A Satisfiability Approach to Speed Assignment for Distributed Real-Time Systems

Pratyush Kumar, Devesh B. Chokshi and Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland

***Abstract*—We study the problem of assigning speeds to resources serving distributed applications with delay, buffer and energy constraints. We argue that the considered problem does not have any straightforward solution due to the intricately related constraints. As a solution, we propose using Real-Time Calculus (RTC) to analyse the constraints and a SATisfiability solver to efficiently explore the design space. To this end, we develop an SMT solver by using the OpenSMT framework and the Modular Performance Analysis (MPA) toolbox. Two key enablers for this implementation are the analysis of incomplete models and generation of conflict clauses in RTC. The results on problem instances with very large decision spaces indicate that the proposed SMT solver performs very well in practice.**

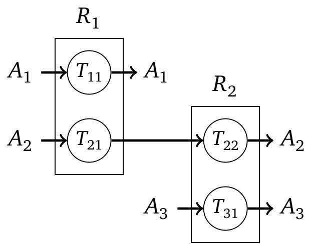
I. Introduction

The overarching theme in computing systems has been the move towards distributed setups, with the increasing convergence of sensing, communication and computation. One of the consequences of such a trend is the increased complexity of the systems, and the resultant high costs of design optimisation. Such design optimisation problems need novel solutions that are *generic* in their scope and *efficient* in computation costs.

In this work, we look at such a design optimisation problem, namely assigning processing speeds on a distributed real-time system with multiple conflicting constraints. The system we consider has multiple resources, whose speeds can be independently assigned, and which serve multiple applications. The speed assignment is to be done under multiple constraints: (a) applications may have end-to-end delay constraints, (b) resources may have finite buffer space for pending jobs, and (c) resources may have a finite energy budget in a given interval of time. An example of such a system is the cyber-physical system within a modern automobile [1]. The key constraints in such a system are the delay constraints which have to be certifiably satisfied. For instance, the time between the application of the brake pedal and the actuation of the relevant braking mechanism has to be safely bounded. Another relevant example is the Network-on-Chip in a multi-core processor, where both performance criteria and buffer constraints can be important [2]. In even larger systems such as data centers the energy consideration assumes significance, while meeting desired performance constraints is also necessary [3].

Intuitively, the assignment of the speeds to resources provides design choices to trade-off between the considered objectives of delay, buffer and energy. For instance, by decreasing the speed of a resource, we can decrease the energy consumption of that resource at the potential cost of added delays in serving tasks or need for larger input buffers. However, the distributed nature of the considered problem implies that these trade-offs are not always intuitive. Let us illustrate this with an example.

Consider the architecture of two resources and three applications with timing properties as shown in Fig. 1. Application A_1 and A_3 have a single task, while application A_2 has two tasks with a data dependency. The mapping of tasks on to resources, and their scheduling is shown. For this specific example, halving the speed of resource R_1 leads to counter-intuitive consequences. Firstly, buffer requirement on the input of task T_{22} increases from 1 to 3, though we may expect that a slower resource for the input task would lead to a smaller buffer level. Secondly, the worst-case energy consumption of resource R_2 within any interval of some given length increases. On the contrary, we may expect that the energy consumption of resource R_2 is unaffected by changing the speed of resource R_1. Thirdly, the delay of application A_3 increases from 2 to 4, though its only task is mapped on to resource R_2 whose speed is unchanged. None of these consequences conform to the intuitive trade-off between the considered objectives. One may expect that these issues would only be further aggravated for systems with more resources and/or task dependencies.



	A_1	A_2	A_3
Period	10	6	6
Jitter	4	0	0
Exec.Time	3	T_{21}: 0.5 T_{22}: 1	1

Priority: $T_{11} > T_{21}$, $T_{22} > T_{31}$

Fig. 1. Example to illustrate the intricacies in the trade-off between different constraints. All times are in ms.

The above discussed intricacies in the relationship between the constraints motivate two responses. Firstly, we need a generic *theory* to systematically compute safe bounds on the end-to-end delay, buffer requirement and energy consumption. Secondly, the problem of choosing the speeds of the processing units while meeting a given set of constraints is not straightforward. For instance, the constraints do not exhibit any well-founded properties of linearity or convexity. Hence a generic design space *exploration engine* has to be designed.

The design space grows exponentially in the number of resources and polynomially in the number of different speeds. For problems with tens of resources and few different speeds, this design space is very large and beyond approach with exhaustive search. One solution to cover a large design space is to use heuristics such as gradient descent, simulated annealing or evolutionary algorithms. However, these methods do not provide any guarantees of optimality, i.e., if they do not find a feasible solution we are not guaranteed that there is none. This brings us to the core issue addressed in this paper: We aim to reconcile the complexity of speed assignment problem with the need for an optimal solution.

