask workloads are compressed.

1) For each subtask $\tau_{i,j}$, define continuous variables $c_{i,j}$, representing the workload assigned to the subtask, and constrained as in (3c):

$$c_{i,j}^{\min} \leq c_{i,j} \leq c_{i,j}^{\max} \tag{4}$$

2) Define a non-negative continuous variable O representing objective (3a):

$$\textbf{minimize } O \tag{5}$$

3) To enforce this intended interpretation on the variable O, we add a constraint of the form:

$$O \geq \sum_{\tau_{i,j}} \frac{1}{E_{i,j}T_i^2}(c_{i,j}^{\max} - c_{i,j})^2$$

where $E_{i,j}$, $c_{i,j}^{\max}$, and T_i are constant values for all i, j. As O is to be minimized, it will take a value equal to the RHS of the inequality. Separating linear, quadratic, and constant terms yields the constraint:

$$O + \sum_{\tau_{i,j}} \frac{2c_{i,j}^{\max}}{E_{i,j}T_i^2} c_{i,j} - \sum_{\tau_{i,j}} \frac{1}{E_{i,j}T_i^2} c_{i,j}^2 \geq \sum_{\tau_{i,j}} \frac{(c_{i,j}^{\max})^2}{E_{i,j}T_i^2} \tag{6}$$

4) For each task τ_i, define a non-negative continuous variable L_i representing its span, which can be expressed as:

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

over the set of paths $\{p_{i,k}\}$ between pairs of vertices in the task's representative DAG. To simplify this, we consider tasks τ_i for which the DAG has a single source vertex s and sink vertex t. (Any task DAG τ_i, even those that are not weakly-connected, can be represented as such: 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, add a 0-weight vertex $v_{i,t}$, connected with edges from all vertices that do not already have outgoing edges.) 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:

$$L_i - \sum_{v_{i,j} \in p_{i,k}} c_{i,j} \geq 0 \tag{7}$$

5) It is also a requirement that L_i does not exceed T_i for τ_i to be schedulable, as $D_i = T_i$. To enforce this, we add constraints of the form:

$$L_i \leq T_i \tag{8}$$

6) For each task τ_i define a non-negative **integer** variable m_i representing the number of cores allocated to the task. This should be in sufficient number to guarantee schedulability according to Eqn. 1. To enforce this intended interpretation, we add constraints of the form:

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

Since m_i is specified to be an integer variable, it will respect the ceiling operator that appears in Eqn. 1. Rearranging, this yields quadratic constraints of the form:

$$-m_i L_i + T_i m_i + L_i - \sum_j c_{i,j} \geq 0 \tag{9}$$

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

$$\sum_i m_i \leq m \tag{10}$$

where m is a constant integer value.

Constraints (7) and (8) jointly constrain the span L_i of each task τ_i to the range:

$$\sum_{v_{i,j} \in p_{i,k}} c_{i,j} \leq L_i \leq T_i \tag{11}$$

L_i will remain strictly less than T_i, as constraint (9) will cause $m_i \to \infty$ as $L_i \to T_i^-$. We note that, for low-utilization tasks, if $L_i = C_i$ then Eqn. 1 assigns $m = 0$. This approach is therefore unsuitable for sequential tasks. A formal proof of correctness is outside the scope of this paper.

IV. Analysis of Constraints

The MIQP, as formulated above, has a single variable O for the objective, variables m_i and L_i for each task τ_i, and variables $c_{i,j}$ for each subtask $\tau_{i,j}$. Equations 6 and10 both represent a single constraint, though with numbers of terms linear in the total number of subtasks and tasks, respectively. Equations 8 and 9 both represent a constraint per task, with each constraint in Eqn. 9 having a number of terms linear in the number of subtasks of the corresponding task. Eqn. 4

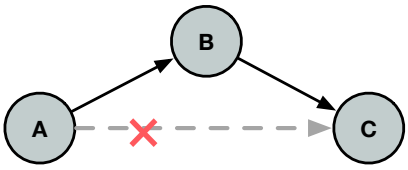


Fig. 2: Remove edges forming redundant paths.

represents a constraint for each subtask. Eqn. 7, however, represents a constraint for every path from each task DAG's source vertex to its sink.

We defer to future work a derivation of a generally tight upper bound on the number of such paths. In this paper, we empirically measure the mean and maximum number of such paths over a large number of synthetically generated DAG tasks. We generate DAGs according to a modified version of the Erdős-Rényi method [14]:

1) Select a number of vertices k for the DAG G (we iterate over values of k from 5 to 30 in steps of 1).
2) For each pair of vertices in $\{v_2, \ldots, v_{k-1}\}$, an edge connecting them is added with some probability p (we iterate over values of p from 0.05 to 0.95 in steps of 0.05). The edge is always directed from smaller to larger vertex index to guarantee that the graph remains acyclic.
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 remove E, as illustrated in Fig. 2. 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).

