A Type-Theoretic Framework for Efficient and Safe Colocation of Periodic Real-time Systems

VATCHE ISHAKIAN
Computer Science Department
Boston University, USA
visahak@cs.bu.edu

AZER BESTAVROS
Computer Science Department
Boston University, USA
best@cs.bu.edu

ASSAF KFOURY
Computer Science Department
Boston University, USA
kfoury@cs.bu.edu

***Abstract*—Desirable application performance is typically guaranteed through the use of Service Level Agreements (SLAs) that specify fixed fractions of resource capacities that must be allocated for *unencumbered* use by the application. The mapping between what constitutes desirable performance and SLAs is not unique: *multiple* SLA expressions might be functionally equivalent. Having the flexibility to transform SLAs from one form to another in a manner that is provably *safe* would enable hosting solutions to achieve significant efficiencies. This paper demonstrates the promise of such an approach by proposing a *type-theoretic* framework for the representation and safe transformation of SLAs. Based on that framework, the paper describes a methodical approach for the inference of efficient and safe mappings of periodic, real-time tasks to the physical and virtual hosts that constitute a hierarchical scheduler. Extensive experimental results support the conclusion that the flexibility afforded by safe SLA transformations has the potential to yield significant savings.**

I. INTRODUCTION

Motivation: The wide proliferation and adoption of virtualization technologies can be attributed to the various benefits they deliver, including cost efficiency (through judicious resource consolidation), deployment flexibility (through just-in-time "cloud" resource acquisition), simplified management (through streamlined business processes), among others. Virtualization delivers these benefits thanks in large to a key attribute: *performance isolation* – the ability of a user (or a set of applications thereof) to acquire appropriate fractions of shared fixed-capacity resources for *unencumbered* use subject to well-defined, binding Service-Level Agreements (SLAs) that ensure the satisfaction of minimal Quality of Service (QoS) requirements. Such SLAs are guaranteed through the underlying resource allocation and scheduling mechanisms of the hosting environment, which operate at a macroscopic scale (*e.g.*, enforcing specific resource utilization ratios over relatively long time scales). While appropriate for most applications, such coarse SLAs do not cater well to the needs of real-time applications, whose QoS constraints require resource allocations at a more granular scale – *e.g.*, through the specification of a worst-case periodic resource utilization.

This research was supported in part by NSF awards #0720604, #0735974, #0820138, and #0952145.

A very effective mechanism for dealing with this mismatch is the use of *hierarchical scheduling*, whereby the granularity of the reservations is refined as virtualization layers are traversed. Using hierarchical scheduling, resources are allocated by a parent scheduler at one level of the hierarchy to a child scheduler (or a leaf application) at the next level of the hierarchy. Conceptually, at any given layer of this hierarchy, the parent scheduler can be seen as allocating a virtual slice of the host at some granularity which is further refined by lower-layer schedulers, until eventually appropriated and consumed by a leaf application.

Independent of the recent proliferation of virtualization technologies, hierarchical scheduling (and in particular hierarchical CPU scheduling) has been a topic of research for over a decade because it allowed multiple scheduling mechanisms to co-exist on the same infrastructure – *i.e.*, regardless of the underlying system scheduler. For example, Goyal *et al* [1] proposed a hierarchical scheduling framework for supporting different application classes in a multimedia system; Shin and Lee [2] further generalized this concept, advocating its use in embedded systems. Along the same lines, there has been a growing attention to building hierarchical real-time scheduling frameworks supporting different types of workloads [3]–[7].

A common characteristic (and/or inherent assumption) in the above-referenced, large body of prior work (which we emphasize is not exhaustive) is that the "clustering" (or grouping) of applications and/or schedulers under a common ancestor in the scheduling hierarchy is known *a priori* based on domain specific knowledge, *e.g.*, all applications with the same priority are grouped into a single cluster, or all applications requiring a particular flavor of scheduling (*e.g.*, periodic real-time EDF or RMS) are grouped into a single cluster managed by the desired scheduling scheme. Given such a *fixed* hierarchical structure, most of this prior body of work is concerned with the schedulability problem – namely deciding whether available resources are able to support this *fixed* structure.

Assuming that the structure of a hierarchical scheduler is known *a priori* is quite justified when all applications under that scheduler are part of the same system. It is also justified in small-scale settings in which the number of such applications (and the scale of the infrastructure supporting these applications) is small, and in which the set of applications to

be supported is rather static. None of these conditions hold in the emerging practices that fuel the use of virtualization technologies. In such settings, *inferring* the structure of a feasible, efficient hierarchical scheduler is *the* challenge.[1] This is precisely the problem we consider in this paper.

Scope and Contributions: Given a set of applications (tasks) each of which specified by minimal resource utilization requirements (SLAs), the problem we aim to address is that of mapping these tasks to the leaves of a forest, whose internal nodes represent virtual hosts and whose roots represent physical hosts.[2] The set of nodes under a host comprise a set of colocated *resource consumers*, which may be leaf nodes (application tasks) or internal nodes (virtual hosts). The allocation of resources by a host to the set of colocated resource consumers under its control is done using one of any number of schedulers. Without loss of generality, in this paper we assume that all leaves are periodic real-time tasks, and that the only scheduling strategy for hosts is Rate Monotonic Scheduling (RMS).[3]

As we hinted above, one way to formulate this mapping problem is to fix the structure of the scheduling hierarchy and infer the minimum capacity of the physical hosts at the root of the tree (or alternatively, check for schedulability of the scheduling hierarchy atop capacitated physical hosts). For instance, multiprocessor scheduling can be seen as restricting the forest to be a set of m 1-level-deep trees, and attempting to find an assignment of the tasks to the leaves that minimizes m (or that checks if there is a feasible/schedulable assignment for a given value of m). In such a formulation, there is also an inherent assumption that the SLAs associated with the various tasks (the leaves of the hierarchy) are immutable, in the sense that any feasible mapping must satisfy these SLAs "verbatim".

In this paper, we generalize this formulation in two substantial ways. First, we do not assume that the structure of the hierarchy is known *a priori*, but rather that it is to be inferred as part of the mapping problem. The scheduling hierarchy is not an input (constraint) but rather an output. Second, we do not assume that the SLAs requested by tasks (or offered by hosts) are immutable. In particular, we recognize that it could be the case that there are multiple, yet functionally equivalent, ways to express the resource requirements of a periodic real-time task. Thus, it is possible to *rewrite* SLAs as long as such rewriting is *safe*. Our ability to make such SLA transformations enables us to consider different colocation possibilities (and associated hierarchical structures), thus allowing us to explore alternative mappings. As we will show later in this paper, exploiting this flexibility yields significant efficiencies.

Paper Overview: The remainder of this paper is organized as follows. We start in Section II with some basic background and illustrative examples that crystalize the basic concepts (albeit at a very rudimentary level) underlying our work. In Section III, we introduce our basic type-theoretic model for periodic, real-time resource supply and demand (for hosts and tasks, respectively), along with necessary notation and basic definitions. In Sections IV and V, we present a series of safe transformations as exemplars of our notion of safe SLA rewrite rules. In light of the NP-hard nature of the problem, in Section VI we present a heuristic that allows us to greedily explore (and prune) the solution space of our mapping problem, through the application of various safe transformations in our arsenal. In Section VI-B, we present results of extensive experiments that demonstrate the efficacy of our approach. In Section VII, we review relevant related work. We conclude in Section VIII with closing remarks and current and future research directions.

II. Background and Illustrative Examples

Liu and Layland [8] provided the following classical result for the schedulability condition of n tasks, each of which requiring the use of a resource for C_i out of every T_i units of time, under RMS:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(\sqrt[n]{2} - 1) \tag{1}$$

Followup work, including that by Lehoczky *et al* [9] and Kuo and Mok [10] showed that by grouping tasks in k clusters such that the periods of tasks in each cluster are multiples of each other (*i.e.*, Harmonic), a tighter schedulability condition is possible – namely:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq k(\sqrt[k]{2} - 1) \tag{2}$$

Given a set of periodic tasks, it might be possible to obtain clusters of tasks with harmonic periods by manipulating the period T_i of some of the tasks in the set. To illustrate this point, consider the task set in Figure 1 (left). This task set consists of 5 clusters with a total utilization of $\frac{1}{4} + \frac{2}{9} + \frac{3}{17} + \frac{4}{34} + \frac{5}{67} = 0.840$, which exceed the 0.743 bound from Equation (1), and thus is not schedulable. Figure 1 (right) shows the result of applying the aforementioned transformation (of reducing the allocation periods) for some of the tasks. This transformed task set consists of one cluster with a total utilization of 0.890, which satisfies the bound from Equation (2), and thus is schedulable.