To the above end, we formulate the problem as a SATisfiability problem [4] interpreted on the background theory of Real-Time

978-3-9815370-0-0/DATE13/©2013 EDAA

Calculus [5]. In the process we design a custom Satisfiability Modulo Theory (SMT) solver [6]. We first show that formally, this setup guarantees the computation of optimal solutions to the speed assignment problem. Two enabling principles for this are (a) analysis of incompletely specified systems in a RTC formulation, and (b) computation of conflict clauses which formalise the non-satisfaction of different constraints in terms of possible speed assignments. Our implementation uses the OpenSMT [7] solver framework which is specifically designed to interface custom theory solvers. The theory solver for Real-Time Calculus is designed using Modular Performance Toolbox [8].

The key advantage of a SMT solver is the deep embedding of the theory solver within the exploration engine. This deep embedding has been shown to perform exceeding well in practice in diverse fields [6]. The main result of this work is to confirm this success of SMT solvers in the considered case of speed assignment with RTC as the background theory. In particular, we show that for problem spaces with very large design spaces, the solver computes a valid solution, if there is one, within hundreds of solver calls to the MPA toolbox. On a typical desktop computer, a solver call is executed within a couple of seconds. This motivates formulation of other design space exploration problems within using our SMT setup.

The rest of the paper is organised as follows. In Section II we formalise the models considered. We motivate our work with an example that illustrates the working of the proposed solver in Section III. We then introduce SAT solvers and the Real-Time Calculus in Section IV. The proposed SMT solver is presented in Section V. We experimentally evaluate the solver in Section VI. We compare our work with existing literature in Section VII and conclude in Section VIII.

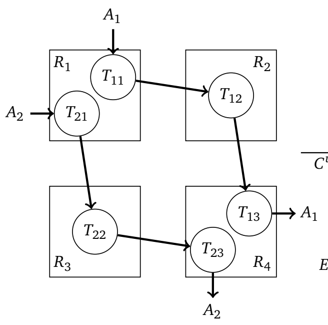
II. Problem Formulation

We will now describe the models and formulate the problem.

Resources: We consider a network of resources denoted by the set $\mathbf{R}$. The ith resource is denoted R_i. The speeds of (some of) the resources can be independently assigned. We denote the speed of R_i by s_i, and it belongs to a given finite set of allowed speeds denoted $\mathbf{S}_i$. In this work, we consider speeds as ratios to a normalised speed. The power consumed by a resource is assumed to be a convex increasing function of its speed denoted as $\phi_i(s_i)$. When not executing any task, the resource consumes a fixed idle power $\phi_i(0)$.

Applications: An application is a chain of tasks (a linear task graph) which have to be sequentially performed for an incoming stream of jobs. We denote the set of all applications as $\mathbf{A}$ and the ith application as A_i. The jth task of the A_i is denoted as $T_{i,j}$. Task $T_{i,j}$ is mapped on to resource $m_{i,j} \in \mathbf{R}$. The worst-case execution time (WCET) and the best-case execution time (BCET) of $T_{i,j}$ on resource $m_{i,j}$ for the normalised speed are $C^u_{i,j}$ and $C^l_{i,j}$, respectively. We assume execution times linearly scale with the speed: If $m_{i,j}$ runs at speed s, then the WCET and BCET become $C^u_{i,j}/s$ and $C^l_{i,j}/s$, respectively. A_i is characterised by an input load given by an arrival curve denoted as α_i. We discuss the interpretation and utility of arrival curves in Section IV-B.

Scheduling: If two tasks are mapped on to the same processing unit then the scheduling is based on pre-emptive fixed priority according to some given priority ordering.



	A_1	A_2
Period	10	10
Jitter	20	20
D	9	13

	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}
$C^u = C^l$	1	2	4	1	3	4
B	3	2	1	3	1	3

Priority: $A_1 > A_2$
$\mathbf{S_i} = \{1, 1.5, 2\}$, $\phi(s) = s^{2.5}$
$E = (210, 210, 580, 760)$, $\tau_i = 1000$

Fig. 2. Problem instance in considered motivating example

Constraints: We can have a combination of some of the following constraints

- An application A_i may have a upper-bound on the end-to-end delay denoted as D_i.
- Each task has an input buffer that queues pending jobs. Some of these buffers may have size constraints denoted as B_{ij} for task T_{ij}.
- A resource R_i may have a upper-bound on the energy consumed in any interval of some given length τ_i, denoted as E_i.

Speed assignment: The aim is to compute the speed of each resource R_i, i.e., evaluate $s_i \in \mathbf{S}_i$, such that the delay, buffer and energy constraints are satisfied. If there is no such assignment, then this must be shown to be so.

