IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 19, 2018, accepted September 19, 2018, date of publication October 8, 2018, date of current version October 31, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2874637

Constraint Solving Approach to Schedulability Analysis in Real-Time Systems

HYUK LEE[1] AND JIN-YOUNG CHOI[2]
[1]Department of Computer Science, Korea University, Seoul 02841, South Korea
[2]Graduate School of Information Security, Korea University, Seoul 02841, South Korea

Corresponding author: Jin-Young Choi (choi@formal.korea.ac.kr)

This work was supported in part by the Ministry of Science and ICT (MSIT), Korea, through the Information Technology Research Center Support Program supervised by the Institute for Information & Communications Technology Promotion (IITP) under Grant 2015-0-00445 and in part by the IITP Grant through the Korea Government (MSIT) [Development of High-Assurance(≥EAL6) Secure Microkernel] under Grant 2018-0-00532.

ABSTRACT In real-time systems, the satisfaction of real-time properties is as important as the correct behavior of the function. There are many safety-critical systems among the real-time systems, and thus, the satisfaction of real-time properties is directly related to safety in those cases. By performing the schedulability analysis, we can predict the behavior of real-time systems and ensure that real-time properties are met. In this paper, we propose the schedulability analysis of a real-time system through a constraint solving approach, that is, by treating a scheduling problem as a constraint solving problem. To do this, we describe a method of representing the task behavior and schedulable properties of real-time tasks in the form of constraints and finding answers that satisfy all the constraints using a constraint solver.

INDEX TERMS Constraint satisfaction problem, satisfiability modulo theories, real-time schedulability analysis.

I. INTRODUCTION

There are various safety-critical systems in modern society. The control systems used in automotive, railway, medical, and nuclear sector are all safety-critical systems. Errors or malfunctions in such safety-critical systems can result in property damage and loss of life. Most of these systems are real-time systems with real-time properties. In a real-time system, not only the correct behavior of the function, but also the satisfaction of real-time properties is very important [1]. That is, the timing of the results, as well as the results of the functions, must be correct. A real-time system is defined by a set of tasks and a scheduling policy, where a task is given by a triple of a period, computation time, and deadline [2]. In a hard real-time system, tasks that miss deadlines are no different than tasks with incorrect results, which can cause the function to fail. In order to guarantee the real-time properties of a system, it is necessary to be able to predict system behavior, which is possible through the schedulability analysis of the system tasks [3].

Since the pioneering work by Liu and Layland [4], there have been many studies on schedulability analysis [5]–[10]. In addition to these, other studies based on formal techniques such as process algebra [2], [11]–[13], state machine [14], [15] and constraint solving [16], [17] have been conducted. These studies, which are based on formal techniques, have tended to focus on specific task sets or specification techniques. Our motivation of this paper is to present how to transform a real-time system into a set of constraints, and that a real-time schedulability can be analyzed by showing satisfiability of such a set of constraints. In the constraint solving approach proposed in this paper, the real-time scheduling problem is represented in the form of simple, clear, and strict logical expressions using first-order logic. In particular, we represent task behavior and schedulability conditions as logical expressions for later input into satisfiability modulo theories (SMT) solver.

First, we will define the tasks in real-time systems and define the behavior of those tasks. Here, we use a state-based framework to describe how a task behaves. In this case, the behavior of a task can be expressed as a transition between states, and a state transition can be further defined as a relation between states. Once transition relations are defined, we can extract the rules from the transition relations and encode the rules into constraints. The conditions for schedulability are defined in a similar way. Based on the state-based task behavior above, we extract rules that must be satisfied in order to allow scheduling and encode these rules as constraints. In doing so, we will have a set of constraints on task behavior

and schedulability that we need to find solutions for. These constraints are then used as input into SMT solver to obtain a solution that satisfies all of the encoded constraints. SMT solver presents a corresponding model of evaluation if all of the constraints are satisfied, otherwise, it yields *Unsatisfiable*. We introduce a simple implementation of this proposed approach using Python with Z3Py which is SMT solver API.

We believe that it is meaningful to conduct schedulability analysis using a constraint solving approach. Representing problems using first-order logic, which is a formal modeling method, is widely used in the field of verification, equivalence checking, scheduling, and optimization [18], and enables the accurate understanding and expression of problems. In addition, the logical expressions represented in this way can be solved with a fully automated SMT solver. The contributions of this paper are 1) A method for transforming a real-time system into a set of constraints of simple, clear, and strict logical expressions is proposed, 2) A way to prove that a real-time schedulability can be analyzed by showing satisfiability of such a set of constraints is presented.

The remainder of this paper is structured as follows. We discuss different methods for schedulability analysis based on formal techniques in Section II. In order to express schedulability as a series of constraints, we explain our definition of a task and state-based task behaviors in Section III. Section IV identifies the constraints according to their definitions and constructs the constraints using first-order logic. The implementation and examples of the proposed method are described in Section V. In Section VI, we present the conclusions of this paper and discuss future research.

II. RELATED WORK

Schedulability analysis methods based on mathematical approaches, such as Liu and Layland [4] and Zhang and Burns [10], do not allow much flexibility in expressing task behavior compared to behavior/simulation based approaches [2], [11]–[17]. In particular, in our proposed approach, the task behaviors are described as a set of constraints, so task behaviors can be defined as long as they can be represented by constraints. This gives us some flexibility to express various task behaviors.

Choi *et al.* [2] propose an approach to the schedulability analysis of real-time systems based on a timed process algebra called ACSR-VP, which is an extension of ACSR [19] that includes value-passing communication and dynamic priorities. They describe a form of the schedulability analysis that checks for bisimulation. Kwak *et al.* [11] also present schedulability analysis using ACSR-VP process algebra to produce linear equation constraints. However, they do not attempt to solve these constraints in their work. Our work therefore primarily focuses on expressing schedulability as a series of constraints and finding satisfactory answers that satisfy these constraints using appropriate constraint solvers.

Cheng and Zhang [17] address the scheduling problem of overloaded tasks using SMT solver with a goal to maximize the total number of tasks that meet their deadlines for a given set of tasks. In this case, they propose SMT-based scheduling algorithm that maximizes the number of schedulable tasks. However, our work is aimed at scheduling hard real-time tasks using conventional scheduling disciplines. Therefore, the tasks in our target hard real-time systems should not be so overloaded that they cannot all be scheduled. Our proposed approach focuses on the satisfiability of a task model for a certain scheduling discipline.