C	1	2	3	4	5
T	4	9	17	34	67

C	1	2	3	4	5
T	4	8	16	32	64

Fig. 1. Example of task transformations.

Obviously, for such a manipulation (as well as many others we will exemplify later in the paper) to be possible, we

[1] We note that even if all applications are catering to the same mission or if the number of applications is small, determining a hierarchical scheduling structure is not a trivial problem. Scheduling in a multiprocessor environment (an NP-hard problem in general) can be seen as an instance of this problem where the clusters are simply groups of applications assigned to the same processor.

[2] Our use of "physical" hosts is meant to imply that the resources available at these root nodes are beyond the purview/control of our framework – they represent the external supply of the resource(s) being managed, whether such supply is provided physically (*e.g.*, a dedicated CPU) or virtually (*e.g.*, a VM).

[3] While the analysis and transformations we provide in this paper are based on RMS, we emphasize that our framework and many of our results naturally extend to other types of schedulers.

must establish that it is *safe* to do so. For example, if the application underlying the periodic task allows the period T_i to be changed (e.g., the periodic task is a periodic zero-order hold feedback measurement process), then reducing the period T_i without reducing the requested resource time C_i is a safe transformation, since it ensures that the task is in effect getting a larger fraction of the resource.

To explain why the above transformation may not be safe (even if it results in an increase in the fraction of the resource alloted to the task), we note that if the period of the application underlying the periodic task cannot be changed (*e.g.*, it is synchronized with a physically-bound process such as the 30hz refresh rate of a frame buffer), then applying the above transformation may lead to missed deadlines. To illustrate this point, consider a task that requires $C = 1$ time units of the resource every period $T = 5$ time units. While reducing the allocation period for this task from $T = 5$ to $T' = 4$ would result in that task being alloted the resource for a larger fraction of time (25% as opposed to 20%), as shown in Figure 2, it is possible for that task to miss its original deadlines. In that figure, the upper row shows the periodic boundaries as originally specified $(T = 5)$, whereas the lower row shows a periodic allocation with $(T' = 4)$, with "X" marking the times when the resource is allocated.

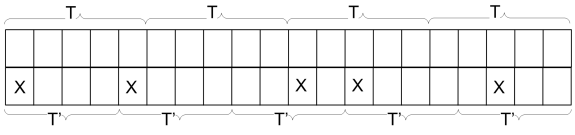

Fig. 2. Illustration: Reducing the allocation period may result in missed deadlines.

In the above example, the fact that the transformation we considered resulted (or may result) in missed deadlines does not mean that it cannot be used. In particular, if the SLA associated with the periodic task in question allows for some percentage of deadline misses, then if one is able to bound the deadline misses resulting from the transformation – and consequently show that the SLA is not violated – then the transformation is indeed safe.

The above illustrative examples provide a good handle on the premise and on the range of questions that need to be considered in support of our hierarchical schedule inference problem, and the ones we address in the remainder of this paper – namely: How could the SLAs of periodic, real-time applications be captured in a concise task model? What are examples of transformations that might be applied to such task models? How do these transformations affect task SLAs? Are these transformations compositional with respect to safety? Could this framework be presented to its potential users through familiar (and well developed) programming concepts such as typing, type checking, and type inference?

III. A Type-Theoretic Model for SLAs

As we established earlier, SLAs can be seen as encapsulators of the resources supplied by hosts (producers) and demanded by tasks (consumers). While this concept is generic enough for a wide variety of resources, in this section (and given the focus of this paper), we provide a specific model for SLAs – namely, one that supports periodic, real-time resource supply and demand.[4] We also provide the basic type-theoretic-inspired definitions that allow us to establish subtyping relationships between SLAs.

A. Periodic Supply/Demand SLA Types

Definition. A Service Level Agreement (SLA) type τ is defined as a quadruple of natural numbers (C, T, D, W), $C \leq T$, $D \leq W$, and $W \geq 1$, where C denotes the resource capacity supplied or demanded in each allocation interval T, and D is the maximum number of times such an allocation is not possible to honor in a window consisting of W allocation intervals.

As is common in the real-time literature, the above definition assumes that the periodic capacity could be allocated as early as the beginning of any interval (or period) and must be completely produced/consumed by the end of that same interval (*i.e.*, allocation deadline is T units of time from the beginning of the period).

The concept of SLA types is general enough to capture the various entities in our hierarchical scheduling framework. The following are illustrative examples.

An SLA of type $(1, 1, 0, 1)$ could be used to characterize a uniform, unit-capacity supply provided by a physical host. An SLA of type $(1, n, 0, 1)$, $n > 1$ could be used to characterize the fractional supply provided under a General Processor Sharing (GPS) model to n processes. An SLA of type $(k, k*n, 0, 1)$, $n > 1, k \geq 1$ could be used to characterize the fractional supply provided in a round robin fashion to n processes using a quantum k. In all of the above examples, the SLA type does not admit missed allocations (by virtue of setting $D = 0$). In the remainder of this paper, we refer to SLAs of the form $(C, T, 0, 1)$ – simply denoted using the shorthand (C, T) as *hard* SLAs, and we refer to the general quadruple form as *soft* SLAs. We also use the (C, T) notation when the consideration of D and W is immaterial.

An SLA of type $(1, 30)$ could be used to represent a task that needs a unit capacity $C = 1$ over an allocation period $T = 30$ and cannot tolerate any missed allocations (hard deadline semantics). An SLA of type $(1, 30, 2, 5)$ is similar in its periodic demand profile except that it is able to tolerate missed allocations (soft deadline semantics) as long as there are no more than $D = 2$ such misses in any window of $W = 5$ consecutive allocation periods.

In the above examples, there are two interpretations of what it means to satisfy an SLA type (C, T, D, W), depending on whether the allocations are defined using overlapping or non-overlapping time intervals. The following definitions – of what it means for a schedule to strongly or weakly satisfy an SLA –

[4]Readers familiar with real-time scheduling work should note that our periodic, real-time SLA model mirrors existing periodic task models in the literature (*e.g.*, [11]–[16]).

formalize these interpretations. First, we do so for SLAs of the form (C,T) (*i.e.*, those that do not admit missed allocations). Next, we generalize these definitions for general SLA types of the form (C,T,D,W).

B. Strong and Weak Satisfaction and Subtyping of Hard SLAs

Definition. A schedule α is a function from $\mathbb{N}$ to $\{0,1\}$.

$$\alpha : \mathbb{N} \rightarrow \{0,1\}$$

A schedule α is said to *weakly satisfy* (denoted by $\vDash_w$) or *strongly satisfy* (denoted by $\vDash_s$) an SLA type (C,T) if the resource is allocated for C units of time in non-overlapping (fixed) or overlapping (sliding) intervals of length T, respectively.[5]

Definition. $\alpha \vDash_w (C,T)$ iff for every $Q \geq 0$ and every $m = Q * T$

$$\alpha(m) + \cdots + \alpha(m+T-1) \geq C$$

Definition. $\alpha \vDash_s (C,T)$ iff for every $m \geq 0$

$$\alpha(m) + \cdots + \alpha(m+T-1) \geq C$$

Fact 1. *Strong satisfiability implies weak satisfiability: If* $\alpha \vDash_s (C,T)$ *then* $\alpha \vDash_w (C,T)$.

The set $[(C,T)]_w$ is defined as comprising all schedules that weakly satisfy (C,T). Similarily, $[(C,T)]_s$ is defined as comprising all schedules that strongly satisfy (C,T). Formally:

Definition. $[(C,T)]_w = \{\alpha : \mathbb{N} \rightarrow \{0,1\} \mid \alpha \vDash_w (C,T)\}$

Definition. $[(C,T)]_s = \{\alpha : \mathbb{N} \rightarrow \{0,1\} \mid \alpha \vDash_s (C,T)\}$

Fact 2. $[(C,T)]_s \subseteq [(C,T)]_w$.