III. Motivating Example

In this section, we will demonstrate the working of the proposed solver with a problem of size that is demonstrable. Consider two applications with three tasks each, executing on four resources as shown in Fig. 2. The figure shows properties of the tasks, the priority ordering, the allowed speeds, power consumption as function of speed and the constraints on the energy consumed in any interval of length 1000, for each of the resources. Each of the resources can be assigned one of three speeds 1, 1.5 and 2, coded as L, M or H, for low, medium and high, respectively.

The design space of speed assignments is of size $3^4 = 81$. By exhaustive search, we found that none of the speed assignments satisfies all the constraints, i.e., the problem is unsatisfiable. We will now demonstrate how our proposed solver comes to the same conclusion, by elaborating the trace obtained from our solver. This trace is visualised in a series of *solver calls*. In each such call, the solver attempts to find a solution by restricting the search to some part of the design space. Formally, this part of the design space is encoded as the *model*. If it can be shown that the constraints will be violated for any speed assignment represented by this model, then we label the solver call as `UNSAT`. In such a case, a part of the design space is deemed to be infeasible. Formally, this part of the design space is encoded as the *conflict clause* and it is eliminated from subsequent searches. Otherwise, we label the solver call as `SAT`, and a new model is considered. This process stops when either the whole design space is shown to be infeasible, or if a solver call with a model representing a single design returns `SAT`.

For the considered problem, the trace of the solver calls is shown in Fig. 3. We represent the models and the conflict clauses by 4-tuples which give the ranges of the speed for the four resources, respectively. We also visualise them in squares with 81 smaller squares that correspond to the 81 speed assignments. If a speed assignment is

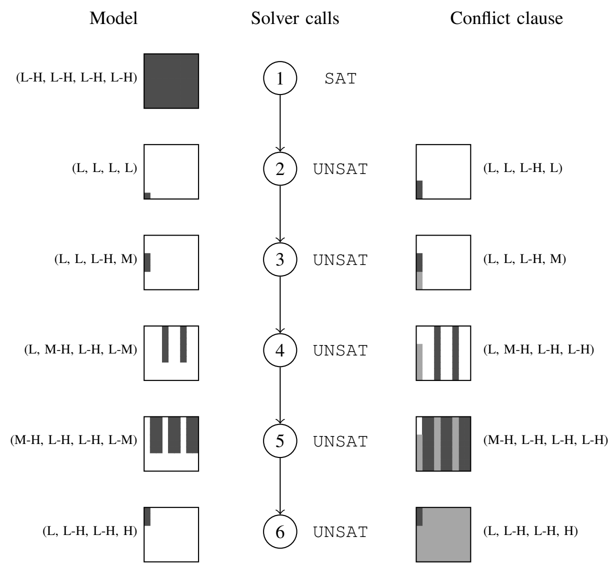


Fig. 3. Trace of solver calls for problem of Fig. 2

included in the model or the conflict clause then the corresponding square is shaded dark. Additionally, in the visualisation of the conflict clause, we shade gray those squares which have already been proved to be infeasible in earlier solver calls. At the end of the search, in the visualisation of the conflict clause, every square is shaded implying that there is no solution.

Let us discuss some salient features. Firstly, note that in just 6 solver calls we prove that the entire design space is infeasible. Secondly, note that the models include several design points. For instance, the model in the first solver call includes the entire design space. Thirdly, in the case of a `UNSAT` solver call, the conflict clause can include several design points, even more than those in the model. For instance, in the fifth solver call, the conflict clause includes 54 design points while the model includes 36 design points.

The three defining advantages of the solver that enable the quick verification are (a) efficient generation of models, (b) checking constraints for models that include multiple design points, and (c) generating conflict clauses that eliminate multiple design points. While the first advantage is due to the SAT engine, the other two advantages are due to specialised operations possible in the theory solver. In Section V, we will formally discuss how these operations can be performed for the considered theory of Real-Time Calculus. Then in Section VI, we will demonstrate that these advantages perform very well in practice and can efficiently search very large design spaces.

IV. Background

In this section, we discuss preliminary ideas from Satisfiability Modulo Theory (SMT) and Real-Time Calculus (RTC).

A. Satisfiability Modulo Theory (SMT)

Satisfiability Modulo Theory (SMT) solvers are an evolution on Satisfiability (SAT) solvers, which check the satisfiability of a logical formula over propositional variables. Though SAT problems are characteristically NP-complete, availability of efficient search techniques, such as DPLL [4], make them good practical choices for decision problems.

The need for SMT solvers arises because fully encoding a decision problem into boolean variables is not always practical or even efficient [9]. In many problems, specialised methods need to be used to provide decision procedures, while generic and efficient SAT routines search over the design space. An SMT problem is an instance of a decision problem on logical formula, where the formula is interpreted in one or more background first-order theories [9].