Our work follows a similar structure to that of the task model in Choi *et al.*'s work [2], but uses a different form of logical representation. Unlike Cheng *et al.*'s work [17], the target of our approach is hard real-time tasks in safety-critical real-time systems.

III. SYSTEM DEFINITION AND BACKGROUND

A. TASK DEFINITION

A real-time system consists of a set of real-time tasks $T = (\tau_1, \tau_2, \cdots, \tau_n)$. Task τ_i, where $\tau_i \in T$, can be denoted as $\tau_i = (\mathsf{p}_i, \mathsf{c}_i, \mathsf{d}_i)$ in which p_i is the period, c_i is the worst-case computation time, and d_i is the deadline for task τ_i respectively. If the deadline is equal to the period, the deadline can be omitted as $\tau_i = (\mathsf{p}_i, \mathsf{c}_i)$. The computation time is the number of jobs a task must perform within a given time (i.e., the deadline) and a job is a schedulable unit of work. The deadline is the time that the task should complete its jobs after the release of a task. If the execution of task τ_i must be completed by a given deadline d_i (i.e., $\mathsf{c}_i \leq \mathsf{d}_i$), then task τ_i is considered a *hard real-time* task. The period is the minimum time interval between the release of jobs. A periodic task is that the task is regularly invoked with the interval of the fixed time term. There is a certain behavioral pattern that repeats infinitely when scheduling periodic tasks. This behavior pattern also has a period and is also known as a hyper-period [20]. The hyper-period is the minimum time interval in which the periodic pattern of all tasks is repeated. It is usually defined as the least common multiple (LCM) of all task periods.

Our assumption for the tasks are as follows:

- All tasks are periodic
- All tasks have the worst-case computation time
- All tasks arrive and are released at the same time
- All tasks are independent, with no shared resources.

B. STATE-BASED TASK BEHAVIOR

The state of a system can be viewed as a snapshot of all of the variables and conditions that describe the system at any given moment [21]. In state-based task behavior, we define the *state* of a task as the current status of the accumulated execution time and the elapsed time of a task. We denote the variables that represent the accumulated execution time and the elapsed time of a task as *ac* and *et*, respectively, and refer to them as *state variables*.

We define state-based task behavior as follows:

$$[CurrentState] \rightarrow [StatePriority : NextState]$$

TABLE 1. Symbols and definitions.

Symbol	Definition
T	set of tasks, $T = \{\tau_1, \tau_2, \cdots, \tau_n\}$
τ_i	index of task, $\tau_i \in T$
p_i	period of task τ_i
c_i	computation time of task τ_i
d_i	deadline of task τ_i
ac_i	accumulated execution time of task τ_i
et_i	elapsed time of task τ_i after release
ac_i^k	accumulated execution time of task τ_i at k-th step
et_i^k	elapsed time of task τ_i at k-th step after release
k	step index
x	goal step index
dl	deadlock flag
ES	execution state
ES_i	execution state of task τ_i
ES^k	ES at k-th step of all tasks; $ES^k = (ac_1^k, et_1^k, \cdots, ac_n^k, et_n^k)$
ES_i^k	ES at k-th step of task τ_i; $ES_1^k = (ac_1^k, et_1^k)$

The condition of *CurrentState* decides the *NextState* to follow, and the *NextState* of the task is determined after a comparison with *StatePriority* of tasks for a preemption.

The *CurrentState* of a task can be one of the following four states, depending on the state variables:

- *Start* state ($ac = \mathsf{c} \wedge et = \mathsf{p}$) The task has completed its execution time and has reached the end of the current period. A task in this state can start a new period in the next state.
- *Deadlock* state ($ac < \mathsf{c} \wedge et = \mathsf{d}$) The task has reached its deadline but has not completed its execution time. This indicates that the given task set is not schedulable under the specified scheduling algorithm because one task is not in a schedulable condition.
- *Wait* state ($ac = \mathsf{c} \wedge et < \mathsf{p}$) The task has completed its execution time and has not reached the end of the current period. A task in this state has to wait until the end of the period to begin a new period.
- *Ready* state ($ac < \mathsf{c} \wedge et < \mathsf{d}$) The task has not completed its execution time and the deadline has not been reached. A task in this state can execute its own jobs.

The *NextState* corresponding to each possible *CurrentState* is described below. Let ES_i^k denote the execution state of task τ_i at certain point k. We denote *CurrentState* as ES_i^k, and *NextState* as ES_i^{k+1} for task τ_i.

- $[Start] \rightarrow [\alpha, 1 : StartNext]$ When a task in the *Start* state is selected, the state variables are initialized at *StartNext* and a new period is started. There are no changes to the state variables for other tasks. The *NextState* for the *Start* state can be defined as:
  - $ES_i^{k+1} = (\cdots, ac_{i-1}, et_{i-1}, 0, 0, ac_{i+1}, et_{i+1}, \cdots)$
- $[Deadlock] \rightarrow [\alpha, 2 : DeadlockNext]$ When a task in the *Deadlock* state is selected, the task execution cannot proceed any further. The given set of tasks can not be scheduled. *DeadlockNext* is:
  - $ES_i^{k+1} =$ DEADLOCK
- $[Wait] \rightarrow [\beta, 0 : WaitNext]$ When a task in the *Wait* state is selected, the task only consumes the elapsed time at *WaitNext*. All other tasks also only consume the elapsed time. *WaitNext* is:
  - $ES_i^{k+1} = (\cdots, ac_{i-1}, et_{i-1} + 1, ac_i, et_i + 1, ac_{i+1}, et_{i+1} + 1, \cdots)$
- $[Ready] \rightarrow [\beta, \pi_i : ReadyNext]$ When a task in the *Ready* state is selected, the task consumes both the execution time and the elapsed time at *ReadyState* All the other tasks only consume the elapsed time. *ReadyNext* is:
  - $ES_i^{k+1} = (\cdots, ac_{i-1}, et_{i-1} + 1, ac_i + 1, et_i + 1, ac_{i+1}, et_{i+1} + 1, \cdots)$

We have described the state-based task behavior for each possible configuration of the state variables. In a uniprocessor environment, only one task can be executed at a time. In the case of *NextState* described above, only one task can transition to *NextState*.

