Formal Verification of SLA Transformations

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

ANDREI LAPETS
Computer Science Department
Boston University, USA
lapets@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 and Scope: Formal verification of the safety properties of software systems has long been the "holy grail" of a number of research communities focusing on safety-critical applications. While significant progress has been achieved in highly-specialized domains (such as for automotive [1] and avionics [2] applications), making formal verification *accessible* to system builders remains elusive. In this paper, we address two issues related to this state-of-affairs. First, we show that the use of "lightweight" formalisms to support the reasoning processes of system builders (programmers, integrators, and operators) can be quite effective as an alternative to approaches that require significant, deep knowledge of the specific application domain and/or those that require fluency in a particular formalism. Second, we show that the notion of "safety" extends well beyond systems in which loss of human life or expensive equipment is at stake. In particular, we consider a cloud setting wherein it is desirable to *safely*, yet *efficiently* aggregate (co-locate) workloads that are subject to contractual Service Level Agreements (SLAs). In this setting, the system/cloud operator cannot be expected to know (let alone be an expert in) the purpose from a particular SLA request, and thus cannot be expected to arbitrarily modify the set of SLAs for efficient co-location purposes. Rather, the only degree of freedom that such an operator could be assumed to have is the ability to transform an SLA from one form to another, as long as the transformation can be formally verified to be equivalent with respect to a given SLA calculus – a calculus which is independent from and does not require expertise in the application domains of the constituent workloads.

Safe SLA Transformations for Efficient Co-location: 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 part 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. In many instances, however, multiple SLA forms may be "equivalent" in terms of their ability to support a given QoS. For instance, a QoS that spells out an upper bound on (say) data availability in a storage solution could be expressed by any number of SLA forms, depending on how device reliability and redundancy are combined. Similarly, a QoS that spells out an upper bound on (say) missed deadlines in a real-time system could be satisfied by any number of SLA forms, depending on reservation granularity and/or on underlying schedulers. The ability to transform (or rewrite) SLAs from one form to another *safely* (*i.e.*, without jeopardizing the safety or QoS requirements of an underlying workload) gives cloud operators significant freedom in efficiently managing their resources.

A Framework for Safe SLA Transformation: We present our ideas for formal verification of SLA transformations within the context of the generic framework depicted in Figure 1. Our framework enables a user (*e.g.*, a cloud service operator) to propose SLA transformation rules, which upon successful verification for soundness by the AARTIFACT system [3] are fed to an SLA transformation engine. The SLA transformation engine utilizes a set of rewriting rules from the "Transformation Rules Repository" to explore the space of possible safe co-location configurations (mappings). If a feasible mapping is found,[1] then the transformation engine informs the "Resource

This research was supported in part by NSF awards #0720604, #0735974, #0820138, #0952145, and #1012798.

[1]Naturally, if multiple feasible mappings are found, then it is possible to use a scoring function to select the optimal one to realize.

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.16

Manager" to realize the desirable configuration, *e.g.*, using operational services such as those presented in [4]. Otherwise, the engine informs the user, who may opt to allocate more resources or propose additional transformations.