An SMT solver can be viewed as comprising of a SAT solver and a theory solver. Crucial to our subsequent discussion is the interaction between the SAT and theory solvers. The solvers interact via *propositional variables*. These variables encode the decision space to be explored. The SAT solver assigns truth values to (some of) these propositional variables. Such an assignment represents a part of the design space and is denoted as the *model* M. The theory solver *interprets* the model M and computes the corresponding theory-model m. It then checks if this model satisfies the constraints to be met, using techniques internal to the theory solver. If the constraints cannot be met for any design represented by the model, the theory solver returns `UNSAT` and computes a subset of the propositional variables $C \subseteq M$, called the *conflict clause*, that is a reason for the unsatisfiability. The SAT solver then factors the computed conflict clauses in generating subsequent models. If on the other hand, constraints can be met, the theory solver simply returns `SAT`. This process is repeated until either the solver returns `SAT` while M represents exactly one design, or the conflict clauses deem the entire decision space infeasible. In the former case, we find a solution and in the latter prove that there is none.

B. Real-Time Calculus

Real-Time Calculus (RTC) [5], [10] is an extension of Network Calculus [11] with applications in modular timing analysis of systems. In this work, we will use it to model the workload demand of tasks and analyse fixed priority systems, in addition to verifying delay, buffer and energy constraints.

Arrival curve: An arrival function, $W(t)$, is the total number of jobs of a task that can arrive up to time t. Let $\mathbf{W}$ denote the set of all possible arrival functions of this task. Then, we characterise the arrival curve of this stream of jobs, denoted as $\alpha = (\alpha^l, \alpha^u)$, as the bounds on the number of jobs arriving in any time-interval, i.e.,

$$\alpha^l(\Delta) \leqslant W(t+\Delta) - W(t) \leqslant \alpha^u(\Delta), \quad \forall t, \Delta \geqslant 0, W \in \mathbf{W}.$$

Arrival curves can be used to represent the input patterns of different kinds of tasks, such as periodic with and without jitter, leaky bucket patterns, amongst others.

Service Curve: A service function, $Q(t)$, denotes the amount of service, in terms of processor cycles, provided by a resource up to time t. Let $\mathbf{Q}$ be the set of all possible service functions of this resource. Then, we characterise the resource availability by the service curve, denoted $\beta = (\beta^l, \beta^u)$, where

$$\beta^l(\Delta) \leqslant Q(t+\Delta) - Q(t) \leqslant \beta^u(\Delta), \quad \forall\ t, \Delta \geqslant 0, \forall\ Q \in \mathbf{Q}.$$

Similar to the arrival curves, the service curves can model a variety of resource availability patterns.

Modular analysis: The output arrival curve, denoted α' is defined similar to the arrival curve, but for the outgoing stream. The remaining service curve, denoted as β' is defined similar to the service curve, but for the amount of processing power remaining. From known results [10], α' and β' can be computed given α, β, the WCET C^u and BCET C^l.

$$\alpha'^u = \min\{(\alpha^u \otimes \beta^u/C^l) \oslash \beta^l/C^u, \beta^u/C^l\}, \tag{1}$$
$$\alpha'^l = \min\{(\alpha^l \oslash \beta^u/C^l) \otimes \beta^l/C^u, \beta^l/C^u\}, \tag{2}$$
$$\beta'^u = (\beta^u - \alpha^l C^l)\ \overline{\oslash}\ 0, \tag{3}$$
$$\beta'^l = (\beta^l - \alpha^u C^u)\ \overline{\otimes}\ 0. \tag{4}$$

For the definitions of $\otimes$, $\oslash$, $\overline{\otimes}$ and $\overline{\oslash}$, refer to [11].

The above equations enable modular analysis of large systems. The arrival curve of the first task of a task-chain is the arrival curve of the application and is an input parameters. The service curve of the highest priority task of a resource is the service curve of that resource, and is also an input parameter. The computed α' is the arrival curve in the subsequent task (if any) and the computed β' is the service curve of the next lower priority task (if any). This enables modular analysis by treating each task as a module and applying the above equations.

Worst-case bounds: Let a stream of jobs with arrival curve α be served by a series of tasks. The service curve available to the ith task is β_i and the WCET (BCET) of the job is C_i^u (C_i^l). Then, the worst-case end-to-end delay is

$$d^{max} = Del\left(\alpha^u, \frac{\beta_1^l}{C_1^u} \otimes \ldots \otimes \frac{\beta_n^l}{C_n^u}\right). \tag{5}$$

The worst-case buffer level at the input of the ith task is

$$b_i^{max} = Buf(\alpha^u, \beta_i^l/C_i^u). \tag{6}$$

For definitions of *Del* and *Buf* refer to [10]. Let $\mathbf{T}$ be the set of tasks executing on the ith resource. The worst-case energy consumption in any interval of length Δ of the ith resource is

$$e_i^{max}(\Delta) = \sum_{j \in \mathbf{T}} \left(\alpha_j'^u(\Delta) \frac{C_j^u}{s_i}(\phi_i(s_i) - \phi_i(0))\right) + \phi_i(0)\Delta. \tag{7}$$

Using the above results from RTC, given a fully specified system (with a speed assignment), we can check the satisfaction of the different delay, buffer and energy constraints.