The preemption rule of *StatePriority* ultimately determines the task to be transitioned to *NextState*. A task that is assigned *StatePriority* α takes precedence over β regardless of the N in *StatePriority*. If two tasks are assigned the same *StatePriority* (α or β), the task with a higher N gets to transition. If there are more than two tasks assigned with exactly the same *StatePriority* and N, then the task is chosen *nondeterministically*. *StatePriority* π_i in *ReadyState* is determined by the scheduling discipline.

Let the preemption rule of *StatePriority* be denoted as $\gg, \ll$. We write $(\alpha \gg \beta)$ to indicate α preempts β

- $(\alpha, N + i) \gg (\alpha, N)$
- $(\alpha, *) \gg (\beta, *)$
- $(\beta, N + i) \gg (\beta, N)$, *where* $(i > 0)$

Suppose, at some point, that two tasks have reached a certain point in time and are in the following states:

- Task A in *Start state* vs. Task B in *Deadlock state*
  - Task B transitions to *DeadlockNext*
- Task A in *Start state* vs. Task B in *Ready state*
  - Task A transitions to *StartNext*
- Task A in *Ready state* vs. Task B in *Ready state*
  - If the priority π of Task A is higher than Task B
    - Task A transitions to *ReadyNext*.

Algorithm 1 describes the overall state-based behavior of a task.

Timing graph presented as Fig. 1 shows the typical scheduling behavior for the example task set T_1 under Earliest Deadline First (EDF) scheduling algorithm where, $T_1 = (\tau_1, \tau_2)$, $\tau_1 = (2, 1)$, $\tau_2 = (3, 1)$. The task set T_1 is supposedly schedulable under an EDF because the utilization bound [4] of task set T_1 is sufficient, where $U = 1/2 + 1/3 = 5/6 = 0.83 \leq 1$.

Table. 2, describes the state-based behavior of task set T_1. The scheduling starts at ES^0, (0, 0, 0, 0). To briefly describe the state-based behavior of T_1, the scheduling starts at the initial step, where all state variables are at their default value.

Algorithm 1 State-Based Behavior of a Task

Input: Task variables of task $\tau_i = (\mathbf{p}_i, \mathbf{c}_i, \mathbf{d}_i)$
Initiate: $(ac_i \leftarrow 0, et_i \leftarrow 0)$
while *NextState* $\neq$ *DEADLOCK* **do**
  if τ_i *is selected* **then**
    if $(ac_i = \mathbf{c}_i \wedge et_i = \boldsymbol{p}_i)$ **then** // *Start*
      assign *StatePriority* of τ_i as $(\alpha, 1)$
      assign *NextState* of τ_i as $(0, 0)$
    else if $(ac_i < \mathbf{c}_i \wedge et_i = \boldsymbol{d}_i)$ **then** // *Deadlock*
      assign *StatePriority* as $(\alpha, 2)$
      assign *NextState* of τ_i as DEADLOCK
      break
    else if $(ac_i = \mathbf{c}_i \wedge et_i < \boldsymbol{p}_i)$ **then** // *Wait*
      assign *StatePriority* as $(\beta, 1)$
      assign *NextState* of τ_i as $(ac_i, et_i + 1)$
    else if $(ac_i < \mathbf{c}_i \wedge et_i < \boldsymbol{d}_i)$ **then** // *Ready*
      assign *StatePriority* as (β, π_i)
      assign *NextState* of τ_i as $(ac_i + 1, et_i + 1)$
  end
end

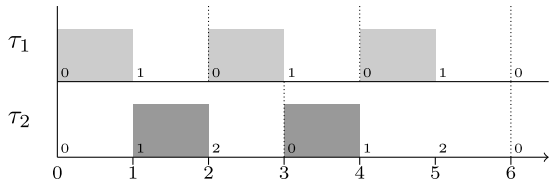

FIGURE 1. Typical scheduling behavior of T_1 under EDF

TABLE 2. State-based behavior of T_1 under EDF.

Current State	τ_1 State	τ_2 State	State Priority	Next State
$ES^0, (0,0,0,0)$	Ready	Ready	$\tau_1 > \tau_2$	$ES^1, (1,1,0,1)$
$ES^1, (1,1,0,1)$	Wait	Ready	$\tau_1 < \tau_2$	$ES^2, (1,2,1,2)$
$ES^2, (1,2,1,2)$	Start	Wait	$\tau_1 > \tau_2$	$ES^3, (0,0,1,2)$
$ES^3, (0,0,1,2)$	Ready	Wait	$\tau_1 > \tau_2$	$ES^4, (1,1,1,3)$
$ES^4, (1,1,1,3)$	Wait	Start	$\tau_1 < \tau_2$	$ES^5, (1,1,0,0)$
$ES^5, (1,1,0,0)$	Wait	Ready	$\tau_1 < \tau_2$	$ES^6, (1,2,1,1)$
$ES^6, (1,2,1,1)$	Start	Wait	$\tau_1 > \tau_2$	$ES^7, (0,0,1,1)$
$ES^7, (0,0,1,1)$	Ready	Wait	$\tau_1 > \tau_2$	$ES^8, (1,1,1,2)$
$ES^8, (1,1,1,2)$	Wait	Wait	$\tau_1 = \tau_2$	$ES^9, (1,2,1,3)$
$ES^9, (1,2,1,3)$	Start	Start	$\tau_1 = \tau_2$	$ES^{10}, (0,0,1,3)$
$ES^{10}, (0,0,1,3)$	Ready	Start	$\tau_1 < \tau_2$	$ES^{11}, (0,0,0,0)$
$ES^{11}, (0,0,0,0)$	···	*Repeat*	*from*	*beginning* ···

At this step, both tasks are in the *Ready* state, and τ_1 has a higher priority than τ_2. Therefore, *NextState* is selected as the *ReadyNext* state of τ_1, which is $ES^1, (1, 1, 0, 1)$. This process is repeatedly executed following the behavior rules of Algorithm 1.