We are now ready to introduce weak and strong SLA *subtyping* relationships (denoted by $\lhd_w$ and $\lhd_s$, respectively) as follows:

Definition. $(C,T) \lhd_w (C',T')$ iff $[(C,T)]_w \subseteq [(C',T')]_w$

Definition. $(C,T) \lhd_s (C',T')$ iff $[(C,T)]_s \subseteq [(C',T')]_s$

C. Strong and Weak Satisfaction and Subtyping of Soft SLAs

We now generalize the above definitions for (the more general) soft SLAs. To do so, we extend our definitions of schedules, strong and weak SLA satisfaction, and of SLA subtyping.[6]

Definition. $\mathbf{A}_{\alpha,C,T}$ is a function from $\mathbb{N}$ to $\{0,1\}$. $\mathbf{A}_{\alpha,C,T} : \mathbb{N} \rightarrow \{0,1\}$

$$\mathbf{A}_{\alpha,C,T}(m) = \begin{cases} 0 & \text{if } \alpha(m) + \cdots + \alpha(m+T-1) < C \\ 1 & \text{if } \alpha(m) + \cdots + \alpha(m+T-1) \geq C \end{cases}$$

A schedule α is said to *weakly satisfy* (denoted by $\vDash_w$) or *strongly satisfy* (denoted by $\vDash_s$) an SLA type (C,T,D,W) if the resource is allocated for C units of time in at least $W-D$ out of every W non-overlapping (fixed) or overlapping (sliding) intervals of length $W*T$, respectively.

Definition. $\alpha \vDash_w (C,T,D,W)$ iff for every $Q \geq 0$ and every $m = Q * W * T$

$$\mathbf{A}_{\alpha,C,T}(m) + \mathbf{A}_{\alpha,C,T}(m+T) + \cdots + \mathbf{A}_{\alpha,C,T}(m+(W-1)*T) \geq W - D$$

Definition. $\alpha \vDash_s (C,T,D,W)$ iff for every $Q \geq 0$ and every $m = Q * W$

$$\mathbf{A}_{\alpha,C,T}(m) + \mathbf{A}_{\alpha,C,T}(m+1) + \cdots + \mathbf{A}_{\alpha,C,T}(m+(W-1)) \geq W - D$$

Fact 3. *Strong satisfiability implies weak satisfiability: If* $\alpha \vDash_s (C,T,D,W)$ *then* $\alpha \vDash_w (C,T,D,W)$.

The set $[(C,T,D,W)]_w$ is defined as comprising all schedules that weakly satisfy (C,T,D,W). Similarily, $[(C,T,D,W)]_s$ is defined as comprising all schedules that strongly satisfy (C,T,D,W). Formally:

Definition. $[(C,T,D,W)]_w = \{\alpha : \mathbb{N} \rightarrow \{0,1\} \mid \alpha \vDash_w (C,T,D,W)\}$.

Definition. $[(C,T,D,W)]_s = \{\alpha : \mathbb{N} \rightarrow \{0,1\} \mid \alpha \vDash_s (C,T,D,W)\}$.

Fact 4. $[(C,T,D,W)]_s \subseteq [(C,T,D,W)]_w$

We generalize the notion of weak and strong *subtyping* for soft SLAs of the form (C,T,D,W) as follows:

Definition. $(C,T,D,W) \lhd_w (C',T',D',W')$ iff $[(C,T,D,W)]_w \subseteq [(C',T',D',W')]_w$.

Definition. $(C,T,D,W) \lhd_s (C',T',D',W')$ iff $[(C,T,D,W)]_s \subseteq [(C',T',D',W')]_s$.

IV. SLA Subtyping and Transformations

In this section, we present a set of SLA transformations that exemplify (and certainly do not exhaust) the range of scheduling results that could be "coded into" the type-theoretic framework that underlies our hierarchical scheduling inference and verification framework.[7] Each one of the transformations presented in this section is cast within a subtyping theorem. Intuitively, establishing a subtyping relationship between two SLAs implies that we can *safely* substitute one for the other (*e.g.*, transform the (supply-side) SLA of a virtual host from a given type to a supertype thereof, or transform the (demand-side) SLA of a task from a given type to a subtype thereof). We start with transformations based on strong subtyping, and follow that with transformations based on weak subtyping.

Theorem 1. $(C,T) \lhd_s (C',T')$ *iff one of the following conditions holds:*

1) $T \leq T'$ *and* $C \geq C'/K$ *where* $K = \lfloor T'/T \rfloor$.
2) $T \geq T'$ *and* $C \geq T - T' + C'$.

[5] Strong satisfiability can be seen as underscoring a pinwheel scheduling model [17], whereas weak satisfiability can be seen as underscoring the traditional strictly periodic model.

[6] While conceptually similar to the simpler hard SLA definitions, the consideration of missed allocations requires a more elaborate notation (which could be skipped on a first read).

[7] We emphasize that many results from the vast real-time scheduling theory literature could be translated into type transformations and leveraged in our framework.

Proof: [8]

Condition 1: [If] Observe that $K*T$ *non-overlapping* intervals will be completely overlapped by T', and necessarily provide an allocation of $K*C$, establishing the subtyping relationship. [Only If] Suppose $\alpha \vDash_s (C,T)$. This implies that for every $m \geq 0$: $\alpha(m)+\cdots+\alpha(m+T-1) \geq C$. Given that $K*T \leq T'$ and $K*C \geq C'$, it follows that

$$\alpha(m) + \cdots + \alpha(m + (K*T) - 1) \geq K*C \geq C'$$

Therefore $\alpha \vDash s(C',T')$ and $(C,T) \lhd_s (C',T')$.

Condition 2: [If] Since $T \geq T'$, it follows that one interval of length T must overlap with at least one sliding interval of length T' (shown in Figure 3). An allocation C in a single interval of length T should satisfy all sliding intervals of length T' that are fully overlapped by the interval T. In the worst case, we need to consider an adversarial allocation of the entire interval spanned by $T-T'$, leaving at least one interval with an unsatisfied allocation (first interval in Figure 3). To satisfy all the sliding intervals overlapping the interval T, we need an additional allocation of C'. Thus we need an allocation $C \geq T - T' + C'$.

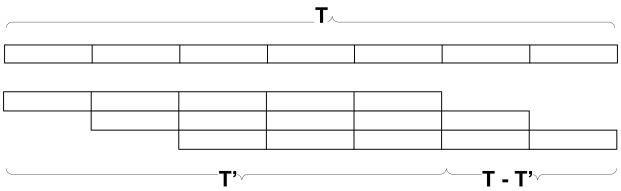

Fig. 3. Allocation throughout $T-T'$ leaves at least one unsatisfied interval.

[Only If] Suppose $\alpha \vDash_s (C,T)$. This implies that for every $m \geq 0$: $\alpha(m) + \cdots + \alpha(m+T-1) \geq C$. We also have $T \geq T'$ and $C \geq T - T' + C'$, which implies that $\alpha(m) + \cdots + \alpha(m+T-1) \geq C$. Let $\gamma = T - T'$, then

$$\alpha(m) + \cdots + \alpha(m + T' - 1) \geq C - \gamma \geq C'$$

Hence $\alpha \vDash s(C',T')$ and $(C,T) \lhd s(C',T')$. ■

Theorem 2. $(C,T) \lhd_w (C',T')$ *if one of the following conditions holds:*

1) $T \leq T'/2$ *and* $C \geq C'/(K-1)$ *where* $K = \lfloor T'/T \rfloor$.
2) $T > T'$ *and* $C \geq T - (T' - C')/2$.
3) $T'/2 < T \leq T'$ *and* $T - (T' - C')/3 \leq C$.

Proof:

Condition 1: $T \leq T'/2$ implies that $K \geq 2$. According to Lemma 1 (see appendix), an interval of length T must overlap with at least $K-1$ fixed intervals of length T'. These $K-1$ intervals provide an allocation of $(K-1)*C'$, enough for interval T. Thus, $(K-1)*C' \geq C$ and $C' \geq C/(K-1)$.

Condition 2: Consider any interval I' of length T'. Since $T > T'$, either I' will be completely overlapped by an interval I of length T, or it will be overlapped by two intervals of length T (as shown in Figure 4). For any interval I of length T, denote the left and right boundaries of I using $l(I)$ and $r(I)$, respectively. Let x be the offset of $l(I')$ from $l(I_1)$ and y be the

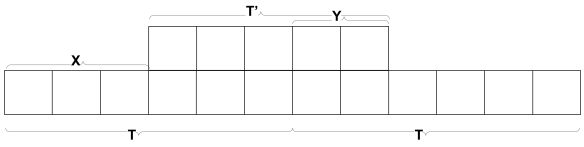

Fig. 4. T' is overlapped by two intervals of size T

offset of $r(I')$ from $l(I_2)$. We observe that $(T'-y)+x = T$, leading to $C \geq 1/2(x+(T-y)+C)$ and $C \geq T-(T'-C')/2$ as a sufficient condition.