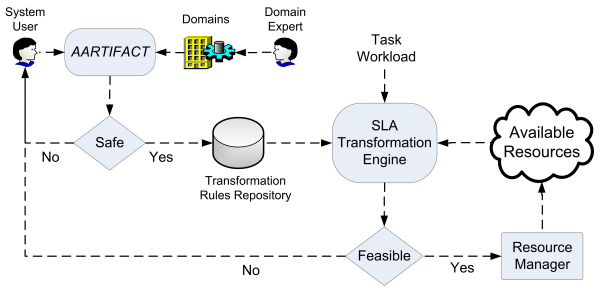

Fig. 1. SLA Transformation Framework Design

While our safe SLA transformation framework depicted in Figure 1 is quite general, ultimately, any instantiation of this framework must be based on a specific "calculus" that enables the manipulation of SLAs. By "calculus", we mean a model that enables reasoning for the purpose of establishing SLA relationships/equivalencies. In this paper, we use a specific calculus that is particularly suited for periodic real-time systems [5]. In particular, our calculus is defined for a setting in which the "Task Workload" consists of a set of periodic real-time tasks, and the set of "Available Resources" are fixed-capacity hosts, which are individually scheduled using a Rate Monotonic Scheduler (RMS) [6]. Here we note that Our RMS-based calculus can be viewed as a simplified variant of the real-time calculus [7]. In general, our framework admits the use of other calculi (*e.g.*, linear algebra or network calculus) if an appropriate library is supplied to the AARTIFACT system.

Paper Outline: The remainder of this paper is organized as follows. In Section II, we overview the various building blocks of our framework. In Section III, we present the specific domain-specific language that we adopt in this paper for the expression of SLAs for real-time workloads. In Section IV, we present examples of SLA transformations with associated machine-readable proofs of correctness that were mechanically verified using the AARTIFACT system. We conclude in Section V with a discussion of related work.

II. Overview of Framework Building Blocks

In this section, we present the basic background necessary to understand the underpinnings of the SLA transformation engine and the resource manager elements of the framework depicted in Figure 1. We follow that with an overview of the salient features and capabilities of the AARTIFACT proof assistant used in our framework.

A. Real-Time Resource Management

The problem of efficiently co-locating tasks on a set of hosts is effectively a "packing" process – given a set of tasks (the workload), a scheduler packs these tasks into groups, such that the SLAs of all tasks in a given group can be satisfied by a single host.[2] For non-real-time tasks, a set of tasks can be packed in a single group as long as the total utilization needs of all tasks in a group is less than the capacity of the host. For real-time tasks, whether or not a set of such tasks can be packed in a single host depends on other details – including the scheduler deployed in the host – and is generally more involved.

Liu and Layland [6] provided classical results for the schedulability condition of n periodic real-time tasks, each requesting a resource for C_i units of time every T_i units of time (the period). Specifically, assuming a preemptive priority scheduler and a *rate-monotic* assignment of priorities to tasks, a group of n tasks is schedulable (*i.e.*, could be packed) on a single host if $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(\sqrt[n]{2}-1)$. Follow-up work [8], [9] showed that if the set of n tasks can be grouped into k subsets such that the periods of tasks in each subset are multiples of one another (*i.e.*, harmonic), then a tighter schedulability condition is $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq k(\sqrt[k]{2}-1)$.

A schedulability condition provides us with the means to check whether a specific set of tasks is schedulable on a single host. In our setting, however, the problem is to partition a large set of tasks (the workload) into smaller sets, each of which would be schedulable on a single host. Moreover, in our setting, safe SLA transformations imply that the specification of the resource requirements (*e.g.*, C_i and T_i) for each task may not be unique. Rather, the SLA for each task could be satisfied by multiple resource requirement specifications. An optimal partitioning of the workload (namely, one that uses the least number of hosts) can be shown to be NP-hard, in general.[3] This necessitates the use of searching and pruning heuristics that explore the solution space – namely, what SLA transform(s) to apply to each task, and how to partition the resulting set of tasks. Details of heuristics that were shown to be fairly effective (achieving significant packing efficiencies when compared to random co-location) and practical (scaling up to dozens of tasks per host) are given in [5], [11].

B. Underlying Verification System and Library

AARTIFACT is formal reasoning assistant,[4] which supports lightweight verification of formal arguments using a database of concepts, propositions, and syntactic idioms that deal with common mathematical concepts such as numbers, vectors, and sets. It has a familiar concrete syntax overlapping with English, MediaWiki markup, and Latex, and a friendly user interface [3]. The AARTIFACT system has been used in the past to formally verify the consistency [12] of a formal framework of safe transformations of constraints governing constrained-flow networks [13].

AARTIFACT's expert-managed database of propositions and syntactic idioms is compiled into two separate applications.