C. SCHEDULING DISCIPLINES

Scheduling disciplines are algorithms that determine how resources are allocated. Some examples are the Rate Monotonic (RM), Earliest Deadline First (EDF), and Least Laxity First (LLF) algorithms, the choice of which depends on the type of application being run on the system. RM is a scheduling algorithm that statically allocates the priority of tasks, whereas EDF and LLF are algorithms that dynamically allocate priorities. Task priority π_i in *StatePriority* is determined by the specific scheduling algorithm. The following formulas are used to calculate task priority:

- RM: $\mathsf{d}_{max} - \mathsf{p}_i$
- EDF: $\mathsf{d}_{max} - (\mathsf{d}_i - et_i)$
- LLF: $\mathsf{d}_{max} - (\mathsf{d}_i - et_i) - (\mathsf{c}_i - ac_i)$
  where $\mathsf{d}_{max} = 1 + max(\mathsf{d}_1, \cdots, \mathsf{d}_n)$

IV. ENCODING CONSTRAINTS

In this section, we describe constraint encoding for schedulability analysis. The encoding of constraints is divided into three steps. The first step is the encoding of constraints that limit the scope of the free variables used in the task behavior representation. The second step is the encoding of the state transitions rule according to the behavior rules defined in Algorithm 1. Finally, the third step is encoding constraints on the schedulability that a given set of tasks must ultimately satisfy.

A. VARIABLE BINDING CONSTRAINTS

The task information, such as the period, computation time and deadline (i.e., p, c, and d), is given as constants. However, other variables that represent the behavior and constraints of the task are declared as free variables. These free variables need to be constrained in order to be executed accurately.

The free variables used to represent state-based task behavior are:

- Accumulated execution time (ac)
- Elapsed time (et)
- Step index (k)
- Goal step index (x)
- Deadlock flag (dl).

The accumulated execution time is the number of jobs completed. Therefore, the variable representing the accumulated execution time cannot exceed the computation time or be negative. Based on these attributes, the constraints on the accumulated execution time can be defined as (1):

$$acScope = \bigwedge_{i=1}^{n} \left(ac_i^k \leq \mathsf{c}_i \wedge ac_i^k \geq 0 \right) \tag{1}$$

Similarly, the variable representing the elapsed time cannot exceed the period or be negative. Based on these attributes, the constraint on the elapsed time can be defined as (2):

$$etScope = \bigwedge_{i=1}^{n} \left(et_i^k \leq \mathsf{p}_i \wedge et_i^k \geq 0 \right) \tag{2}$$

In state-based task behavior, the state of a task can be expressed by the status of the state variables. However, the state of a task only reflects the status of the state variables based on relative time rather than absolute time. Therefore, we use the step variables k and x, representing the step index and the goal step index, respectively, to indicate the state transition. The step index cannot exceed the goal step index or be negative. Based on these attributes, the constraint on the step index can be defined as (3):

$$kScope = (k \geq 0 \wedge k \leq x) \tag{3}$$

In addition to the state variables and step variables, we used a flag variable to indicate whether the state transition is in a *Deadlock* state. The variable used to represent the *Deadlock* state in the state transition is denoted as dl. The deadlock flag can either be *true* or *false*. Based on this attribute, the constraint on the deadlock flag can be defined as (4):

$$dlScope = (dl^k \geq 0 \wedge dl^k \leq 1) \tag{4}$$

So far, we have presented the constraint encodings for the variables used in the schedulability analysis. This constrains the scope of the variables that represent state-based behavior.

Suppose we have an example task set $T_1 = (\tau_1, \tau_2)$, $\tau_1 = (2, 1)$, $\tau_2 = (3, 1)$. Let $ES(T_1)_p$ denote a possible execution state for T_1. Based on a 32 bit integer, the total number of possible execution states for T_1 before variable binding would be $|ES(T_1)_p| = 1.717986918 * 10^{10}$, which represents the product of the four integer variables (ac_1, et_1, ac_2, and et_2), when each variable has the scope of the entire integer range. However, after constraining the free variables, the number of possible execution states reduces significantly. For task set T_1, the maximum number of possible states is 48, as shown in (5):

$$ES(T_1)_p = \begin{Bmatrix} (0,0,0,0) & (0,0,0,1) & (0,0,0,2) & (0,0,0,3) \\ (0,0,1,0) & (0,0,1,1) & (0,0,1,2) & (0,0,1,3) \\ (0,1,0,0) & (0,1,0,1) & (0,1,0,2) & (0,1,0,3) \\ (0,1,1,0) & (0,1,1,1) & (0,1,1,2) & (0,1,1,3) \\ (0,2,0,0) & (0,2,0,1) & (0,2,0,2) & (0,2,0,3) \\ (0,2,1,0) & (0,2,1,1) & (0,2,1,2) & (0,2,1,3) \\ (1,0,0,0) & (1,0,0,1) & (1,0,0,2) & (1,0,0,3) \\ (1,0,1,0) & (1,0,1,1) & (1,0,1,2) & (1,0,1,3) \\ (1,1,0,0) & (1,1,0,1) & (1,1,0,2) & (1,1,0,3) \\ (1,1,1,0) & (1,1,1,1) & (1,1,1,2) & (1,1,1,3) \\ (1,2,0,0) & (1,2,0,1) & (1,2,0,2) & (1,2,0,3) \\ (1,2,1,0) & (1,2,1,1) & (1,2,1,2) & (1,2,1,3) \end{Bmatrix} \tag{5}$$

In the next section, constraints on state-based behavior are encoded to determine legitimate cases of transition between the possible states.

B. STATE-BASED BEHAVIOR CONSTRAINTS

All tasks behave according to Algorithm 1 rules at all steps. The behavior of a task can be expressed by the state transition, that is, the relationship between the *CurrentState* and the *NextState*; this state transition must also satisfy all of the rules of Algorithm 1. First, we specify the initial step for the state-based behavior of task scheduling.