Condition 3: Consider any interval I' of length T'. Since $T'/2 < T \leq T'$, I' will overlap with either two or three intervals of length T. The case in which I' overlaps two intervals of length T follows from Condition 2, resulting in $C \geq (2T' - T + C')/2$. The case in which I' overlaps three intervals I_1, I_2, and I_3 of length T is shown in Figure 5. Let x be the offset of $l(I')$ from $l(I_1)$ and y be the offset

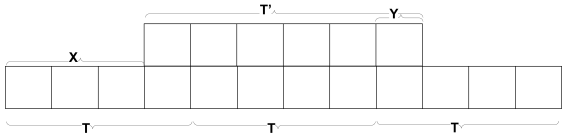

Fig. 5. T' is overlapped by three intervals of size T

of $r(I')$ from $l(I_3)$. We observe that $(T'-y)+x = 2T$. Thus, a sufficient condition is $C \geq 1/3(x+(T-y)+C)$ and $C \geq T-(T'-C')/3$. Thus a bound for both cases is the maximum of the two bounds, namely $C \geq T-(T'-C')/3$. ■

Conjecture 1. *The converse of Theorem 2 holds. That is, if* $(C,T) \lhd_w (C',T')$ *then*

1) $T \leq T'/2$ *and* $C \geq C'/(K-1)$ *where* $K = \lfloor T'/T \rfloor$.
2) $T > T'$ *and* $C \geq T - (T' - C')/2$.
3) $T'/2 < T \leq T'$ *and* $T - (T' - C')/3 \leq C$.

Proof: We provide a proof for condition 1.

Condition 1: Suppose $\alpha \vDash_w (C,T)$. This implies that for every $Q \geq 0$ and every $m = Q*T$: $\alpha(m)+\cdots+\alpha(m+T-1) \geq C$. Given that $K*T \leq T'$ and $T \leq T'/2$, we have $K \geq 2$ and $(K-1)*T < T'$. Given that $(K-1)*C \geq C'$, we get:

$$\alpha(m) + \cdots + \alpha(m + ((K-1)*T) - 1) \geq K - 1 * C \geq C'$$

Therefore $(C,T) \lhd_w (C',T')$. ■

Theorem 3. $(C,T,D,W) \lhd_s (C',T',D',W')$ *if one of the following conditions holds:*

1) *(*$T \leq T'$ *and* $C \geq C'/K$*) and* $D \leq D'$ *and* $W \geq D*W'/D'$*, where* $K = \lfloor T'/T \rfloor$.
2) *(*$T \geq T'$ *and* $C \geq T - T' + C'$*) and* $D \leq D'$ *and* $W \geq D*W'/D'$.

Proof:

[8] This proof (as well as others in this paper) was mechanically verified using the theorem prover of [18].

Condition 1: The proof for the bracketed part of the conjuntion is identical to that under Condition 1 of Theorem 1. For the remaining part, Lemma 2 (see appendix), which states that every missed interval of length T' corresponds to missing an interval of length T, implies that $D \leq D'$ must hold.
Condition 2: The proof for the bracketed part of the conjuntion is identical to that under Condition 2 of Theorem 1. For the remaining part, Lemma 3 (see appendix), which states that every missed interval of length T corresponds to missing an interval of length T', implies that $D \leq D'$ must hold. ■

Conjecture 2. *The converse of Theorem 3 holds. That is, if* $(C, T, D, W) \lhd_s (C', T', D', W')$ *then*

1) *(*$T \leq T'$ *and* $C \geq C'/K$*) and* $D \leq D'$ *and* $W \geq D * W'/D'$*, where* $K = \lfloor T'/T \rfloor$.
2) *(*$T \geq T'$ *and* $C \geq T - T' + C'$*) and* $D \leq D'$ *and* $W \geq D * W'/D'$.

Theorem 4. $(C, T, D, W) \lhd_w (C', T', D', W')$ *if one of the following conditions holds:*

1) *(*$T \leq T'/2$ *and* $C \geq C'/(K-1)$*) and* $D \leq D'/2$ *and* $W \geq D * W'/D' * (K+1)$ *where* $K = \lfloor T'/T \rfloor$.
2) *(*$T > T'$ *and* $C \geq T - (T' - C')/2$*) and* $D \leq D'/2K$ *and* $W \geq D * W'/D' * (K+1)$ *where* $K = \lfloor T'/T \rfloor$.
3) *(*$T'/2 < T \leq T'$ *and* $T - (T' - C')/3 \leq C$*) and* $D \leq D'/2$ *and* $W \geq 2 * D * W'/D'$.

Proof: We use the fact from Lemma 4 (see appendix) that $D/W \leq D'/W'$ is a necessary condition.
Condition 1: The proof for the bracketed part of the conjuntion is identical to that under Condition 1 of Theorem 2. For the remaining part, we note that since missed deadlines might be stacked at the end of one window and at the beginning of the next contributing to a window of size W, it follows that $D \leq D'/2$. Also, since $K+1$ consecutive intervals of length T will span one interval of length T', it follows that every missed interval of length T out of $K+1$ intervals will result in missing an interval of length T'. Thus, $W \geq (K+1) * (D * W/D')$ must hold.
Condition 2: The proof for the bracketed part of the conjuntion is identical to that under Condition 2 of Theorem 2. For the remaining part, in the worst case, missing an interval of length T results in missing up to $(K+1) * T'$ intervals, where $K = \lfloor T'/T \rfloor$. Thus $D \leq D'/(K+1)$ must hold as well as $W \geq (K+1) * (D * W/D')$. However, since missed deadlines might be stacked at the end of one window and at the beginning of the next contributing to a window of size W, it follows that $D \leq D'/2(K+1)$ must hold.
Condition 3: The proof for the bracketed part of the conjuntion is identical to that under Condition 3 of Theorem 2. For the remaining part, the proof is similar to that in Condition 2 by taking $K = \lfloor T'/T \rfloor$ and consequently $K = 1$. Thus, $W \geq (K+1) * (D * W/D')$ must hold. ■

Conjecture 3. *The converse of Theorem 4 holds. That is, if* $(C, T, D, W) \lhd_w (C', T', D', W')$ *then*

1) *(*$T \leq T'/2$ *and* $C \geq C'/(K-1)$*) and* $D \leq D'/2$ *and* $W \geq D * W'/D' * (K+1)$ *where* $K = \lfloor T'/T \rfloor$.
2) *(*$T > T'$ *and* $C \geq T - (T' - C')/2$*) and* $D \leq D'/2K$ *and* $W \geq D * W'/D' * (K+1)$ *where* $K = \lfloor T'/T \rfloor$.
3) *(*$T'/2 < T \leq T'$ *and* $T - (T' - C')/3 \leq C$*) and* $D \leq D'/2$ *and* $W \geq 2 * D * W'/D'$.

V. Additional Transformations

Having characterized some basic notions of subtyping for both overlapping and non-overlapping SLA types, for our experimental evaluation – which we consider in the next sections – we will focus on non-overlapping SLA types. Thus, in this section, we present additional transformations that allow for safe rewriting of such types.

Theorem 5. *Let* $\tau = (KC, KT)$ *be an SLA type for some* $K \geq 1$ *and* $\tau' = (C, T)$ *be a host-provided SLA type. Then* $\tau' \lhd_w \tau$.

Proof: One can observe that one interval of τ will contain K intervals of τ' with each interval providing C computation time. Thus τ is satisfied. ■

Definition. $(C', T') \lhd_{w,a,b} (C, T)$ where a is the bound on the missed deadlines over b intervals of length T.

Theorem 6. *Let* $\tau = (C, T)$ *be an SLA type, and* $\tau' = (C', T')$ *be a host-provided SLA type, where* $T' = KT$ *for some* $K > 1$ *then:*

1) *If* $0 \leq C' < K * (C-1) + 1$ *then* $\tau' \lhd_{w,a,b} \tau$*, where* $a = K * T$ *and* $b = K$*. Moreover in such a case,* α *will miss at least one allocation deadline every* K *intervals.*
2) *For every* $J \in \{1, \ldots, K-1\}$*, if*

$$K * (C-1) + (J-1) * (T - (C-1)) + 1 \leq C' < K * (C-1) + (J) * (T - (C-1)) + 1$$

then $\tau' \lhd_{w,a,b} \tau$*, where* $a = (K - J)$ *and* $b = K$
3) *For* $J = K$*, if*

$$K * (C-1) + (J-1) * (T - (C-1)) + 1 \leq C' \leq T'$$

then $\tau' \lhd_w \tau$.