[2]Without loss of generality, in this paper, we assume that all hosts are identical.

[3]Our problem can be easily reduced to a multiprocessor real-time scheduling problem, which is known to be NP-hard [10].

[4]An interactive demonstration is available at `http://www.aartifact.org`.

The first is a server-side verifier that can be executed when requests are submitted to it. It parses arguments authored using a familiar concrete syntax that consists of common LaTeX macros and English phrases [14]. It validates these arguments using an inference algorithm the context of which has been augmented with a data structure that uses the database of propositions to compute congruence closures [15] throughout the verification process. The second application is a client-side JavaScript that automatically informs users of syntactic idioms available in the ontology as they author formal arguments.

The current AARTIFACT database of supported propositions and definitions contains a collection of hundreds of entries. Each proposition deals with semantic concepts (numbers, sets, vectors), properties they may have (the sums of their elements, the subsets and subvectors they may have and their properties, etc.), and relationships that may hold between them. The following proposition represents a very simple example:

> " for any x, y, z,
> $x \in \mathbb{R},\ y \in \mathbb{R},\ z \in \mathbb{R},\ x < y,\ y < z$
> implies that
> $x < z$".

Many of these propositions simply state an equivalence between two forms of notation or syntax. They can be viewed as establishing a normal form for representing certain concepts or properties thereof. For example, the following proposition converts the typical notation for a sum of a finite range of components in a vector, "$v_i + \ldots + v_j$", into a predicate that is then used in other propositions about the properties of this concept:

> " for any v, i, j,
> v is a vector , $v \in \mathbb{Z}^{|v|},\ 0 \leq i,\ i \leq j,\ j < |v|$,
> implies that
> $v_i + \ldots + v_j$ is the sum of components in
> v in index range i to j".

The purpose of the database in its current incarnation is to support common algebraic manipulations and concepts in arithmetic and naive set theory.

The AARTIFACT system's inference algorithm relies on a collection of inference rules corresponding to those found in a typical definition of higher-order logic [16] (i.e. those governing conjunction, disjunction, negation, and quantification), and variants thereof found in common sequent calculus formulations [17]. The validation procedure verifies arguments with respect to these inference rules as well as the entire database of propositions. The verification capabilities provided by the AARTIFACT system are "lightweight" in that it is not essential that any guarantee be provided about the logical consistency or completeness of the validation process (allowing a user to validate that, for example, the proper syntax is used, that all variables are bound, or that only "intuitive" symbolic manipulations are employed). However, the soundness of the validation process can be guaranteed if the system can use only a subset of the inference rules and database propositions that is consistent with a particular logic. This capability has been demonstrated for propositional and first-order logic, and the proofs verified in this paper are verified using a consistent subset of the propositions that includes many facts about the properties of vectors and sets.

III. A Domain-Specific Language for Real-Time SLAs

As we hinted earlier, SLA transformations must be based on a given "calculus". Thus, for the purposes of SLAs governing real-time workloads (and without loss of generality), in this section we present a specific "language" for expressing SLAs and transformations thereof.

A. Periodic Supply/Demand SLA Types

SLAs can be seen as encapsulators of the resources supplied by hosts (producers) and demanded by tasks (consumers). We provide a specific model for SLAs that supports periodic, real-time resource supply and demand.[5] We also provide basic type-theory-inspired definitions that allow us to represent SLA transformations as a collection of inference rules.

Definition. A soft SLA type τ is a tuple of natural numbers $(C, T, D, W) \in \mathbb{N}^4$, $C \leq T$, $D \leq W$, and $W \geq 1$, where C denotes the resource capacity supplied or demanded in each time interval that is T units in length, and D is the maximum number of times such an allocation cannot be honored in a window consisting of W allocation intervals.

Definition. A hard SLA type τ is a tuple of natural numbers $(C, T) \in \mathbb{N} \times \mathbb{N}$, $C \leq T$, where C denotes the resource capacity supplied or demanded in each interval of time that is T units long; such a type requires that the allocation of resources is honored at all times.