1) CONSTRAINING THE INITIAL STEP

The initial step of task scheduling refers to the point at which scheduling begins (i.e., when the step variable k is zero). In this step, the state variables of all tasks must be initial values because the task has not been executed, and deadlock flag must be set to its default value. Based on these attributes, the constraint on the initial step can be defined as (6):

$$InitialStep = (k == 0) \rightarrow \bigwedge_{i=1}^{n} \left(ac_i^k == 0 \wedge et_i^k == 0 \wedge dl^k == 0\right) \tag{6}$$

2) CONSTRAINING THE STATE TRANSITION

As described earlier, in state-based task behavior, the task transition from *CurrentState* to *NextState* depends on the state variables and the state priority. There are certain relations that can occur between *CurrentState* and *NextState*, and we can encode these as constraints. The constraints for *NextState* corresponding to each possible state in *CurrentState* are as follows:

1) If task τ_i in a *Start* state at step k is chosen, the constraint for *StartNext* can be defined as (7):

$$StartNext = (ac_i^{k+1} == 0 \wedge et_i^{k+1} == 0) \bigwedge_{i \neq j, j=1}^{n} \left(ac_j^{k+1} == ac_j^k \wedge et_j^{k+1} == et_j^k\right) \tag{7}$$

2) If task τ_i in a *Deadlock* state at step k is chosen, the constraint for *DeadlockNext* can be defined as (8):

$$DeadlockNext = (dl^{k+1} == 1) \tag{8}$$

3) If task τ_i in a *Wait* state at step k is chosen, the constraint for *WaitNext* can be defined as (9):

$$WaitNext = \bigwedge_{i=1}^{n} \left(ac_i^{k+1} == ac_i^k \wedge et_i^{k+1} == et_i^k + 1\right) \tag{9}$$

4) If task τ_i in a *Ready* state at step k is chosen, the constraint for *ReadyNext* can be defined as (10):

$$ReadyNext = (\pi_i^k \geq \pi_j^k) \rightarrow \left((ac_i^{k+1} == ac_i^k + 1) \wedge (et_i^{k+1} == et_i^k + 1) \wedge \bigwedge_{i \neq j, j=1}^{n} \left(ac_j^{k+1} == ac_j^k \wedge et_j^{k+1} == et_j^k + 1\right)\right) \tag{10}$$

The constraints on the initial steps and the state transitions defined above determine where to start scheduling and what the next state will be. Considering the example task set T_1 and the possible states described in the previous section, the set of states that satisfy the initial step constraint is shown in (11).

Let $ES^k(T_1)_p$ denote the possible execution state of T_1 at step k.

$$ES^0(T_1)_p = \{(0, 0, 0, 0)\} \tag{11}$$

In addition, the set of states satisfying the state transition constraints at the initial step and the next step is shown in (12).

$$ES^1(T_1)_p = \{(1, 1, 0, 1)\},\ ES^2(T_1)_p = \{(1, 2, 1, 2)\} \tag{12}$$

If states have the same *StatePriority*, such as the 9th step of Fig. 1, then the next state is chosen nondeterministically. The set of states satisfying the state transition constraints at such step is shown in (13).

$$ES^9(T_1)_p = \{(0, 0, 1, 3),\ (1, 2, 0, 0)\} \tag{13}$$

C. SCHEDULABILITY CONSTRAINTS

A given set of tasks can be scheduled if all the tasks are able to execute their computation time within the deadline for all their respective periods. Also, following the previous definition of the hyper-period, a given set of tasks can be scheduled infinitely if it can be scheduled within the hyper-period.

Lemma 1: If a real-time task set T is schedulable, then the state-based behavior of T does not include any Deadlock state.

Proof: Let T be a real-time task set. To prove Lemma 1, assume T is a schedulable task set. Following the schedulability definition, for every τ, if $\tau \in T$, then τ should execute the computation time within the deadline. Therefore, τ does not transition into a *Deadlock* state. □

Lemma 2: If a periodic task set P can be scheduled up to the hyper-period, then P is infinitely schedulable.

Proof: Let P be a periodic task set. To prove Lemma 2, assume P is a schedulable task set to the hyper-period. Following the definition of the hyper-period, the scheduled periodic pattern repeats infinitely. Therefore, a periodic task set P is schedulable infinitely. □

Constraining the Goal Step: The goal step in task scheduling refers to the point at which the schedulability analysis ends. Following Lemma 2, this point is the hyper-period in which all tasks have completed their computation time and period. Therefore, a given set of tasks is schedulable if the goal step can be found (or reachable) without violating the *NoDeadlock* constraint (14). Based on these attributes, the constraint on the goal step can be defined as (15).

$$NoDeadlock = (dl^k == 0) \tag{14}$$

$$GoalStep = (k == x) \rightarrow \bigwedge_{i=1}^{n} \left(ac_i^k = \mathsf{c}_i \wedge et_i^k = \mathsf{p}_i\right) \tag{15}$$

All encodings for the schedulability constraints have thus been completed, and all of the constraints for the schedulability analysis of state-based task behavior have been also encoded. By putting all of these constraints together, the constraints on the schedulability analysis of a given task set can be defined as (16):

$$\begin{aligned} \text{SCHEDCSTR} &= InitialStep \wedge GoalStep \\ &\wedge \forall k, (kScope) \rightarrow dlScope \wedge NoDeadlock \\ &\wedge \bigwedge_{i=1}^{n} \big(StartNext \wedge DeadlockNext \\ &\wedge WaitNext \wedge ReadyNext \\ &\wedge acScope \wedge etScope\big) \end{aligned} \tag{16}$$

Theorem 3: The real-time periodic task set T is schedulable iff the state-based behavior of T satisfies the set of constraints SCHEDCSTR.

Proof: ($\Rightarrow$) : Let T be a real-time periodic task set. To prove Theorem 3, assume T is a schedulable task set. Following the Lemmas 1 and 2, the state-based behavior of T does not include any *Deadlock* state up to the hyper-period. Because the constraints, *GoalStep* and *NoDeadlock* in SCHEDCSTR do not allow any *Deadlock* state within the hyper-period, the state-based behavior of T does not contain any *Deadlock* state if SCHEDCSTR is satisfied.