V. Custom SMT solver with RTC as background theory solver

In this section, we present the main contribution of our work, namely a custom SMT solver with the Real-Time Calculus (RTC) as the background theory. We show how this solver can be used for the problem of speed assignment on a distributed system with delay, buffer and energy constraints.

The SMT solver is implemented using OpenSMT [7]. As a theory solver we use the publicly-available Modular Performance Analysis (MPA) [8] toolbox for MATLAB.

A. Propositional variables

As discussed in the previous section, the SAT part of the SMT solver works only with the propositional variables. Thus, we must encode the entire decision space in terms of these variables. Let $S_{i,j}$ is the jth lowest speed in set $\mathbf{S}_i$. We define propositional variables p_{ij} for every resource R_i and for every speed $S_{i,j}$, except the highest speed $S_{i,|\mathbf{S}_i|}$. The inclusion of an assignment to a propositional variable in the model M is interpreted as follows.

$$(p_{ij} \mapsto \top) \in M \Rightarrow s_i \leqslant S_{i,j} \tag{8}$$
$$(p_{ij} \mapsto \bot) \in M \Rightarrow s_i > S_{i,j} \tag{9}$$

Recall that S_{ij} is the jth lowest speed in $\mathbf{S_i}$. Imposing these condition, given the above interpretation, the propositional variables need to satisfy following relations.

$$(p_{ij} \mapsto \top) \in M \Rightarrow (p_{ik} \mapsto \top) \in M, \ \forall\, k > j, \forall\, i \tag{10}$$
$$(p_{ij} \mapsto \bot) \in M \Rightarrow (p_{ik} \mapsto \bot) \in M, \ \forall\, k < j, \forall\, i. \tag{11}$$

B. Interpreting model

The theory solver must interpret the model M into a theory-specific model m. As discussed, model M can have some propositional variables unassigned. We interpret such an incomplete representation by a model m which specifies speeds of resources by sets. For example, in the first solver call of Fig. 3, the model M is empty which is interpreted as all possible speed assignments in m. In contrast, in the second solver call, the model M is complete representing only one specific assignment of speeds in m.

We denote sets of speeds with a tilde accent. Thus, the set of speeds of resource R_i is denoted as $\widetilde{s_i}$. This is an abstraction that is imposed by incomplete models. The upper and lower bounds of such a set are denoted as $(\widetilde{s_i})^u$ and $(\widetilde{s_i})^l$, respectively. Given the propositional variables, these bounds on the speeds can be computed by the following relations.

$$(\widetilde{s_i})^u = S_{i,j}, j = \min(\{k \mid (p_{ik} \mapsto \top) \in M\} \cup \{|\mathbf{S}_i|\}) \tag{12}$$
$$(\widetilde{s_i})^l = S_{i,j}, j = \max(\{k \mid (p_{ik} \mapsto \bot) \in M\} \cup \{0\}) + 1 \tag{13}$$

C. Analysing incomplete model

Having identified the speeds of the resources as sets, we have to propagate this *incompleteness* to the other quantities in the semantics of the RTC solver, namely arrival and service curves, and by extension the delay, buffer and energy bounds. We denote the arrival curve for the task $T_{i,j}$ as $\alpha_{i,j}$ and the service curve provided to this task as $\beta_{i,j}$. We continue to use the tilde accent to represent sets of the quantities. For instance, $\widetilde{\alpha_{i,j}^u}$ denotes the set of all arrival curves of task $T_{i,j}$ admissible by an incomplete speed assignment.

The interpretation of these quantities is counter-intuitive to the way they are conventionally used in timing analysis. For instance, consider a service curve that is remaining after a full resource serves a high priority task of period 2 and execution time 1. We plot this service curve given that the speed of the processor can vary in the range $[0.5, 1]$ in Fig. 4. Here $(\widetilde{\beta^l})^u$ ($(\widetilde{\beta^u})^l$) is the lower (upper) service curve, maximised (minimised) across different allowed speeds. It is plausible, indeed as is the case in Fig. 4, that $(\widetilde{\beta^l})^u$ is not smaller than $(\widetilde{\beta^u})^l$. This is contrary to the conventional notion of using

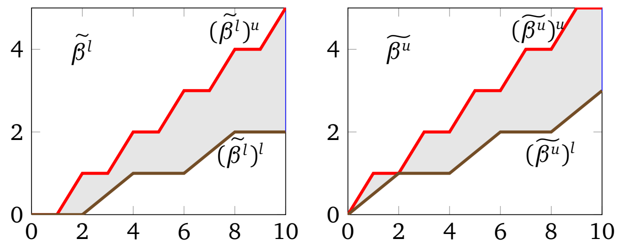


Fig. 4. Service curves for an incomplete model. Gray area denotes uncertainty due to the incompleteness

curves in RTC, where upper bounds are always above lower bounds. However, the abstraction of the incomplete models, can change this. The definition of these new curves (with tildes) forms the key step in the use of RTC as a background theory solver.