Note that these definitions imply that a type (C, T) is equivalent to $(C, T, 0, 1)$. These definitions assume that the periodic capacity could be allocated as early as the beginning of any interval (or period) and must be completely produced or consumed by the end of that same interval (that is, the allocation deadline is T units of time from the beginning of the period).

As presented, our SLA types are general enough to express a wide range of supply/demand elements. For example, 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 $(1, 30)$ 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

[5] Our SLA model mirrors existing periodic task models in the real-time scheduling literature [18]–[20].

allocations. An SLA of type $(k, k \cdot n, 0, 1)$, $n > 1, k \geq 1$ can characterize the fractional supply provided in a round robin fashion to n processes using a quantum k. In all of these examples, the SLA type does not admit missed allocations (since $D = 0$). An SLA of type $(1, 30, 2, 5)$ is an example of a task that tolerates 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.

B. Strong Satisfaction of Hard SLAs

We consider what it means to satisfy an SLA type. The following definitions formalize the notion of satisfaction by considering a schedule that governs a single task. We first do so for SLAs of the form (C, T) (*i.e.*, those that do not admit missed allocations).

Definition. A schedule a is a vector $a \in \{0, 1\}^\infty$.

A schedule is an infinite sequence (or vector) of 0s and 1s in which each entry represents whether or not a resource is allocated to the single task in question at that discrete point in time. We use the subscript notation a_i to refer to an entry in the sequence corresponding to a particular point in time. A schedule a is said to *strongly satisfy* (denoted by $\models$) a hard SLA type (C, T) if the resource is allocated for C units of time in non-overlapping (fixed) intervals of length T.[6]

Definition. We say $a \models (C, T)$ iff for every $m \geq 0$,

$$a_m + \cdots + a_{m+(T-1)} \geq C.$$

C. Strong Satisfaction of Soft SLAs

We generalize the above definition for the more general soft SLA types of the form (C, T, D, W). To do so, we introduce an aggregate vector $A(a, C, T)$ that characterizes a schedule a with respect to some C and T. We then extend our notion of strong SLA satisfaction. While conceptually similar to the simpler hard SLA definition, the consideration of missed allocations requires a slightly more elaborate notation.

Definition. $A(a, C, T) \in \{0, 1\}^\infty$ is a vector defined as:

$$A(a, C, T)_m = \begin{cases} 0 & \text{if } a_m + \cdots + a_{m+(T-1)} < C \\ 1 & \text{if } a_m + \cdots + a_{m+(T-1)} \geq C \end{cases}$$

A schedule a is said to *strongly satisfy* (denoted by $\models$) 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) intervals of length $W \cdot T$.

Definition. $a \models (C, T, D, W)$ iff for every $Q \geq 0$ and $m = Q \cdot W$,

$$A(a, C, T)_m + A(a, C, T)_{m+1} + \cdots + A(a, C, T)_{m+(W-1)} \quad \geq W - D$$

[6]Strong satisfiability of an SLA type is in contrast to weak satisfiability, wherein the intervals are overlapping. In this paper we restrict our notion of safety to strong satisfiability.

IV. Machine-assisted Verification of SLA Transformations

In this section, we present a set of SLA transformations that exemplify (and certainly do not exhaust) the range of conjectures that a user of our framework (*cf.* Figure 1) would need to formally check for safety before allowing the SLA transformation engine to use. Recall that the verification of these transformations must be based on specific assumptions regarding the underlying scheduler. As we mentioned before, in this paper we assume that the underlying scheduler is a Rate Monotonic Scheduler.

The transformations presented in this section are "coded" as rewrite rules using our type-theoretic SLA specification language. We do so first for SLAs that do not allow for missed allocations (Hard SLAs) and then provide additional transformations that apply only to SLAs that tolerate missed allocations (Soft SLAs).

