Extensible Energy Planning Framework for Preemptive Tasks

Jin Hyun Kim
University of Pennsylvania, jinhyun@seas.upenn.edu

Deepak Gangadharan
University of Pennsylvania, deepakg@seas.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Axel Legay

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Extensible Energy Planning Framework for Preemptive Tasks

Abstract

Cyber-physical systems (CSPs) are demanding energy-efficient design not only of hardware (HW), but also of software (SW). Dynamic Voltage and and Frequency Scaling (DVFS) and Dynamic Power Manage (DPM) are most popular techniques to improve the energy efficiency. However, contemporary complicated HW and SW designs requires more elaborate and sophisticated energy management and efficiency evaluation techniques. This paper is concerned about energy supply planning for real-time scheduling systems (units) of which tasks need to meet deadlines. This paper presents a modelbased compositional energy planning technique that computes a minimal ratio of processor frequency that preserves schedulability of independent and preemptive tasks. The minimal ratio of processor frequency can be used to plan the energy supply of real-time components. Our model-based technique is extensible by refining our model with additional features so that energy management techniques and their energy efficiency can be evaluated by model checking techniques. We exploit the compositional framework for hierarchical scheduling systems and provide a new resource model for the frequency computation. As results, our use-case for avionics software components shows that our new method outperforms the classical real-time calculus (RTC) method, requiring 36.21% less frequency ratio on average for scheduling units under RM than the RTC method.

Disciplines

Computer Engineering | Computer Sciences

Comments

20th International Symposium on Real-Time Computing (ISORC 2017), Toronto, Canada, May 16-18, 2017

Extensible Energy Planning Framework for Preemptive Tasks

Jin Hyun Kim, Deepak Gangadharan,Oleg Sokolosky,
University of Pennsylvania, USA
{jinhyun,deepakg}@seas.upenn.edu,
sokolosky@cis.upenn.edu

Axel Legay
INRIA/IRISA, FRANCE
axel.legay@inria.fr

Insup Lee
University of Pennsylvania, USA
lee@cis.upenn.edu

Abstract—**Cyber-physical systems (CSPs) are demanding energy-efficient design not only of hardware (HW), but also of software (SW). Dynamic Voltage and and Frequency Scaling (DVFS) and Dynamic Power Manage (DPM) are most popular techniques to improve the energy efficiency. However, contemporary complicated HW and SW designs requires more elaborate and sophisticated energy management and efficiency evaluation techniques. This paper is concerned about energy supply planning for real-time scheduling systems (units) of which tasks need to meet deadlines. This paper presents a model-based compositional energy planning technique that computes a minimal ratio of processor frequency that preserves schedulability of independent and preemptive tasks. The minimal ratio of processor frequency can be used to plan the energy supply of real-time components. Our model-based technique is extensible by refining our model with additional features so that energy management techniques and their energy efficiency can be evaluated by model checking techniques. We exploit the compositional framework for hierarchical scheduling systems and provide a new resource model for the frequency computation. As results, our use-case for avionics software components shows that our new method outperforms the classical real-time calculus (RTC) method, requiring 36.21% less frequency ratio on average for scheduling units under RM than the RTC method.**

I. Introduction

As many CPS devices are minimized and compact, the energy efficient design of CPS gains more importance. For last decade, various energy-aware scheduling algorithms utilizing Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) have been proposed [2]. Energy-aware scheduling algorithms dynamically switch processor's voltage and frequency to minimize the energy consumption according to tasks' timing requirements, but throttling the voltage and frequency of processors reduces processor's performance, resulting in delay of SW execution time and missed deadline of real-time applications.

Moreover, it is significant to set up appropriate energy supply plan for CPS real-time components that can maximize the energy utilization. In particular, automotive software (SW) components are being developed in a compositional way such that heterogeneous SW components using different operating environments are composed into a high-performance host operating environment. In such a case, it is significant to statically set up the energy supply plan, i.e., the baseline of energy supply for individual components, prior to running the guest real-time components using energy-aware scheduling algorithms, so that no running component suffers the lack of energy that leads tasks to miss the deadline.