Let d_i^{max} denote the worst-case end-to-end delay of application A_i, $b_{i,j}^{max}$ denote the worst-case buffer requirement on the input of task $T_{i,j}$, and e_i^{max} denote the worst-case energy consumption of resource R_i in any interval of length τ_i. Interpreting incomplete models can lead to sets of these quantities, which are also represented with a tilda accent.

In any solver call, the solver must return `UNSAT`, if and only if for that model, the constraints are violated for all speed assignments represented by the model. This can be shown to be the case only if

1) $(\widetilde{d_i^{max}})^l > D_i$, for some application A_i, or
2) $(\widetilde{b_{ij}^{max}})^l > B_{ij}$, for some task T_{ij}, or
3) $(\widetilde{e_i^{max}})^l > E_i$, for some resource R_i.

To ascertain these, we need to compute the above quantities. It can be shown that these quantities are given by the following result.

$$\begin{aligned}
(\widetilde{d_i^{max}})^l &= \mathit{Del}((\widetilde{\alpha_{i,1}^u})^l, (\widetilde{\beta_{i,1}^l})^u/C_{i,1}^u \otimes \ldots \otimes (\widetilde{\beta_{i,n}^l})^u/C_{i,n}^u) \\
(\widetilde{b_{i,j}^{max}})^l &= \mathit{Buf}((\widetilde{\alpha_{i,j}^u})^l, (\widetilde{\beta_{i,j}^l})^u/C_{i,j}^u) \qquad (14) \\
(\widetilde{e_i^{max}})^l &= \sum_{\{(x,y)|m_{x,y}=i\}} (\widetilde{\alpha_{x,y}'^u})^l(\tau_i)\frac{C_{x,y}^u}{(\widetilde{s_i})^l}(\phi_i((\widetilde{s_i})^l) - \phi_i(0)) \\
&\qquad + \phi_i(0)\tau_i
\end{aligned}$$

Note that in the right hand sides of all above equations only lower-bounds of sets for upper curves and upper-bounds of sets for lower curves are involved. For instance, for $\widetilde{\alpha^u}$ only $(\widetilde{\alpha^u})^l$ is involved, and not $(\widetilde{\alpha^u})^u$. For brevity, we call these bounds the *cross-bounds*.

We now describe the computation of the cross-bounds. The arrival curve of the first task of every application is set to the arrival curve of that application: $(\widetilde{\alpha_{i,1}^u})^l = \alpha_i^u, (\widetilde{\alpha_{i,1}^l})^u = \alpha_i^l$. Let $T_{x,y}$ be the highest priority task of resource R_i. Then, the service curve provided to $T_{x,y}$ is $(\widetilde{\beta_{x,y}^u})^l(\Delta) = (\widetilde{s_i})^l \cdot \Delta, (\widetilde{\beta_{x,y}^l})^u(\Delta) = (\widetilde{s_i})^u \cdot \Delta$. Again, note that, in contrast to intuition from RTC, the cross-bound of a lower service curve can be larger than the cross-bound of the upper service curve. Consequently, this applies to other derived cross-bounds.

We re-formulate the standard equations (1)-(4) to compute the other cross-bounds. The proofs for these depend on manipulation of the RTC operators.

$$\begin{aligned}
(\widetilde{\alpha'^u})^l &= \min\{((\widetilde{\alpha^u})^l \otimes (\widetilde{\beta^u})^l/C^l) \oslash (\widetilde{\beta^l})^u/C^u, (\widetilde{\beta^u})^l/C^l\}, \\
(\widetilde{\alpha'^l})^u &= \min\{((\widetilde{\alpha^l})^u \oslash (\widetilde{\beta^u})^l/C^l) \otimes (\widetilde{\beta^l})^u/C^u, (\widetilde{\beta^l})^u/C^u\}, \\
(\widetilde{\beta'^u})^l &= ((\widetilde{\beta^u})^l - (\widetilde{\alpha^l})^u C^l) \overline{\oslash} 0, \\
(\widetilde{\beta'^l})^u &= ((\widetilde{\beta^l})^u - (\widetilde{\alpha^u})^l C^u) \overline{\otimes} 0.
\end{aligned}$$

Again, the right hand sides only involve cross-bounds. In other words, working with only the cross-bounds suffices for interpreting and analysing partial models M, in terms of verifying satisfiability of the considered constraints. This is a key property of Real-Time Calculus (RTC) which enables interfacing with a SAT solver.