Proof:
Condition 1: Since $T' = KT$, we have K intervals. No matter how the distribution of C' is going to be over the K intervals, it is always the case that the resource allocation will be less that the KC units needed over the K intervals. Thus we conclude that the schedule will always include at least one interval with a missed allocation.
Condition 2: Consider the left inequality in the conjunction, *i.e.,* $K * (C-1) + (J-1) * (T - (C-1)) + 1 \leq C'$. Assume that there are $(K - J)$ unsatisfied intervals with at most $(K - J) * (C-1)$ allocation units. Thus, there should be J satisfied intervals containing at least $C' - (K-J) * (C-1)$ allocation units. Therefore we have:

$$(K-J) * T < C' - (K-J) * (C-1)$$

$$C' > (K-J) * T + (K-J) * (C-1)$$

$$C' > J * (T - (C-1)) + K * (C-1)$$

Since $(K-J)*T$ is the total time in all the satisfied intervals, it follows that the total time in the satisfied interval is strictly less than the allocations in the satisfied interval – a contradiction. Therefore,

$$C' \leq J*(T-(C-1))+K*(C-1)$$
$$C' < J*(T-(C-1))+K*(C-1)+1.$$

Now, consider right inequality in the conjunction, *i.e.,* $C' < K*(C-1)+(J)*(T-(C-1))+1$. If $C' < K*(C-1)+(J-1)*(T-(C-1))+1$, then there exists a schedule such that the number of satisfied interval is strictly less than J. Let $C' = K*(C-1)+(J-1)*(T-(C-1)) < K*(C-1)+(J-1)*(T-(C-1))+1$. We can simply distribute $C-1$ allocation units over K intervals such that none of the intervals are satisfied. Furthermore, we distribute $T-(C-1)$ allocation units over $J-1$ windows, thus completely filling $J-1$ intervals with T allocation units. Thus, we end up with at least $J-1$ satisfied intervals.

Condition 3: To guarantee all intervals, in the worst case, we need to have $K-1$ intervals filled with T allocation units. In addition, we need to have at least C allocation units in the last interval. By substituting K for J in the above equation we get:

$$\begin{aligned} C &\geq K*(C-1)+(K-1)*(T-(C-1))+1 \\ &K*(C-1)+(K-1)*T-(K-1)*(C-1)+1 \\ &(C-1)+(K-1)*T+1 \\ &C+(K-1)*T. \end{aligned}$$

■

Theorem 7. *Let $\tau=(C,T)$ be an SLA type and $\tau'=(C,T')$ be a host-provided SLA type, where $(T+C)/2 < T' < T$ and $C \leq T'$. If $m=\mathrm{lcm}(T,T')/T$, and $n=\mathrm{lcm}(T,T')/T'$ where lcm is the least common multiple, then*

1) *We can guarantee at least $s=n-m+1$ satisfied intervals out of total m intervals.*
2) *We can guarantee at least $l=\lceil \frac{m}{(C+1)} \rceil$ satisfied intervals out of the total m intervals.*

We can bound the number of missed deadlines every m intervals to be $a=m-\max(s,l)$. Therefore $\tau' \lhd_{w,a,b} \tau$ where $a=m-\max(s,l)$ and $b=m$.

Proof:

Condition 1: Since $T > T'$, we observe that the number of satisfied intervals of length T is at least equal to the number of completely overlapping intervals of length T'. Let $f(T,T')$ be the number of completely overlapped unique intervals of τ in τ', then

$$f(T,T') = f(\frac{T}{\gcd(T,T')}, \frac{T'}{\gcd(T,T')})$$

where $\gcd$ is the greatest common divisor of T and T'. Thus $\frac{T}{gcd(T,T')}$ and $\frac{T'}{gcd(T,T')}$ are prime with respect to each other. Let $R=\{K*T' \mod T \mid 1 \leq K \leq T\}$, then $|R|=T$.

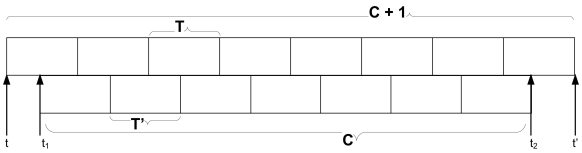

Fig. 6. Adversary trying to miss maximum possible deadlines

Furthermore $R=\{1,\ldots,T\}$.[9] Since the remainders in R are unique, let us mark the remainders on a circle starting from 1 and ending at T. We observe that every remainder that is marked at the region starting from 1 to $\frac{T}{\gcd(T,T')}-\frac{T'}{\gcd(T,T')}$ will not pass the cycle ending at T because $T > T'$. In addition, the interval that starts at position T will also not pass the cycle, therefore the total number of overlapping intervals is:

$$\begin{aligned} &\frac{T'}{\gcd(T,T')}-\frac{T}{\gcd(T,T')}+1 \\ &=\frac{\mathrm{lcm}(T,T')}{T'}-\frac{\mathrm{lcm}(T,T')}{T}+1 \\ &=n-m+1. \end{aligned}$$

Condition 2: Since $T' < T$, the first interval will always be satisfied. To bound the number of missed allocations, we assume an adversary whose purpose is to maximize the number of missed allocations by allocating the resource to intervals that are already satisfied. Under such conditions, we prove that every $C+1$ intervals of length T will contain at least one satisfied interval.

Consider any schedule, assume it has a sub-sequence S of $C+1$ unsatisfied intervals of length T denoted by $T_1,\ldots,T_{c+1}$ where $T_1=[t,t+T],\ldots,T_{c+1}=[t+CT,t']$. Exactly C intervals of length T' are completely contained in S denoted as $T'_1,\ldots,T'_c$. Let T' start at t_1 and T_c end at t_2. The total computation time in $[t_1,t_2]$ is at most equal to the total computation time in $[t,t']$. The total computation time scheduled in $[t_1,t_2]=C*C=C^2$. Since all the intervals T_i in S are unsatisfied, the total computation time scheduled for each interval of length T can be at most $C-1$. Therefore the total computation time in $[t,t'] \leq (C+1)*(C-1)=C^2-1$. Contradiction. S must contain some satisfied intervals. To generalize, we have m intervals of length T. Since every $C+1$ intervals of length T will contain at least one satisfied interval, we can bound the number of missed allocations to be at most equal to $m-\lceil \frac{m}{(C+1)} \rceil$. ■

We also define a two step transformation of an SLA type by applying the transformation in Theorem 6 followed by applying the transformation in Theorem 7.

Theorem 8. *Let $\tau_1=(C_1,T_1)$, and $\tau_2=(C_2,T_2)$ such that $\tau_2 \lhd_{w,a,b} \tau_1$ by applying the transformation in Theorem 6. Let $\tau_3=(C_3,T_3)$ such that $\tau_3 \lhd_{w,x,y} \tau_2$ by applying the*

[9]For simplicity, we choose to enumerate from 1 to T instead of from 0 to $T-1$.

transformation in using Theorem 7. Then $\tau_2 \lhd_{w,c,d} \tau_1$ *where* $c = (b * x + (y - x) * a)$ *and* $d = (b * y)$.

Proof: $\tau_2 \lhd_{w,a,b} \tau_1$ will miss at most a allocations over b intervals. $\tau_3 \lhd_{w,x,y} \tau_2$ will miss at most x allocations over y intervals. Every missed allocation in τ_2 corresponds to the failure of satisfying an entire window b in τ_1, and every satisfied window in τ_2 corresponds to missing at most a allocations in τ_2. Thus at most, the total number of missed allocations over a window $d = (b*y)$ is the sum of all possible missed allocations $c = (b * x + (y - x) * a)$. ■

Theorem 9. *Applying the transformations in Theorems 6 and 7, in this order, is equivalent to applying the transformations in Theorems 7 and 6, in this order. That is, the two transformations commute.*

Proof: Theorem 8 highlights results of applying Theorem 6 followed by Theorem 7. We would like to show that the results for Theorem 7 followed by Theorem 6 are equal.

$\tau_2 \lhd_{w,a,b} \tau_1$ will miss at most a allocations over of b intervals. $\tau_3 \lhd_{w,x,y} \tau_2$ will miss at most x allocations over y intervals. Every missed allocation in τ_2 corresponds to the failure of satisfying an entire window b in τ_1, and every satisfied window in τ_2 corresponds to missing at most a allocations in τ_2. Thus at most, the total number of allocations over a window $d = (b * y)$ is the sum of all possible missed allocations $c = (b * x + (y - x) * a)$. ■

Other two step transformations are possible – *e.g.,* applying the transformation in Theorem 5 followed by Theorem 6.