CMOS (Complementary Metal Oxide Semiconductor) circuit used to operate at a supply voltage level much higher than the threshold voltage, making dynamic power dissipation dominant w.r.t. static power dissipation. As a result, DVFS approaches, such as [1], [24], [25], are suitable for reducing dynamic power dissipation. Contemporary nanometer scale chips lower the supply voltage by shrinking the transistor size and reduce dynamic power consumption. This leads to a significant increase in the leakage consumption, because the smaller gap, the higher the subthreshold dissipation. As a result, the static power dissipation has become as important as dynamic power consumption, and DPM approaches that target reduction of the leakage power have recently gained more popularity [2].

Mosse et al., [14] provided the first five computation techniques to determine a minimal ratio of frequency of processors that preserves schedulability of real-time tasks. However, those techniques should limit all deadlines of tasks to a common deadline. Bini et al., [4] presented sufficient conditions to compute the minimum processor speed that satisfies EDF [17] and RM tasks[15], [16]. However, they are limited to trivial timing parameters of real-time components under EDF and RM, thus they have the limitation in analyzing energy aspects of emerging complicated real-time systems.

Energy efficiency evaluation has been more complicated as the scheduling systems are becoming complicated, such as OS hypervisor. Tchamgoue et al., [21] provides schedulability conditions aware of energy consumption for the compositional framework for OS hypervisor technology. The design trend of CPS is leading to a need of more extensible and flexible framework for planning energy consumption and evaluation energy efficiency for real-time systems.

a) Problem Description: The problem that we address is how to compute a minimal ratio of frequency of processors requiring a minimal energy supply that schedule preemptive tasks under RM and EDF scheduling algorithms. A component is a scheduling unit, where each component can deploy various scheduling mechanisms, EDF and RM, and set to a specific static frequency that schedules tasks of the component. In terms of CPU frequency, the component is compositional in that the frequencies of each component can be composed into a representative frequency that satisfies the composed components. In short, we statically compute a minimal ratio of frequency of each component in a compositional way, which leads to all tasks being schedulable.

b) Approach and Contributions: This paper presents a model-based technique to facilitate the selection of a minimal ratio of frequency of processors for real-time components under RM and EDF. Our technique is based on the schedulability theory of the compositional framework. The minimal frequency can be used to estimate the energy consumption of real-time components, and set up an energy supply plan for real-time components not to violate timing requirements. This paper exploits the concept of interface in the compositional framework [19]. An interface of real-time components in this paper consists of timing requirements of a component and a fraction of CPU's cycles that executes tasks of a real-time component.

An interface is a contract between a resource demander and a resource supplier. Thus, it can be used as collective timing requirements of resource-demanding component and as a resource supply pattern of the supplier. A resource supply pattern of suppliers is called resource model. For a given resource, the schedulabilty of a real-time component can be verified by comparing a resource demand pattern of a real-time component and a resource supply pattern of the resource supplier. That is, if a resource model satisfies an interface of a real-time component, the component is schedulable. A resource model can easily be represented as a simple formula, but the demand pattern of a workload depending on scheduling algorithms is not easy to be modeled as a simple formula. In addition, this paper leverages a model-based analysis technique to achieve extensibility and flexibility of our framework for energy planning and efficiency evaluation. A variety of software features required by customers exponentially increases the complexity of CPS systems beyond the capability of classical analysis techniques. This paper adopts the model-based techniques of [5] to build a resource model and its demand pattern of a given component and presents a way to compute a minimal ratio of frequency of a processor.

For the energy-aware design of CPS, this paper contributes to:

- A new formal technique to derive a minimal ratio of frequency of processors satisfying a minimal cycle demand of a component consisting of preemptive tasks,
- An analysis framework based on model checking techniques.