A. Rewriting Rules for Hard SLAs

Each SLA transformation is a type inference rule associated with a set of pre-conditions. Intuitively, each inference rule implies that we can *safely* substitute (rewrite) some SLA type (C, T) for some other type (C', T') as long as the pre-conditions of the inference rule are met. We present the inference rules for rewriting hard SLAs in Figure 2.

In order to establish the soundness of these inference rules for a particular satisfaction relation (*i.e.*, $\models$), we must provide a proof for each rule. In this context, a rule is *sound* if it is consistent with the underlying AARTIFACT library of facts governing vectors, sets, and numbers. We present mechanically verifiable proofs of the soundness of each of the rewriting rules in Figure 2 with respect to strong satisfiability. All mechanically verifiable proofs (using AARTIFACT) are enclosed in boxes. For example, we reproduce below machine-readable definitions for hard SLA types and strong satisfiability.

```
Assume for any C, T ∈ ℕ, (C, T) is a hard SLA type iff
C ≤ T.

Assume for any a, C, T,
    a satisfies (C, T) iff
        (C, T) is hard SLA type,
        a ∈ {0, 1}^∞,
        and for all m ∈ ℕ, a_m + ... + a_{m+(T-1)} ≥ C.
```

We can now present proofs for each rewriting rule.

Theorem 1. *The* [TIME-↑] *rule is sound with respect to* $\models$.

Proof: We proceed by taking advantage of the fact that the sum of any sequence in a schedule a is at least as large as the sum of any of its subsequences.

$$\textsc{Time-}\uparrow \frac{a \models (C,T) \quad T' \geq T}{a \models (C,T')} \qquad \textsc{Capacity-}\downarrow \frac{a \models (C,T) \quad C' \leq C}{a \models (C',T)}$$

$$\text{CT-}\downarrow \frac{a \models (C,T) \quad K > 0 \quad T' = T - K \quad C' \leq C - K}{a \models (C',T')}$$

$$\text{CT-}\uparrow \frac{a \models (C,T) \quad K > 0 \quad T' = T \cdot K \quad C' \leq C \cdot K}{a \models (C',T')}$$

Fig. 2. Rewriting rules for hard SLAs.

Assert for any a, C, T,
  if (C,T) is a hard SLA type and a satisfies (C,T) then
    for all $T' \in \mathbb{N}$, if $T' \geq T$ then
      (C,T') is a hard SLA type, and
      for all $m \in \mathbb{N}$,
        $a_m + ... + a_{m+(T-1)} \geq C$,
        $a_m + ... + a_{m+(T'-1)} \geq a_m + ... + a_{m+(T-1)}$
        $a_m + ... + a_{m+(T'-1)} \geq C$,
      a satisfies (C,T').

■

Theorem 2. *The* [CAPACITY-↓] *rule is sound with respect to* $\models$.

Proof: The proof for this rule simply takes advantage of the transitivity of $\geq$.

Assert for any a, C, T,
  if (C,T) is a hard SLA type and a satisfies (C,T) then
    for all $C' \in \mathbb{N}$, if $C' \leq C$ then
      (C',T) is a hard SLA type, and
      for all $m \in \mathbb{N}$,
        $a_m + ... + a_{m+(T-1)} \geq C$,
        $a_m + ... + a_{m+(T-1)} \geq C'$,
      a satisfies (C',T).

■

Theorem 3. *The* [CT-↓] *rule is sound with respect to* $\models$.

Proof:

Assert for any a, C, T,
  if (C,T) is a hard SLA type and a satisfies (C,T) then
    for all $K, C', T' \in \mathbb{N}$,
    if $K > 0$, $T' = T - K$, and $C' \leq C - K$ then
      (C',T') is a hard SLA type, and
      for all $m \in \mathbb{N}$,
        $a_m + ... + a_{m+(T-1)} \geq C$,
        $(a_m + ... + a_{m+(T-1)}) - K \geq C - K$,
        $a_m + ... + a_{(m+(T-1))-K} \geq (a_m + ... + a_{m+(T-1)}) - K$,
        $a_m + ... + a_{(m+(T-1))-K} \geq C'$,

$$\begin{aligned}(m + (T-1)) - K &= m + ((T-1) - K) \\ &= m + (T - 1 - K) \\ &= m + ((T-K) - 1) \\ &= m + (T' - 1),\end{aligned}$$

        $a_m + ... + a_{m+(T'-1)} \geq C'$,
      a satisfies (C',T').