($\Leftarrow$) : Let a state-based behavior of T satisfy the set of constraints SCHEDCSTR. Following the definition of *SCHEDCSTR* in this section, the schedulability constraints *GoalStep* and *NoDeadlock* should also be satisfied. Then, the state-based behavior of T does not contain any state transition to a *Deadlock* state within the hyper-period. Thus, by Lemmas 1 and 2, the given set of tasks T is schedulable. □

V. IMPLEMENTATION AND EXPERIMENTS

We implemented schedulability analysis using the constraint solving approach and related encoding mechanisms presented in the previous section. We used Python with Z3Py as the implementation language, which is the Python API for SMT solver Z3 [22]. Z3Py allows us to use logical expressions to represent the constraints in Python. Once the constraints are given, the solver returns an answer on whether the given constraints can be satisfied or not. If they can, the solver then provides an evaluation of the variables that satisfy all of the constraints, referred to as the *Model*. Otherwise, the solver returns *Unsatisfiable*. This indicates that there is no evaluation of the variables that satisfy all the constraints.

The source code presented in Fig. 2 represents part of a simple implementation example for the schedulability analysis of task set T_1 using Python with Z3Py. It includes the function and variable declarations, and the constraints on the behavior rules of the task. The results of the schedulability analysis for T_1 is shown in Fig. 3, indicating that T_1 satisfies all of the constraints. The corresponding evaluation model is represented at each step index. Another example task set T_2 is shown in Table. 3, where $T_2 = (\tau_3, \tau_4)$, $\tau_3 = (2, 1)$, $\tau_4 = (3, 2)$. The task set T_2 is supposedly not schedulable under EDF because the utilization bound of task set T_2 is insufficient, $U = 1.16 \geq 1$. In Table. 3,

```
# Function declaration
f = Function ('f',IntSort(),IntSort(),IntSort())

# state variables declaration
ac = [ Int('ac%s' % i) for i in range(2) ]
et = [ Int('et%s' % i) for i in range(2) ]

# Step variables declaration
k, x  = Ints('k x')

# Deadlock variable declaration
dl  = Int('dl')

# Solver declaration
s  = Solver()

# Period, Deadline, Computation time
p[0], d[0], c[0] = 2, 2, 1
p[1], d[1], c[1] = 3, 3, 1

acScope = ([ForAll([k],
            f(ac[i],k) >= 0, f(ac[i],k) <= c[i]
            for i in range(2)])

etScope = ([ForAll([k],
            f(et[i],k) >= 0, f(et[i],k) <= p[i]
            for i in range(2)])

InitialStep = ([And(f(ac[i],0) == 0,
                    f(et[i],0) == 0,
                    f(dl,0) == 0))
                for i in range(2)])

FinalStep = ([And(f(ac[i],x) == c[i],
                  f(et[i],x) == p[i])
              for i in range(2)])

InitDeadlock = (f(dl,0) == 0)

NoDeadlock = (ForAll([k],
              Implies(And(k >= 0, k <= x),
                      f(dl,k) == 0)))

              # ··· code omitted ··· #

s.add(acScope, etScope, InitialStep, FinalStep,
      NoDeadlock)

              # ··· code omitted ··· #
```

FIGURE 2. Python implementation of schedulability analysis.

```
Time elapsed: 0.0817639827728
Satisfiable!

Total length x is:  10
Printing model evaluation...

Step: 0      ac0: 0  et0: 0      ac1: 0  et1: 0
Step: 1      ac0: 1  et0: 0      ac1: 0  et1: 1
Step: 2      ac0: 1  et0: 2      ac1: 1  et1: 2
Step: 3      ac0: 0  et0: 0      ac1: 1  et1: 2
Step: 4      ac0: 1  et0: 1      ac1: 1  et1: 3
Step: 5      ac0: 1  et0: 1      ac1: 0  et1: 0
Step: 6      ac0: 1  et0: 2      ac1: 1  et1: 1
Step: 7      ac0: 0  et0: 0      ac1: 1  et1: 1
Step: 8      ac0: 1  et0: 1      ac1: 1  et1: 2
Step: 9      ac0: 1  et0: 2      ac1: 1  et1: 3
```

FIGURE 3. The result of schedulability analysis for task set T_1.

TABLE 3. State-based behavior of T_2 under EDF.

Current State	τ_3 State	τ_4 State	State Priority	Next State
$ES^0,(0,0,0,0)$	Ready	Ready	$\tau_3 > \tau_4$	$ES^1,(1,1,0,1)$
$ES^1,(1,1,0,1)$	Wait	Ready	$\tau_3 < \tau_4$	$ES^2,(1,2,1,2)$
$ES^2,(1,2,1,2)$	Start	Ready	$\tau_3 > \tau_4$	$ES^3,(0,0,1,2)$
$ES^3,(0,0,1,2)$	Ready	Ready	$\tau_3 < \tau_4$	$ES^4,(0,1,2,3)$
$ES^4,(0,1,2,3)$	Ready	Start	$\tau_3 < \tau_4$	$ES^5,(0,1,0,0)$
$ES^5,(0,1,0,0)$	Ready	Ready	$\tau_3 > \tau_4$	$ES^6,(1,2,0,1)$
$ES^6,(1,2,0,1)$	Start	Ready	$\tau_3 > \tau_4$	$ES^7,(0,0,0,1)$
$ES^7,(0,0,0,1)$	Ready	Ready	$\tau_3 < \tau_4$	$ES^8,(0,1,1,2)$
$ES^8,(0,1,1,2)$	Ready	Ready	$\tau_3 = \tau_4$	$ES^9,(0,2,2,3)$
$ES^9,(0,2,2,3)$	Deadlock	Start	$\tau_3 > \tau_4$	$DEADLOCK$
$ES^8,(0,1,1,2)$	Ready	Ready	$\tau_3 = \tau_4$	$ES^{9'},(1,2,1,3)$
$ES^{9'},(1,2,1,3)$	Deadlock	Start	$\tau_3 > \tau_4$	$DEADLOCK$

TABLE 4. Experiment task set information.

Utilization	2 Tasks	3 Tasks	4 Tasks
Minimum	(16,1), (16,1)	(16,1), (16,1), (16,1)	(16,1), (16,1), (16,1), (16,1)
50%	(16,3), (16,3)	(16,2), (16,2), (16,4)	(16,2), (16,2), (16,2), (16,2)
75%	(16,6), (16,6)	(16,3), (16,4), (16,5)	(16,3), (16,3), (16,3), (16,3)
100%	(16,8), (16,8)	(16,6), (16,6), (16,4)	(16,4), (16,4), (16,4), (16,4)

the state-based behavior indicates that the task state is in a *Deadlock* state in either step 9 or step 9' (i.e., either of the two nondeterministic choices), which is a violation of the schedulability constraints. Because, the solver cannot find a satisfying evaluation for T_2, it outputs *Unsatisfiable* as a result.