The rest of the paper is organized as follows: Section II discusses the related work. Section III presents the background theory of this work. Section IV presents a new resource model and schedulability conditions to compute a minimal frequency ratio. In addition, we present a Real-Time Calculus-based technique to compute a minimal frequency ratio so that frequency ratios computed by a new technique and the RTC-based techniques can be compared to show individual pessimism. In Section V, we provide a model-based technique, based on the schedulability conditions based on the resource model in Section IV. Section VI shows comparison results of minimal frequency ratios computed by our mode-based and RTC-based techniques and provide our observations. In Section VII, we conclude this paper with the potential future work.

II. Related Work

For energy-aware scheduling techniques, early efforts, such as [1], [10], [23], [24], [25], leveraged the performance slack available in real-time applications such that reducing the performance of a processor does not cause any applications to miss their deadlines. They have developed scheduling algorithms, exploiting the property of real-time tasks that the actual task execution times are much less than the assumed worst-case execution time, thus the remaining time to each deadline is a guaranteed slack time where no higher priority task would intervene. However, they are not extensible enough for contemporary scheduling systems to instantly deal with their complexity due to variety of real-time features from costumers.

Lie et al., in [12], one of the researches comparable with this work, presents a processor frequency selection technique which is based on Real-Time Calculus (RTC). In this research, a series of processing elements process inputs streams and are represented by lower and upper bounds of their services. The main contribution of this work is a computation of the frequency range that needs to be supported by each processing element in order to realize their service bounds. This paper compares minimal frequencies computed by our model-based technique with those computed by a RTC-based technique in Section VI.

To compute a minimal ratio of frequency, we construct our model based on our previous work [6]. The model of [6] is extended with multi-core scheduling algorithms, and various overhead features including interrupt handling time, scheduling time and context switching time. However, in this paper, we do not include the overhead parameters in our analysis to fairly compare our model-based techniques with RTC techniques. Furthermore, since we focus on SW-related and energy-relevant features of the system, our model does not incorporate HW aspects, such as pipelines, caches etc, in this paper.

III. Preliminaries

A. Power Model

Based on CMOS technology, the power consumption is dominated by dynamic power dissipation P_d which is given by $P_d = \alpha C_{ef} V_{dd}^2 f + \alpha V_{dd} I_{short} + V_{dd} I_{leak}$, where C_{ef} is the effective switched capacitance, α is the gate activity factor (i.e. the probability of gate switching), V_{dd} is the supply voltage, and f is a clock frequency, I_{short} is the current between the supply voltage and ground during gate switching, and I_{leak} is the leakage current, which is independent of the actual frequency and system activity [2].

The processor speed is almost linear with respect to the voltage: $f = k \cdot \frac{(V_{dd}-V_{th})^2}{V_{dd}}$ where k is a constant and V_{th} is the threshold voltage. Thus, the power P_d is almost cubically related to the frequency $f : P_d \approx C_{ef} \cdot \frac{f^3}{k^2}$ The computation time e_i is characterized by $\frac{C}{f}$, where C is the number of cycles to complete the execution of a task i. Then, the energy consumption E of the task is $E = P_d \cdot e_i \approx C \cdot C_{ef} \cdot \frac{f^2}{k^2}$ Both the frequency and the supply voltage can be reduced together. As a result, the processor power can be cubically reduced and the energy expense can be quadratically saved.

In this paper, we utilize only the frequency as energy efficiency evaluation parameter from the above power model, assuming that the other parameters are constant.

B. Setting of the System

A scheduling system in this paper is defined as follows: A scheduling unit is defined by $C = (W, A)$, where a workload W is a set of tasks $\{T_1, T_2, ..., T_i\}$, A is a scheduling algorithm. A preemptive periodic task is characterized by $T_i = (p_i, e_i)$, where i is a task id, p_i is a period, and e_i is the worst-case execution time. The deadline of tasks is implicit throughout this paper, thus the period and the deadline of a task are the same unless it is specified explicitly as if $T_i = (p_i, e_i, d_i)$. For generality, the unit of task's real-time parameters is a cycle, not a time. The energy consumption of components is represented by a frequency of processors, a fraction of CPU cycles, that satisfies the needs of individual tasks in a component. The frequency of each component is captured by an interface, which would be defined at Section IV-B. The problem is how to compute a static minimal frequency ratio for a given task set (workload) under a specific scheduling algorithm.