■

Theorem 4. *The* [CT-↑] *rule is sound with respect to* $\models$.

Proof:

Assert for any a, C, T,
  if (C,T) is a hard SLA type and a satisfies (C,T) then
    for all $K, C', T' \in \mathbb{N}$,
    if $K > 0$, $T' = T \cdot K$, and $C' \leq C \cdot K$ then
      (C',T') is a hard SLA type, and
      for all $m \in \mathbb{N}$,
        $a_m + ... + a_{m+((T \cdot K)-1)} \geq C \cdot K$,
        $a_m + ... + a_{m+(T'-1)} \geq C'$,
      a satisfies (C',T').

■

B. Type Rewriting Rules for Soft SLAs

The above set of SLA rewrite rules did not allow transformations that involve the manipulation of SLAs with non-zero D and W parameters. In Figure 3, we provide additional rewriting rules that are applicable to soft SLA types.

The first two rules in Figure 3 are identical to the corresponding rules in Figure 2, except that the D and W parameters have been included (recall that in a hard SLA type (C, T), $D = 0$ and $W = 1$). The new, third rule [S-DROPS-↑] allows for adjustment of the D parameter. Since we are now dealing with soft SLAs, a few new definitions are needed.

Assume for any $C, T, D, W \in \mathbb{N}$,
(C, T, D, W) is a soft SLA type iff $C \leq T$, $D \leq W$, and $W \geq 1$.

Introduce A.
Assume for any $a \in \{0,1\}^\infty, C, T \in \mathbb{N}$, $A(a, C, T) \in \{0,1\}^\infty$.

Assume for any a, C, T, D, W, a satisfies (C, T, D, W) iff
(C, T, D, W) is a soft SLA type, $a \in \{0,1\}^\infty$, and
for all $m, Q \in \mathbb{N}$,
if $m = Q \cdot W$ then
$A(a, C, T)_m + ... + A(a, C, T)_{m+(W-1)} \geq W - D$.

We present an abstract proof of the rule [S-DROPS-↑]. The proofs for the first four rules are similar to this proof and are omitted.

Theorem 5. *The* [S-DROPS-↑] *rule is sound with respect to* $\models$.

Proof:
We take an abstract approach. Because $A(a, C, T)$ is already a vector, it is not necessary to expand (or even introduce) the explicit definition of A.

Assert for any $a \in \{0,1\}^*, C, T, D, W \in \mathbb{N}$,
if (C, T, D, W) is a soft SLA type and a satisfies (C, T, D, W) then
for all $D' \in \mathbb{N}$, if $D' \geq D$ then
(C, T, D', W) is a soft SLA type, and
for all $m, Q \in \mathbb{N}$,
if $m = Q \cdot W$ then
$A(a, C, T)_m + ... + A(a, C, T)_{m+(W-1)} \geq W - D \geq W - D'$,
a satisfies (C, T, D', W).

■

V. Related Work and Conclusion

The work described in this paper brings to bear the premise of formal verification using the AARTIFACT lightweight proof assistant on the problem of efficient co-location of real-time workloads. We believe that the capabilities enabled through this framework are crucial for emerging shared infrastructure cloud and grid environments, and as far as we can tell there is no prior work that proposed or developed such capabilities. That said, there are two distinct bodies of related work to consider – namely those concerned with applying real-time scheduling theory to real-time workload management, and those concerned with the development and applications of formal verification systems.