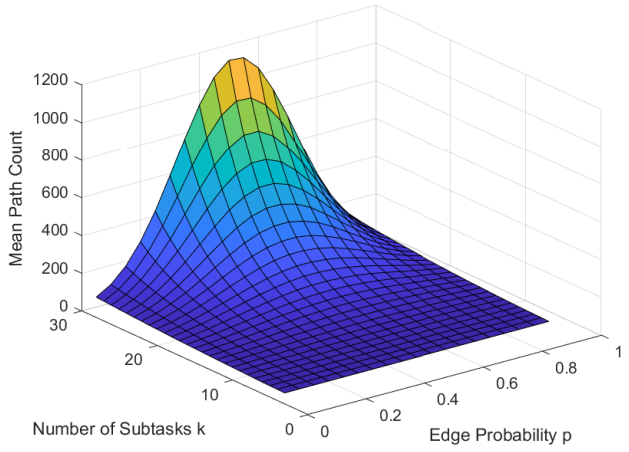
For each value of k and p, we randomly generate 100 000 graphs according to this procedure. For each DAG, we count the number of paths from the source to the sink vertex, then calculate the mean and maximum count for each pair (k, p). Results are plotted in Fig. 3.

We observe that an edge probability of 0.5 is expected to produce the largest number of critical path candidates. For tasks with 30 subtasks and an edge probability of 0.5, almost 1200 constraints of the form in Eqn. 7 will be added, with a maximum observed of over 10 000.

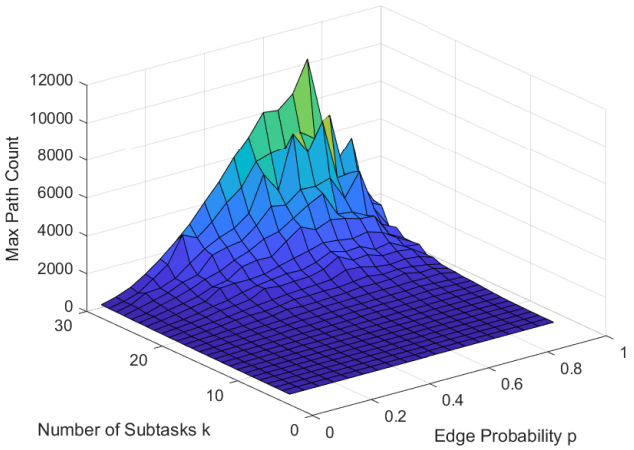
In fact, for k vertices, there exists a DAG with $3^{\lfloor (k-2)/3 \rfloor}$ paths that might form a critical path. Consider the pathological case, illustrated in Fig. 4, of a DAG consisting of a source S and sink T with all other vertices arranged in a sequence of groups of 3. An edge connects each vertex in one group to all vertices in the subsequent group. We do not claim this to be a tight upper bound, but it illustrates that the problem size may grow exponentially in the number of subtasks.

V. Preliminary Evaluation with SCIP

We perform a preliminary evaluation of the feasibility of using an off-the-shelf MIQP solver to assign execution times



(a) Mean count.



(b) Maximum count.

Fig. 3: Number of paths from source to sink.

to subtasks according to the optimization problem listed in Eqn. 3. We randomly generate task sets of size n from 2 to 10 in steps of 2. Every task in a task set is assigned the same number k of subtasks; for each value n, we consider values of k from 5 to 10 in steps of 1. For each pair (n, k), we generate 20 task sets, for a total of 600. Each task DAG has edges assigned according to the modified Erdős-Rényi method outlined in Section IV with an edge probability $p = 0.5$.

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 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