VI. Inference of Efficient and Safe Colocations

With the underpinnings of our framework as well as the set of SLA transformations we have developed (as exemplars) in place, in this section we show how to leverage our framework to map a set of real-time, periodic tasks into a hierarchical scheduling structure. In particular, we present sample results from extensive simulation experiments that demonstrate the efficiencies that can be achieved through judicious colocation of periodic tasks under separate physical/virtual hosts comprising a hierarchical scheduler. Before doing so, we start by describing the specific strategy we use to come up with a colocation arrangement.

A. Mapping Heuristic

Independent of the flexibility enabled through safe SLA transformations, the crux of the problem at hand is that of identifying an efficient mapping of tasks to *multiple* hosts (each of which underscoring a resource and associated scheduler). We say "efficient" as opposed to optimal because multi-processor real-time scheduling has been shown to be NP-Hard [19], [20] (and our problem by reduction is also NP hard), thus requiring the use of heuristics. Many such heuristics (approximations) have been proposed in the literature (*e.g.,* based on the use of a bin packing or greedy strategy).

Our safe SLA transformations provide us with another degree of freedom (*i.e.,* another dimension in the search space): Rather than finding the best packing of a set of tasks with fixed SLA requirements (the original NP-hard problem), we have the flexibility of safely manipulating the SLAs with the hope of achieving a better packing. Towards that end, we have implemented heuristic algorithms that utilize Breadth First Search (BFS) and Depth First Search (DFS) techniques to explore the solution search space.

Our (BFS or DFS) heuristic starts with a preprocessing stage, in which we generate all possible transformations for each task (using our arsenal of safe transformations). Next, it proceeds by setting up the search space (tree or forest) of all the alternative task sets that could be colocated. Finally, it proceeds to explore that search space with the aim of finding a feasible transformation.

In the worst case, our heuristic may end up searching the entire solution space, which is obviously impractical.[10] To manage the exponential nature of the search space, our heuristic utilizes two optimization (pruning) strategies.

Our first optimization strategy adopts an early-pruning approach: at each stage of our search, if the aggregate utilization (demanded SLA) of the tasks under consideration thus far (whether these tasks are transformed or not) is greater than the capacity of the host (supplied SLA), then we prune that branch of the tree on the assumption (not necessarily correct) that a feasible solution cannot exist down that path.

Our second optimization adopts a smaller-degree-first approach: we build the search space (tree) by greedily starting with tasks that have the smallest number of transformations. This ensures that when pruning is applied (using the above strategy) we are likely to maximize the size of the pruned subspace. This optimization strategy has been shown to be quite effective in reducing the solution search space for network embedding problems [21].

B. Experimental Evaluation

We evaluate the efficiency of our proposed mapping technique by comparing the schedulability of a workload (comprising a set of synthetically-generated real-time period tasks) with and without the application of our safe SLA transformations. We do so for both uniprocessor (tree) and multiprocessor (forest) settings.

Uniprocessor Set-up and Results: Tasks are created by initially generating a period T based on a uniform distribution between (T_{min}, T_{max}), followed by generating a periodic resource (CPU) demand C uniformly at random between ($\alpha*T$ and $\beta*T$), such that the generated tasks are schedulable under RMS using the feasibility test of Liu and Layland [8]. Next, we perturb the task set by modifying the values of Cs and Ts such that the task set is no longer schedulable under RMS. These perturbed (C, T) values constitute the SLAs of the tasks in the workload. For each such an unschedulable workload, we

[10] In case it is not clear, the colocation mapping techniques in this paper are not meant to be used in an on-line fashion. As a service supporting the use of virtualization infrastructures, these mappings are meant to be applied over long time scales (and over a large number of tasks to benefit from the efficiencies of scale).

Overload	SuccessRate	ErrMargin	Unmodified
0 - 7%	52.27%	1.28%	57.62%
7% - 14%	26.27%	1.09%	57.94%
14% - 21%	7.73%	0.73%	61.61%
21% - 28%	1.47%	0.30%	59.09%
28% - 35%	0.13%	0.09%	60.00%

TABLE I
RESULTS FOR HARD SLAS ON A UNIPROCESSOR.

Overload	SuccessRate	ErrMargin	Unmodified
0 - 7%	36.80%	1.28%	57.40%
7% - 14%	20.53%	0.87%	62.16%
14% - 21%	5.33%	0.74%	65.91%
21% - 28%	0.13%	0.09%	75.00%

TABLE II
RESULTS FOR HARD SLAS ON A UNIPROCESSOR. ($\alpha = 0.02$ AND $\beta = 0.02$)

define the *overload* as the difference between the aggregate SLA demand of all tasks in the workload (*i.e.*, the sum of the requested fractions of the resource) and the utilization bound of RMS for the same task set.

To evaluate our mapping technique, we apply our heuristic to identify a possible set of safe SLA transformations (if any) that would make the workload schedulable as determined by the RMS test, reporting whether or not we are successful in transforming an unschedulable task set to a schedulable one, the number of tasks that were transformed, and the specific transformations that were applied. Each experiment consisted of evaluating 25 random task sets (using the same settings) to obtain a success ratio. To ascertain statistical significance by reporting a margin of error (ErrMargin), we report the average results of running 40 independent experiments.

Table I present results for $\alpha = 0.15$ and $\beta = 0.25$ with $D = 0$ and $W = 1$, *i.e.*, under a hard SLA setting whereby no misses are allowed. The results show that as the overload (Overload) increases, the chances of finding a feasible transformation (SuccessRate) decreases. An interesting observation from these results is that the majority of tasks (Unmodified) do not need transformations. Similar observations were noted for other settings of α and β as shown in Tables II and III.

In the next set of experiments, we report on the effectiveness of our transformations when soft SLAs are considered (*i.e.*, with arbitrary D and W. Notice that in this case, we are able to apply a wider range of transformations (compared to the previous hard SLA setting). We use the same model described above to generate C and T. For each task, we choose W uniformly at random between W_{min} and W_{max}, and choose D uniformly at random between 0 and $\theta * W$.

Since RMS' schedulability test does not allow us to account for the flexibility resulting the soft SLAs, we use the following empirical approach to determine if the task set (without transformations) is schedulable. We initially use the feasibility test to check whether the generated tasks are schedulable, if they are not, then we simulate task execution using a *greedy* RMS to validate whether the tasks are schedulable. The *greedy* RMS tries to schedule tasks following the regular RMS, but once a minimum $W - D$ is satisfied, the task does not get scheduled again during the window W. If the task set is not schedulable, we then apply our transformations and check whether they are schedulable using only the schdulability test of RMS – thus providing a conservative measure of success since no task in our task set would miss a single deadline, even though such misses are allowed under soft SLAs.[11]

Overload	SuccessRate	ErrMargin	Unmodified
0 - 7%	36.93%	1.10%	57.94%
7% - 14%	17.33%	0.95%	62.65%
14% - 21%	2.80%	0.41%	64.38%
21% - 28%	0.27%	0.12%	75.00%

TABLE III
RESULTS FOR HARD SLAS ON A UNIPROCESSOR. ($\alpha = 0.005$ AND $\beta = 0.195$)

Overload	SuccessRate	ErrMargin	Unmodified
0 - 7%	90.00%	0.64%	54.71%
7% - 14%	66.53%	1.79%	48.01%
14% - 21%	48.93%	1.24%	43.97%
21% - 28%	28.93%	0.99%	39.13%
28% - 35%	17.07%	0.90%	36.83%
35% - 42%	7.06%	0.64%	27.73%
42% - 49%	4.27%	0.48%	31.82%
49% - 56%	2.93%	0.36%	29.07%
56% - 63%	2.00%	0.31%	32.20%
63% - 70%	0.93%	0.21%	17.39%
70% - 77%	0.67%	0.19%	21.43%
77% - 84%	0.80%	0.20%	14.29%
84% - 91%	0.13%	0.09%	33.33%

TABLE IV
RESULTS FOR SOFT SLAS ON A UNIPROCESSOR.

Table IV presents results for $\alpha = 0.15$ and $\beta = 0.25$, with random D and W as described above. Again, these results show that as the overload increases, the chances of finding a feasible transformation decreases. However, having tasks that allow some flexibility in missing deadlines allowed us to find feasible transformations under higher overloads. Another interesting observation is that (unlike the case with hard SLAs), as the overload increases more tasks are amenable to transformations.