Efficient Co-location of Real-Time Workloads: In a cloud or grid setting, the hosting environment operates at a macroscopic scale (*e.g.*, enforcing specific resource utilization ratios over relatively long time scales). While appropriate for many 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. 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.

Numerous previous studies dealt with hierarchical scheduling frameworks [21]–[24]. Regehr and Stankovic [21] 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 [22] 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 has also been extended further [23] to include context switching overhead and incremental analysis.

A common assumption in hierarchical scheduling frameworks is that the grouping of applications and/or schedulers under a common ancestor in the scheduling hierarchy is known *a priori* based on domain specific knowledge; for example, all applications with the same priority are grouped into a single cluster, or all applications requiring a particular flavor

$$\text{S-CT-}\downarrow \quad \frac{a \models (C, T, D, W) \qquad K > 0 \qquad T' = T - K \qquad C' \leq C - K}{a \models (C', T', D, W)}$$

$$\text{S-CT-}\uparrow \quad \frac{a \models (C, T, D, W) \qquad K > 0 \qquad T' = T \cdot K \qquad C' \leq C \cdot K}{a \models (C', T', D, W)}$$

$$\text{S-DROPS-}\uparrow \quad \frac{a \models (C, T, D, W) \qquad D' \geq D}{a \models (C, T, D', W)}$$

Fig. 3. Rewriting rules for soft SLAs.

of scheduling are grouped into a single cluster managed by the desired scheduling scheme. Most of this prior body of work is concerned with addressing the schedulability problem given such a *fixed* hierarchical structure (*i.e.*, 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 addressed in [5] with the use of SLA transformations that were proven to be safe using domain-specific knowledge. In this paper, we showed that domain expertise is not necessary by showing how the use of AARTIFACT could assist the process of deriving SLA transformations.

The idea of transforming task periods for improving schedulability is not new. Related work [25], [26] introduced 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 possible to derive for overlapping as well as traditional, non-overlapping intervals.

Machine Verification: The work presented in this paper illustrates the usefulness of several characteristics of a formal reasoning system that prioritizes usability in its design. The concrete syntax of the system is familiar to the community in question, and it is not necessary to cite explicitly within a formal argument the definitions and propositions that are being employed. These features are recognized as important within several efforts and projects to design systems with similar characteristics [27]–[29]. Furthermore, the system takes advantage of an extensive library of definitions and propositions dealing with common mathematical concepts and provides native support for some of these concepts. This is inspired by work in a subdiscipline of artificial intelligence that deals with the assembly and application of ontologies, such as the Cyc Project [30]. The augmented context integrated into the inference algorithm and its closure operation are similar to structures and algorithms found in work on congruence closures [15].

There exist few examples of applications of lightweight formal reasoning systems within novel research. Some examples include applications within cryptography [31], security in computation [32], and economic mechanism design [33]. In this work, we have utilized a formal reasoning system to define and reason about a novel framework for rewriting SLAs to enable efficient co-location in a cloud setting. We showed that this framework is consistent with respect to its semantics by providing machine-verified proofs of its consistency. These proofs are accessible to humans and are of a manageable size, demonstrating the value of employing a user-friendly verification system.

REFERENCES

[1] J. Botaschanjan, L. Kof, C. Kühnel, and M. Spichkova, "Towards verified automotive software," in *SEAS '05: Proceedings of the second international workshop on Software engineering for automotive systems*. New York, NY, USA: ACM, 2005, pp. 1–6.

[2] O. Laurent, "Using formal methods and testability concepts in the avionics systems validation and verification (v&v) process," *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 1–10, 2010.

[3] A. Lapets, "User-friendly Support for Common Concepts in a Lightweight Verifier," in *Proceedings of VERIFY-2010: The 6th International Verification Workshop*, Edinburgh, UK, July 2010.

[4] V. Ishakian, R. Sweha, J. Londoo, and A. Bestavros, "Colocation as a Service: Strategic and Operational Services for Cloud Colocation," in *NCA '10: The 9th IEEE International Symposium on Network Computing and Applications*. Boston, MA: IEEE Computer Society, July 2010.