A. PERFORMANCE EVALUATION

We performed schedulability verification experiments on several sample tasks. All experiments were performed in the following system environments: Intel i7 CPU 3.40GHz Octa-core CPU with 8 GB RAM, Linux kernel 3.19.0-32-generic x86_64, Z3 version 4.4.2, Python 2.7.6.

Table. 4 shows the task set information used in the experiments. A total of 12 task sets are used, each with a different number of tasks and utilization. Fig. 4, Fig. 5, and Fig. 6 show the time spent, the max memory used, and the number of decisions made in the experiment for every task set, respectively.

In Fig. 6, the decision indicates that the number of tries on variables, when traversing state-space to find a satisfying answer. Within each graph, the different shaded bars represent the minimum utilization to the maximum utilization of the experiment task set.

We observed several points on following aspects through the experiment. Firstly, the performance values, such as the time spent, the maximum memory used and the number of decisions made, are proportional to the number of tasks. As the number of tasks increases, the number of combinations of states to be represented also increases, which then affects

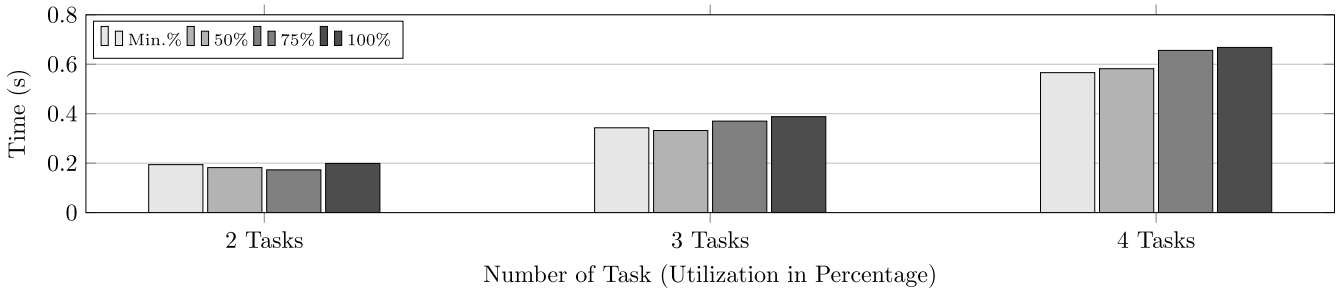

FIGURE 4. Time spent on experiment task set verification.

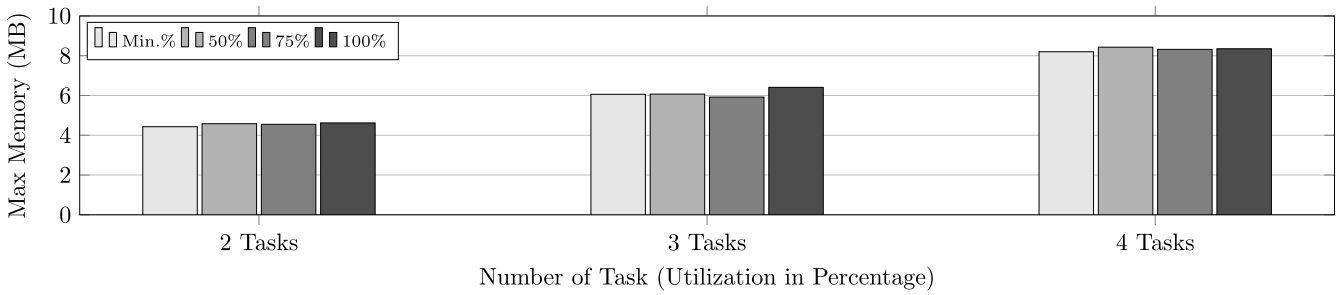

FIGURE 5. Memory used on experiment task set verification.

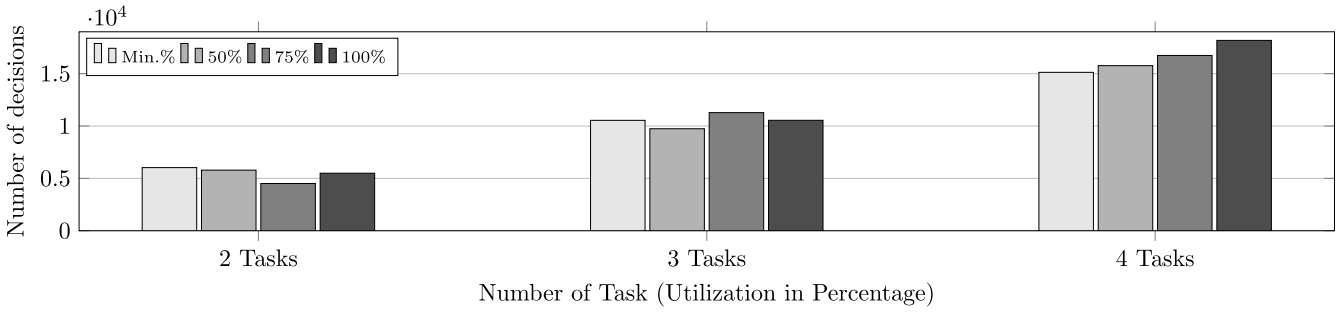

FIGURE 6. Number of decisions during experiment task set verification.

the size of the state space to be searched. With regard to task utilization, it is difficult to know how task utilization affects overall performance. The results of all experiments using the four tasks were somewhat proportional to the task utilization, but not the results for the other number of tasks. To sum up the experiment results, the memory used and the time spent indicate the size of the state space and the cost of searching that space. Even though SMT solver explores state space in efficient manner, eventually, the size of this state space depends on how effectively the constraints are expressed.

B. APPLICABILITY TO MULTICORE SYSTEM ENVIRONMENTS

Although the method proposed in this paper is based on the assumption of a uni-processor environment, it can still be extended to apply to multicore system environments.