D. Generating conflict clause

A conflict clause must be generated whenever a delay, buffer or energy constraint is violated. The conflict clause must identify the subset of the propositional variables assigned in the model which caused the violation. We will now discuss the conflict clauses for violations of the different constraints.

Delay constraint: Violation of the delay constraint for an application implies that the service available to its tasks were smaller than necessary. For each resource R_i on which tasks of the considered application are mapped, low service is due to (a) smaller value of $(\widetilde{s_i})^u$, and (b) larger value of the $(\widetilde{\alpha_{x,y}^u})^l$, where $T_{x,y}$ is any higher priority task on R_i. In turn, the larger value of $(\widetilde{\alpha_{x,y}^u})^l$ is due to (a) larger value of $(\widetilde{s_j})^l$ and smaller value of $(\widetilde{s_j})^u$, where $j = m_{x,y-1}$, and (b) larger values of $(\widetilde{\alpha_{u,v}^u})^l$ and small values of $(\widetilde{\alpha_{u,v}^l})^u$, where $T_{u,v}$ is any task of priority higher than $T_{x,y-1}$ on R_j, and (c) higher value of $(\widetilde{\alpha_{x,(y-2)}^u})^l$, if there is are preceding tasks $T_{x,(y-1)}$ and $T_{x,(y-2)}$. This recursive process is repeated until we arrive on the first tasks of application whose arrival curves are fixed. In this process, note that the violation is attributed to only smaller (not larger) values of $(\widetilde{s_i})^u$ and larger (not smaller) values of $(\widetilde{s_i})^l$. For resource R_i, whenever the violation is attributed to lower value of $(\widetilde{s_i})^u$, the conflict clause includes all variables p_{ij} such that $\{p_{ij} \mapsto \top\} \in M$. Similarly, for higher values of $(\widetilde{s_i})^l$ we include in the conflict clause all variables p_{ij} such that $\{p_{ij} \mapsto \bot\} \in M$.

Buffer constraint: Let the input buffer for task $T_{i,j}$ be violated. This can be attributed to (a) larger value of $(\widetilde{\alpha_{i,j}^u})^l$, (b) smaller value of $(\widetilde{s_i})^u$, and (c) larger value of $(\widetilde{\alpha_{x,y}^u})^l$, where $T_{x,y}$ is any higher priority task on the resource. As in the case of delay constraints, recursively we elaborate these bounds to obtain the conflict clause.

Energy constraint: If the energy constraint of resource R_i is not met, then this violation is attributed to (a) larger value of $(\widetilde{s_i})^l$, and (b) larger values of $(\widetilde{\alpha_{x,y}^u})^l$ where $T_{x,y}$ is any task with $m_{x,y} = i$. As before the conflict clause is obtained by recursively analysing these conditions.

We can formally show that, in the absence of any additional information about the problem instance, the above conflict clauses are minimal, i.e., they do not include any "unnecessary" propositional variables. Recall that the smaller the conflict clause, the larger is the benefit in terms of eliminating a large fraction of the design space.

VI. Experimental Results

In this section, we will describe the results of using the proposed solver on problem instances with large design spaces. To enable a comparison of different features of the SMT technique, we consider three variants of the solver. The solver variant S_{AB} is the one that has been discussed thus. In solver variant S_A, we trivially set the conflict clauses to be equal to the model. In solver variant S_B, we only consider models which are complete, while conflict clauses are generated as discussed in the previous section. Thus, the variants S_A and S_B are designed to understand the individual advantages of analysing incomplete models and computing compact conflict clauses, respectively.

We consider problem instances with 25 different resources and 5 distinct speed levels for each resource, amounting to a design space of $5^{25} \approx 3 \times 10^{17}$ different design points. On this architecture, we consider the execution of multiple applications. We consider different problem instances, where the number of applications (in the range of 10-25, with 3-5 tasks each), their binding to resources, their timing properties and the constraints are randomly varied. For each such problem instance, we first find the average number of solver calls needed to identify a solution when randomly exploring the design space. In all experiments, we limit our search to a maximum of 50000 solver calls.

#	Type	S_{AB}	S_A	S_B	Random
1	SAT	64	38	9	38
2	SAT	163	4509	108	122
3	SAT	77	41	30	278
4	SAT	153	406	434	980
5	SAT	289	839	12195	3166
6	SAT	259	552	1067	40771
7	SAT	366	3122	405	>50k
8	SAT	409	4269	1367	>50k
9	SAT	521	7931	>50k	>50k
10	UNSAT	1	1	>50k	>50k
11	UNSAT	80	98	>50k	>50k
12	UNSAT	306	852	>50k	>50k
13	UNSAT	416	6210	>50k	>50k

TABLE I
NUMBER OF SOLVER CALLS UNTIL TERMINATION FOR THE DIFFERENT ALGORITHMS FOR DIFFERENT PROBLEM INSTANCES

We present the results in Table I. Firstly, note that S_{AB} computes optimal results for such a large space within hundreds of solver calls, for all the considered problem instances. On a typical desktop machine, a solver call takes about 2 seconds. This confirms the efficacy of the solver for large design spaces. This result holds both for feasible and infeasible problem instances.