[5] V. Ishakian, A. Bestavros, and A. Kfoury, "A Type-Theoretic Framework for Efficient and Safe Colocation of Periodic Real-time Systems," in *RTCSA '10: The 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Macau, China: IEEE Computer Society, August 2010.

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

[7] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *ISCAS 2000: Proceedings of the 2000 IEEE International Symposium on Circuits and Systems*, vol. 4, 2000, pp. 101–104.

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

[9] T.-W. Kuo and A. Mok, "Load adjustment in adaptive real-time systems," in *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, Dec 1991, pp. 160–170.

[10] M. Garey and D. Johnson, *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, CA, 1979.

[11] V. Ishakian and A. Bestavros, "MORPHOSYS: Efficient Colocation of QoS-Constrained Workloads in the Cloud," CS Department, Boston University, Tech. Rep. BUCS-TR-2011-002, January 24 2011.

[12] A. Lapets and A. Kfoury, "Verification with Natural Contexts: Soundness of Safe Compositional Network Sketches," CS Dept., Boston University, Tech. Rep. BUCS-TR-2009-030, October 1 2009.

[13] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean, "Safe Compositional Network Sketches: The Formal Framework," in *HSCC '10: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (in conjunction with CPSWEEK)*, Stockholm, Sweden, April 2010, pp. 231–241.

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

[15] L. Bachmair and A. Tiwari, "Abstract congruence closure and specializations," in *CADE-17: Proceedings of the 17th International Conference on Automated Deduction*. London, UK: Springer-Verlag, 2000, pp. 64–78.

[16] S. C. Kleene, *Mathematical Logic*. Dover Publications, 1967.

[17] G. Gentzen, *The Collected Papers of Gerhard Gentzen*. Amsterdam: North-Holland Publishing Co., 1969, edited by M. E. Szabo.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[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. Abel, B. Chang, and F. Pfenning, "Human-readable machine-verifiable proofs for teaching constructive logic," in *PTP '01: IJCAR Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs*, U. Egly, A. Fiedler, H. Horacek, and S. Schmitt, Eds., Siena, Italy, 2001.

[28] C. E. Brown, "Verifying and Invalidating Textbook Proofs using Scunak," in *MKM '06: Mathematical Knowledge Management*, Wokingham, England, 2006, pp. 110–123.

[29] D. McMath, M. Rozenfeld, and R. Sommer, "A Computer Environment for Writing Ordinary Mathematical Proofs," in *LPAR '01: Proceedings of the Artificial Intelligence on Logic for Programming*. London, UK: Springer-Verlag, 2001, pp. 507–516.

[30] K. Panton, C. Matuszek, D. Lenat, D. Schneider, M. Witbrock, N. Siegel, and B. Shepard, "Common Sense Reasoning – From Cyc to Intelligent Assistant." in *Ambient Intelligence in Everyday Life*, ser. LNAI, Y. Cai and J. Abascal, Eds., vol. 3864. Springer, 2006, pp. 1–31.

[31] M. Backes, C. Jacobi, and B. Pfitzmann, "Deriving cryptographically sound implementations using composition and formally verified bisimulation," in *Formal Methods Europe*, ser. Lecture Notes in Computer Science, vol. 2931. Springer-Verlag, 2002, pp. 310–329.

[32] M. Backes, C. Hritcu, and M. Maffei, "Automated verification of remote electronic voting protocols in the applied pi-calculus," in *Proceedings of 21st IEEE Computer Security Foundations Symposium (CSF)*, June 2008.

[33] E. M. Tadjouddine and F. Guerin, "Verifying dominant strategy equilibria in auctions." in *CEEMAS*, ser. Lecture Notes in Computer Science, H.-D. Burkhard, G. Lindemann, R. Verbrugge, and L. Z. Varga, Eds., vol. 4696. Springer, 2007, pp. 288–297.