For example, in the case of a partitioned scheduling [23]–[27], because the temporal and spatial aspects are handled separately, the proposed method can be used by adding parts that handle task assignment, which is the spatial aspect. As for the task assignment, it is possible to treat task assignment as a bin-packing problem [28], which can be solved using constraint solving approach. For a semi-partitioned scheduling [29] or a global scheduling [25], [27], where the temporal and spatial aspects are considered together, the state-based task behavior used in this paper needs to be modified to support task migration or task splitting. Because our method does not take these behaviors into account, so it may be too complicated or impossible to express without modifying current task behaviors. However, these behaviors can still be treated as constraint solving problem and can be resolved with the approach proposed in this paper. Since it is outside of our problem-solving domain of this paper, we will leave it for future work.

VI. CONCLUSION AND FURTHER WORK

A schedulability analysis can be used to predict the behavior of a system to ensure that its behavior satisfies real-time properties. In this paper, we have described the schedulability analysis of real-time system tasks using a constraint solving

approach. In order to express the scheduling problem as a series of constraints, the scheduling problem can be divided into two major section, one focusing on the behavior of the task that is being scheduled and the other on the attributes that these tasks must meet in order to be scheduled. We have encoded these constraints in first-order logic form for use as input into SMT solver, and have let SMT solver find an answer. Several experiments were conducted to investigate the performance of the proposed method and the factors affecting performance were examined. We observed that the time spent and the memory used for the schedulability analysis increased in proportion to the number of tasks.

The proposed method in this paper is one of several methods for schedulability analysis that uses the constraint solving approach. However, even if the same method is used, it will be possible to express task behavior in more detailed ways with more sophisticated formulas and expressions. In this paper, we focused more on the overall method itself than on comparing the various ways to express the associated constraints. In terms of future work, our plan is to extend and optimize the constraints on task behavior, so that we can 1) reduce the size of the state space, 2) increase performance to a sufficient level and 3) support tasks under multicore system environments.

REFERENCES

[1] R. Mall, *Real-Time Systems: Theory and Practice*. London, U.K.: Pearson, 2009.
[2] J.-Y. Choi, I. Lee, and H.-L. Xie, "The specification and schedulability analysis of real-time systems using ACSR," in *Proc. 16th IEEE Real-Time Syst. Symp.*, Dec. 1995, pp. 266–275.
[3] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Syst.*, vol. 8, nos. 2–3, pp. 173–198, 1995.
[4] 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.
[5] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *Proc. 12th Real-Time Syst. Symp.*, Dec. 1991, pp. 129–139.
[6] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogramm.*, vol. 40, nos. 2–3, pp. 117–134, 1994.
[7] J. C. P. Gutiérrez, J. J. G. García, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proc. 9th Euromicro Workshop Real-Time Syst.*, Jun. 1997, pp. 136–143.
[8] J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 26–37.
[9] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 706–735, 2004.
[10] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.
[11] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky, "Symbolic schedulability analysis of real-time systems," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 409–418.
[12] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y. S. Kim, I. Lee, and H.-L. Xie, "A process algebraic approach to the schedulability analysis of real-time systems," *Real-Time Syst.*, vol. 15, no. 3, pp. 189–219, 1998.
[13] A. N. Fredette and R. Cleaveland, "RTSL: A language for real-time schedulability analysis," in *Proc. Real-Time Syst. Symp.*, Dec. 1993, pp. 274–283.
[14] S.-J. Kim and J.-Y. Choi, "Formal modeling for a real-time scheduler and schedulability analysis," in *Proc. Int. Conf. Parallel Comput. Technol.* Berlin, Germany: Springer, 2003, pp. 253–258.
[15] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking," in *Proc. IFIP Int. Workshop Softw. Technol. Embedded Ubiquitous Syst.* Berlin, Germany: Springer, 2007, pp. 263–272.
[16] A. de Matos Pedro, D. Pereira, L. M. Pinho, and J. S. Pinto, "Logic-based schedulability analysis for compositional hard real-time embedded systems," *ACM SIGBED Rev.*, vol. 12, no. 1, pp. 56–64, 2015.
[17] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using SMT solver," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May/Jun. 2016, pp. 189–194.
[18] E. Ábrahám and G. Kremer, "Satisfiability checking: Theory and applications," in *Proc. Int. Conf. Softw. Eng. Formal Methods*. Cham, Switzerland: Springer, 2016, pp. 9–23.
[19] I. Lee, P. Brémond-Grégoire, and R. Gerber, "A process algebraic approach to the specification and analysis of resource-bound real-time systems," *Proc. IEEE*, vol. 82, no. 1, pp. 158–171, Jan. 1994.
[20] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyperperiod in periodic task systems," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1813–1822, Sep. 2013.
[21] J. Z. Lavi and J. Kudish, *Systems Modeling and Requirements Specification Using ECSAM: An Analysis Method for Embedded and Computer Based Systems*. New York, NY, USA: Dorset House Publishing Co., 2004.
[22] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer, 2008, pp. 337–340.
[23] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *Proc. 26th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2005, p. 9.
[24] T. P. Baker, "Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time," Dept. Comput. Sci., Florida State Univ., Tallahassee, FL, USA, Tech. Rep. TR-050601, 2005.
[25] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, 2011, Art. no. 35.
[26] J. M. López, M. García, J. L. Diaz, and D. F. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Proc. Euromicro-RTS*, Jun. 2000, pp. 25–33.
[27] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," in *Proc. IEEE 31st Real-Time Syst. Symp. (RTSS)*, Dec. 2010, pp. 14–24.
[28] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM J. Comput.*, vol. 7, no. 1, pp. 1–17, 1978.
[29] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *Proc. 21st Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2009, pp. 249–258.

HYUK LEE received the B.S. degree in information technology from the University of Technology Sydney, Sydney, NSW, Australia, in 2006, and the M.S. degree in computer science and engineering from Korea University, Seoul, South Korea, in 2009, where he is currently pursuing the Ph.D. degree. His current research interests are in formal methods, constraint solving, and secure software engineering.

JIN-YOUNG CHOI received the M.S. degree from Drexel University, Philadelphia, PA, USA, in 1986, and the Ph.D. degree from the University of Pennsylvania, Philadelphia, PA, USA, in 1993. He is currently a Professor with the Graduate School of Information Security, Korea University, Seoul, South Korea. His current research interests are in real-time computing, formal methods, programming languages, process algebras, security, and secure software engineering.