Multiprocessor Set-up and Results: Many multiprocessor scheduling algorithms and associated schedulability tests have been proposed in the literature (*e.g.*, [22]–[24]). For our purposes, we use the schedulability condition from Andersson *et al* [24], which rely on a global static priority scheduling algorithm – namely that any number of arbitrary tasks can be scheduled on m identical multiprocessors if $U(t) < m^2/3m - 2$, where $U(t)$ is the sum of the utilization of the set of tasks. However if the tasks were harmonic, then the bound would be $U(t) < m^2/2m - 1$.

In our experiments, we use Andersson *et al*'s schedulability condition to determine the number of processors needed to

[11]We note that checking for schedulability using *greedy* RMS is likely to significantly improve our results (but it would also increase our search space since not much pruning can be done in that case).

satisfy the SLA demand of the task set (ProcBefore). Next, we apply our mapping technique to identify an efficient packing into a smaller number of processors (ProcAfter) that makes use of the safe SLA transformations in our arsenal. Unlike the uniprocessor case, and given the much larger search space in a multiprocessor setting, we modified our system so it would stop after a certain percentage reduction (*e.g.*, a threshold of 50%) in the number of multiprocessors needed to support the task set is achieved.

Tables V and VI present results for $\alpha = 0$ and $\beta = 0.5$ and $\alpha = 0$ and $\beta = 0.75$, respectively. T is generated uniformly at random between T_{min} and T_{max}, and C is generated uniformly at random between $\alpha * T$ and $\beta * T$ with $W = 1$ and $D = 0$ (hard SLA setting). As shown, with the use of transformations, we are able to decrease the number of processors needed to support the workload's SLA by a factor of two, matching our target threshold.

ProcBefore	SuccessRate	AvgTasks	ProcAfter	Unmodified
3	94.89%	4.67	2.00	78.03%
4	96.67%	5.84	2.03	76.51%
5	99.33%	6.92	2.72	73.90%
6	99.78%	8.25	3.04	69.81%
7	100.00%	9.64	3.48	67.60%
8	100.00%	10.88	4.08	68.00%
9	100.00%	12.10	4.44	65.99%
10	100.00%	13.63	5.05	65.62%
11	100.00%	15.10	5.48	64.00%
12	100.00%	16.28	6.1	64.39%
13	100.00%	17.71	6.6	64.06%
14	100.00%	18.82	7.12	64.65%
15	100.00%	20.36	7.65	64.07%
16	100.00%	21.40	8.24	64.06%

TABLE V
RESULTS FOR HARD SLAS ON MULTIPROCESSORS ($\beta = 0.5$)

ProcBefore	SuccessRate	AvgTasks	ProcAfter	Unmodified
3	92.00%	3.16	2.00	65.81%
4	92.67%	3.97	2.06	70.83%
5	98.22%	4.86	2.63	67.43%
6	98.67%	5.69	3.17	64.47%
7	99.78%	6.54	3.46	63.25%
8	100.00%	7.49	4.16	65.09%
9	100.00%	8.25	4.54	62.19%
10	100.00%	9.22	5.16	63.19%
11	99.78%	10.02	5.62	62.44%
12	99.78%	10.83	6.1	61.00%
13	100.00%	11.79	6.72	61.37%
14	100.00%	12.82	7.27	62.56%
15	100.00%	13.6	7.74	61.79%
16	100.00%	14.49	8.54	63.52%

TABLE VI
RESULTS FOR HARD SLAS ON MULTIPROCESSORS ($\beta = 0.75$)

VII. RELATED WORK

Numerous previous studies dealt with hierarchical scheduling frameworks [3]–[7]. Regehr and Stankovic [3] introduced a hierarchical scheduling framework in support of various types of guarantees. They use rewriting rules to convert a guarantee provided under a specific scheduling algorithm to a guarantee provided under another. This notion of rewriting is different from ours as it does not accommodate workload transformations.

Shin and Lee [4] present a compositional real-time scheduling framework based on workload bounding functions, and resource bounding functions. They utilize a tree data structure, where a child scheduling system is the immediate descendant of the parent scheduling system, and the parent scheduling system is the immediate ancestor of the child scheduling system. Their model assumes that the parent and children scheduling systems can utilize different types of scheduling algorithms. Under their framework any given system composed of a workload, resources, and a scheduling algorithm, will be schedulable if the minimum resource curve bounds the maximum workload curve. This model is extended further in [5] to include context switching overhead and incremental analysis.

Our work complements these models, which did not focus (or consider) the problem of inferring the scheduling hierarchy (which set of tasks to be colocated under a common scheduler). We believe that this capability is crucial, especially when coupled with the possibility of safely transforming the workload characteristics, which is a novel aspect of our work.

The idea of transforming task periods for improving schedulability is not new. It was highlighted in [25], [26], where the authors defined a period transformation method that involves halving the C and T elements of the periodic task specification. The purpose of this transformation was to increase the priority of a task under RMS. In our work, we consider much more general transformations targeting not only hard, but also soft deadline semantics, and which are derived for overlapping as well as traditional, non-overlapping intervals.

The work by Buttazzo et al [15] and its generalization in [16] present an elastic work model based on a tasks defined using a tuple $(C, T, T_{min}, T_{max}, e)$ where T is the period that the task requires, while T_{min} and T_{max} define the max and min periods that a task can accept. Our transformations allows us to serve workloads under completely different (C, T) server supplied resources.

As we have emphasized throughout, the real-time scheduling literature is huge, in particular as it relates to scheduling algorithms and task models [8], [11]–[16], [27]. We view our contributions mostly as providing a layer above "scheduling" – a layer that leverages the many results in the literature to enable SLA transformations.

VIII. CONCLUSION

The value proposition of virtualization technologies is highly dependent on our ability to identify judicious mappings of physical resources to virtualized instances that could be acquired and consumed by applications subject to desirable performance (*e.g.*, QoS) bounds. These bounds are often spelled out as a Service Level Agreement (SLA) contract between the resource provider (hosting infrastructure) and the resource consumer (application task). By necessity, since infrastructure

providers must cater to very many types of applications, SLAs are typically expressed as fixed fractions of resource capacities that must be allocated (or are promised) for *unencumbered* use. That said, the mapping between "desirable performance bounds" and SLAs is not unique. Indeed, it is often the case that *multiple* SLA expressions might be functionally equivalent with respect to the satisfaction of these performance bounds. Having the flexibility to transform SLAs from one form to another in a manner that is *safe* would enable hosting solutions to achieve significant economies of scale.

This paper presented a particular incarnation of this vision – a vision that we are also pursuing in other settings – targeting the colocation of periodic real-time systems. The particular approach we advocate (and the specific tools we have developed) rely on a *type-theoretic modeling* of SLAs and transformations thereof. In that regard, we have presented a specific type-theoretic model for the specification of the SLAs of periodic, real-time resource supply and demand (for hosts and tasks, respectively), along with a number of provably-safe SLA type transformations. Using that formal framework, we have developed a methodical approach for the inference of efficient and safe assignment of colocated periodic, real-time tasks to the physical and virtual hosts that constitute a hierarchical scheduler. Our experimental results support our expectation that the flexibility afforded from safe SLA transformations has the potential to yield significant savings.

The work presented in this paper underscores a number of thrusts of our broader research agenda, which includes: leveraging colocation for efficient management of virtual machine (VM) cloud infrastructures [28], using type-theoretic formulations in the modeling and analysis of network compositions [29], [30], and the development of automated and semi-automated tools based on light-weight formalisms [18]. With respect to the specific framework presented in this paper, our immediate future plans include enriching our SLA type hierarchy through the introduction of additional semantics (*e.g.*, the consideration of other real-time schedulers), and the codification of prior results in the vast real-time scheduling literature into type transformations that could be used by our mapping service.

ACKNOWLEDGEMENTS

We would like to thank Debajyoti Bera, Gabriel Parmer, Andrei Lapets, and Jorge Londoño for their help with and feedback on various aspects of this work.