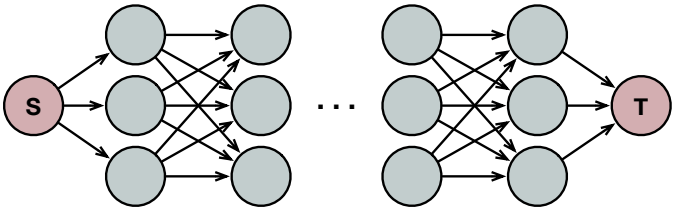
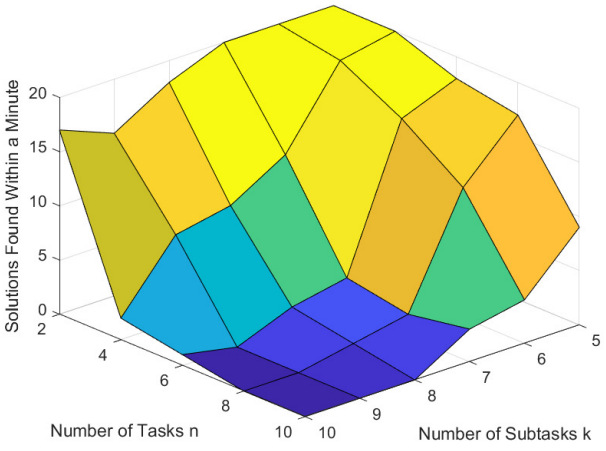
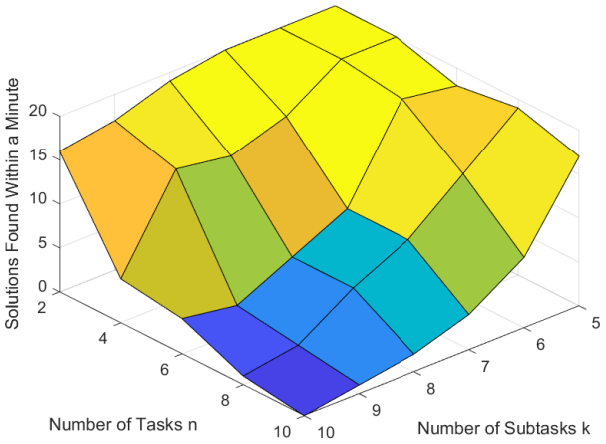


Fig. 4: A DAG with $3^{(k-2)/3}$ candidate critical paths.



(a) Number of solutions (out of 20) found within a minute.



(b) Number of solutions (out of 20) found within an hour.

Fig. 5: SCIP solver results.

remains high in utilization even if all subtasks are assigned their minimum execution times, we randomly select T_i as an integer from the range $[L_i^{\max} + 1, C_i^{\min} - 1]$ (if $D \leq L_i$, the core assignment in Eqn. 1 becomes invalid). If for some task τ_i, $L_i^{\max} + 1 > C_i^{\min} - 1$, values of $c_{i,j}$ are regenerated.

The values $C_i^{\min}$, $L_i^{\min}$, $C_i^{\max}$, $L_i^{\max}$, and T_i thus generated are used with Eqn. 1 to determine the minimum and maximum number of cores to guarantee schedulability for each task system; the integer value m of total cores available is generated uniformly in this range. For each task set, we formulate an MIQP according to the procedure in Section III. We use a custom C++ wrapper into which we link version 8 of the SCIP solver [13] to execute the MIQP. We evaluate the execution times on a server with two Intel Xeon Gold 6130 (Skylake) 2.1 GHz processors and 32GB of memory.

As a preliminary evaluation, we count the task sets for which a single-threaded run of SCIP is able to find an optimal value in a minute or less, then in an hour or less. Results are illustrated in Fig. 5. For smaller problem sizes, SCIP can rapidly find a solution. But for task sets with at least 8 tasks, each having at least 9 subtasks, SCIP was unable to find a solution in this limited time frame. Of the 600 problems considered, SCIP solved 268 (about 45%) within a minute, and 344 (about 57%) within an hour.

VI. Conclusions and Future Work

In this paper, we present a new model of subtask-level elasticity for federated scheduling of parallel tasks. The model considers the joint impact of compressing the workloads of *each* subtask within the task system, including changes to each task's span, which also affects the assignment of processor cores to each task. While still in preliminary stages of evaluation, this work suggests that the problem can be formulated as an MIQP and solved for small task systems using SCIP, a non-commercial off-the-shelf solver. However, as the problem size increases, the number of constraints grows exponentially in the worst-case; the problem may therefore become infeasible for complex sets of parallel tasks.

As future work, we plan to evaluate the problem using other solvers. We will also consider a refinement to the problem formulation, whereby the set of constraints imposed by every possible critical path through the task DAG (Eqn. 7) can be pruned. For any pair of paths $p_{i,a}$, $p_{i,b}$, if the maximum weighted length of $p_{i,a}$ is less than the minimum weighted length of $p_{i,b}$ (as defined by the range of execution times assigned to each corresponding subtask), then the constraint corresponding to $p_{i,a}$ can be removed. Finally, we will consider alternative solution approaches, e.g., using iterative techniques or an MIQP solver to find the *amount* of compression, λ, that must be applied to the task system for it to become schedulable, similarly to the approaches in [15].

Acknowledgment

Thanks to Ao Li of Washington University in St. Louis for beautifully illustrating Figures 1, 2, and 4.

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] 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

[4] ——, "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

[5] 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

[6] 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

[7] 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

[8] 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.

[9] 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.

[10] 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.

[11] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, Jul 2009. [Online]. Available: https://doi.org/10.1007/s12532-008-0001-1

[12] 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

[13] K. Bestuzheva *et al.*, "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html

[14] 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.

[15] 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.