Secondly, the comparison between S_A and S_B is not conclusive. In some problem instances, where large fraction of the design space is feasible, S_B performs very well, even bettering S_{AB}. This can be attributed to the way S_B works: solver calls happen only for completely specified models. With large number of feasible design points, it may be more efficient to 'guess' complete models rather than learn conflict clauses while building up the model incrementally, as done in S_A and S_{AB}. Otherwise, in most problem instances, especially ones which are UNSAT, S_B performs much worse, proving ineffective in finding a solution or proving unsatisfiability.

Thirdly, S_{AB} performs much better than either S_A and S_B, for most of the problem instances. This confirms that analysis of incomplete models and the generation of conflict clauses are complementary contributories towards efficient exploration of the design space.

VII. Related Work

Other researchers have studied the use of SAT solvers for problems in system design. The problem of allocation of resources and binding of tasks to optimise performance has received particular interest [12], [13]. In these and similar works the timing analysis is restricted to specific nature of the problem, for instance fixed priority periodic scheduling in [12]. In contrast, our approach is the first instance of a generic modular performance analysis tool being coupled with a SAT solver. This enables us to inherit the established advantages of Real-Time Calculus (RTC) to analyse different properties such as delay, buffer and energy, for various task models, scheduling policies and mapping decisions.

Reimann *et al* in [14] and [15] solve the allocation and binding problem while specifically using RTC as the theory solver. Similar to our approach, they also compute conflict clauses to guide the SAT solver [15], though the specific problem is different. However, the authors do not explicitly analyse incomplete models in RTC, which in our approach required the analysis of sets of curves (for instance $\widetilde{\alpha^u}$). Instead, the specific problem of binding and allocation is staged with intermediate solver calls for subsets of tasks to be bound [14]. The incomplete models generated by the SAT solver are in fact complete theory-models, albeit of smaller problem instances. We believe that our approach of specifying incomplete models and establishing the necessary conditions for their analysis using standard RTC operations is a key contribution of our work, and is broader in its scope of application. Specially, as demonstrated in our results (S_B vs S_{AB}), the efficiency of the solver can be much worse without such a feature.

VIII. Conclusion

In this work, we have shown that it is possible to couple a SAT solver with the background theory of Real-Time Calculus (RTC). This was specifically enabled by analysis of incomplete models, by representing curves in RTC by sets. As a key advantage, we showed that it suffices to compute with *cross-bounds*. We applied this solver framework to the specific problem of assigning speeds of resources in a distributed real-time system with delay, buffer and speed constraints. For this problem, we showed how we can compute compact conflict clauses. For several problem instances considered, on very large decision spaces, the performance of the solver was encouraging. The outlook of this work is the strong motivation to design custom SMT solvers for other optimisation problems one encounters in the design of distributed systems, where traditional methods do not perform as well.

Acknowledgements

This work is supported in part by the EU FP7 projects EURETILE and PRO3D, under grant numbers 247846 and 248776, respectively.

References

[1] "National workshop on high-confidence automotive cyber-physical systems." http://varma.ece.cmu.edu/Auto-CPS/, April 2008.

[2] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *ICCAD*, 2004.

[3] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ICSA*, 2007.

[4] S. Malik and L. Zhang, "Boolean satisfiability from theoretical hardness to practical success," *Commun. ACM*, vol. 52, no. 8, pp. 76–82, 2009.

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

[6] L. de Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[7] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich, "The opensmt solver," in *TACAS*, pp. 150–153, Springer, 2010.

[8] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox." http://www.mpa.ethz.ch/Rtctoolbox, 2006.

[9] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, IOS Press, 2009.

[10] E. Wandeler, *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, ETH Zurich, 2006.

[11] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, vol. 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[12] A. Metzner and C. Herde, "Rtsat– an optimal and efficient approach to the task allocation problem in distributed architectures," in *RTSS*, pp. 147–158, 2006.

[13] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1382–1389, 2011.

[14] F. Reimann, M. Glaß, C. Haubelt, M. Eberl, and J. Teich, "Improving platform-based system synthesis by satisfiability modulo theories solving," in *CODES+ISSS*, pp. 135–144, 2010.

[15] F. Reimann, M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Symbolic system synthesis in the presence of stringent real-time constraints," in *DAC*, pp. 393–398, 2011.