APPENDIX

Lemma 1. *Given the periods T and T' such that $T \leq T'/2$. Then an interval of length T' would contain at least $(K-1)$ intervals of length T where $K = \lfloor T'/T \rfloor$.*

Proof: Figure 7 highlights the existence of a schedule such that T' overlaps $(K-1) * T$ intervals where $T' = 7$ and $T = 3$ and $K = 2$. Assume the existance of a schedule

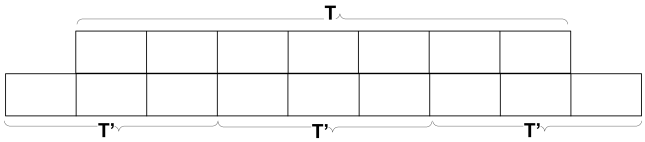

Fig. 7. Example of T' overlapping $(K-1) * T$ intervals

where T overlaps only with $(K-2) * T'$ intervals as shown in Figure 8. We observe that $T = x + (K-2) * T' + y$. From

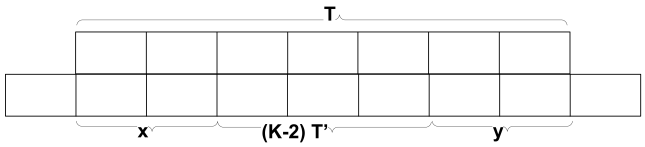

Fig. 8. Schedule where T overlaps with $(K-2) * T'$

the definition we have $K * T' \leq T$. Therefore,

$$K * T' \leq x + (K-2) * T' + y$$

$$2T' \leq x + y.$$

But by definition $x < T'$ and $y < T'$ – a contradiction. ■

Lemma 2. *Let $(C,T) \lhd_s (C',T')$ such that $T \leq T'$ and $C'/K \leq C$ where $K = \lfloor T'/T \rfloor$, then missing an interval of length T will result in missing an interval of length T'.*

Proof: Assume missing an interval of length T corresponds to missing two overlapping intervals of length T' as shown in Figure 9. This implies that the second interval of

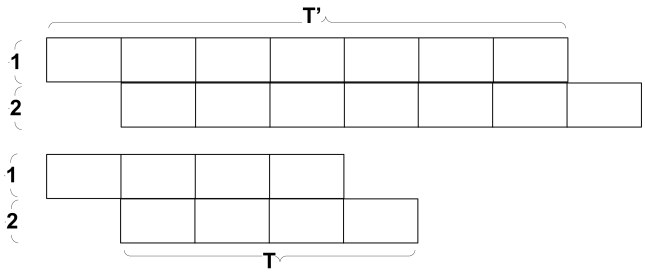

Fig. 9. Misses assuming $T \leq T'$

length T' will get at most $C'-1$ resource units. But the second interval of length T' overlaps $K * T$ non-overlapping intervals, where each interval needs to supply it with at least $\lceil C'/K \rceil$ units. Since the second interval of T gets only $C'-1$ computation times, one of the K non-overlapping intervals

provided it with an allocation time less than $\lceil C'/K \rceil$. Thus one of the K non-overlapping intervals has not been satisfied – a contradiction. Therefore, missing an interval of length T will result in missing an interval of length T'. ■

Lemma 3. *Let $(C,T) \lhd_s (C',T')$ such that $T \geq T'$ and $C \geq T - T' + C'$, then missing an interval of length T will result in missing an interval of length T'.*

Proof: Assume missing an interval of length T corresponds to missing two intervals of length T' as shown in Figure 10. This implies that the second interval of length T'

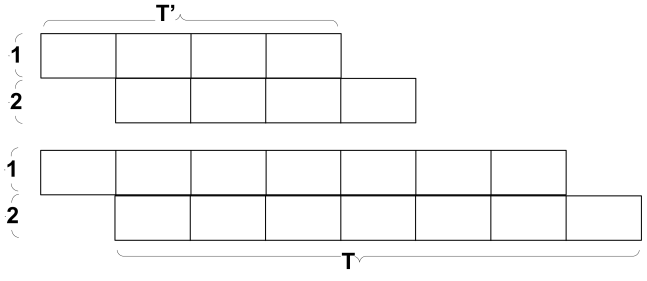

Fig. 10. Misses assuming $T \geq T'$

will get at most $C' - 1$ resource units. But by definition, the second interval of length T needs to contain $T - T' + C'$ allocation units. That interval will receive at most $T - T' + C' - 1$ allocation units – a contradiction. Therefore, missing an interval of length T will result in missing an interval of length T'. ■

Lemma 4. *Given $(C,T,D,W) \lhd_s (C,T,D',W')$ or $(C,T,D,W) \lhd_w (C,T,D',W')$, it is necessary for $D/W \leq D'/W'$.*

Proof: We provide counter-examples under different possible values of D and W.

- $D \leq D'$ and $W \leq W'$. Then unless $D/W \leq D'/W'$, we could have $(C,T,D,D) \lhd_s (C,T,D',W')$ or $(C,T,D,D) \lhd_w (C,T,D',W')$. Contradiction.
- $D \leq D'$ and $W \geq W'$ Then necessarily $D/W \leq D'/W'$.
- $D \geq D'$ and $W \leq W'$ Then unless $D/W \leq D'/W'$, we could have $(C,T,W,W) \lhd_s (C,T,D',W')$ or $(C,T,W,W) \lhd_w (C,T,D',W')$. where $D = W$. Contradiction.
- $D \geq D'$ and $W \geq W'$ Then unless $D/W \leq D'/W'$, we could have $(C,T,W,W) \lhd_s (C,T,D',W')$ or $(C,T,W,W) \lhd_w (C,T,D',W')$. Contradiction.

■

REFERENCES

[1] P. Goyal, X. Guo, and H. M. Vin, "A hierarchical cpu scheduler for multimedia operating systems," in *OSDI*, 1996, pp. 107–121.

[2] I. Shin and I. Lee, "A compositional framework for real-time embedded systems," in *ISAS*, 2005, pp. 137–148.

[3] J. Regehr and J. A. Stankovic, "Hls: A framework for composing soft real-time schedulers," in *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2001, p. 3.

[4] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2003, p. 2.

[5] A. Easwaran, I. Lee, I. Shin, and O. Sokolsky, "Compositional schedulability analysis of hierarchical real-time systems," in *ISORC '07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 274–281.

[6] T. A. Henzinger and S. Matic, "An interface algebra for real-time components," in *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 253–266.

[7] E. Wandeler and L. Thiele, "Interface-based design of real-time systems with hierarchical scheduling," in *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 243–252.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] J. P. Lehoczky, L. Sha, and Y. Ding, "Rate-monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. of the 11th IEEE Real-time Systems Symposium*, Dec. 1989, pp. 166–171.

[10] T.-W. Kuo and A. Mok, "Load adjustment in adaptive real-time systems," Dec 1991, pp. 160–170.

[11] R. West, K. Schwan, and C. Poellabauer, "Dynamic window-constrained scheduling of real-time streams in media servers," *IEEE Transactions on Computers*, vol. 53, pp. 744–759, 2004.

[12] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, pp. 1443–1451, 1995.

[13] G. Bernat, A. Burns, and A. Llamos, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, pp. 308–321, 1999.

[14] Y. Zhang, R. West, and X. Qi, "A virtual deadline scheduler for window-constrained service guarantees," in *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 151–160.

[15] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, p. 286.

[16] T. Chantem, X. S. Hu, and M. Lemmon, "Generalized elastic scheduling," in *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, Dec. 2006, pp. 236–245.

[17] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: a real-time scheduling problem," in *System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, vol. 2, Jan 1989, pp. 693–702 vol.2.

[18] A. Lapets, "Improving the accessibility of lightweight formal verification systems," CS Department, Boston University, Tech. Rep. BUCS-TR-2009-015, April 30 2009.

[19] S. K. Baruah, R. R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[20] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[21] J. Londoño and A. Bestavros, "Netembed: A network resource mapping service for distributed applications," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008.

[22] S. Lauzac, R. Melhem, and D. Moss, "An efficient rms admission control and its application to multiprocessor scheduling," in *IPPS '98: Proceedings of the 12th. International Parallel Processing Symposium*

on International Parallel Processing Symposium. Washington, DC, USA: IEEE Computer Society, 1998, p. 511.

[23] A. Burchard, J. Liebeherr, Y. Oh, and S. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1429–1442, Dec 1995.

[24] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, Dec. 2001, pp. 193–202.

[25] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling," in *IEEE Real-Time Systems Symposium*, 1986, pp. 181–191.

[26] L. Sha and J. B. Goodenough, "Real-time scheduling theory and ada," *Computer*, vol. 23, no. 4, pp. 53–62, 1990.

[27] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, p. 123.

[28] J. Londoño, A. Bestavros, and S.-H. Teng, "Collocation Games And Their Application to Distributed Resource Management," in *HotCloud'09: Workshop on Hot Topics in Cloud Computing*. USENIX, 2009.

[29] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean, "Safe compositional network sketches: The formal framework," in *13th International Conference on Hybrid Systems: Computation and Control (HSCC 2010)*, April 12-16 2010.

[30] ——, "Safe compositional network sketches: Tool and use cases." in *CRTS'09: The IEEE/RTSS Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2009.