C. Compositional Framework

The compositional framework in this paper is a schedulability analysis framework for hierarchical scheduling systems (HSS), where each component of real-time tasks is analyzed in a compositional way. In this framework, a resource requirement of real-time tasks of components are introduced by an interface, and a supply pattern of resources available to tasks is represented by a resource model. The schedulability of tasks is done by checking if an interface of a component is satisfied by a resource model, i.e. the quantity of resources demanded by tasks is always provided by a resource supply model.

For any time interval $[0, t]$, the amount of resources demanded by a workload W, a set of real-time tasks, is modeled by a demand bound function (dbf). Given a task set T_i, the dbf under the scheduling policies EDF (Earliest Deadline First) [18] is defined by:

$$\mathsf{dbf}_{EDF}(W, t) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor \cdot e_i \tag{1}$$

In the similar way, the dbf under the scheduling policies RM (Rate Monotonic) [18] is defined by:

$$\mathsf{dbf}_{RM}(W, t, i) = e_i + \sum_{T_k \in HP(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \tag{2}$$

where $HP(i)$ is a set of tasks whose priorities are higher than that of T_i.

For any time interval $[0, t]$ and any resource model Γ, the amount of resources supplied by a resource model is mathematically modeled by a supply bound function (sbf). For a given component C and a resource model Γ, the schedulability condition is generally defined by:

$$\forall t_{\mathbb{N}_{\geq 0}} \ \mathtt{dbf}_A(W, t) \leq \mathtt{sbf}_\Gamma(t) \tag{3}$$

where A is a scheduling algorithm of the component C.

An interface I representing a quantity of resources demanded collectively by tasks of C is mathematically computed by the above inequality between dbf and sbf [19]. In Section IV-B, we present a new supply bound function that is used to check the schedulability and to compute an interface that satisfies a given component.

D. Timed Automata and Model Checking

The formalism that we use for modeling scheduling systems is Timed Automata (TA), particularly, Stopwatch Automata (SWA)[7] which are a subclass of linear hybrid automata. Basically, a stopwatch automaton is a regular timed automaton augmented with stopwatches. A stopwatch is a continuous variable (clock) that can stop and resume without necessarily performing a reset. This is done by assigning to such clocks a progress rate of 0 or 1, i.e., the derivative of the stopwatch is assigned to a constant or an expression which evaluates to 0 or 1, so that the stopwatch progresses with the same rate as logical time.

For requirement specifications, we use a subset of Computational Tree Logic. The grammar of this subset is

$$\varphi ::= A\Box P \mid A\Diamond P \mid E\Box P \mid E\Diamond P$$

where A and E are paths operators, meaning respectively "for all the paths" and "there exists a path". $\Box$and $\Diamond$are state operators, meaning respectively "all the states of the path" and "there exists a state in the path". P is an atomic proposition that is valid in some state. For example the formula "A$\Box$ not deadlock" specifies that in all the paths and all the states on these paths we will never reach a deadlock state in which the system is permanently blocked.

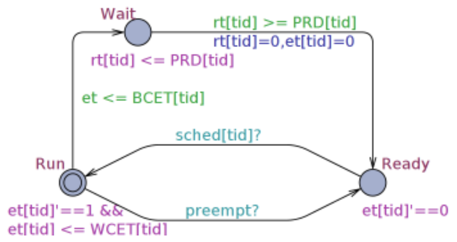


Fig. 1. Conceptual task in SWA

A conceptual task modeled by SWA is shown in Fig. 1, where a task tid executes between BCET[tid] and WCET[tid] time units every PRD[tid] time units (The initial location in the model is Run with double-circle by UPPAAL syntax).

The execution time of tasks, the CPU-using time, is denoted by et[tid], and its running time, the time that has elapsed since its new period began, is denoted by rt[tid]. The task joins the location Run when it synchronizes with the event channel sched[tid]!, then the clock et[tid] makes progress. However, if the task moves into the location Ready when it synchronizes the event channel preempt!, then the clock et[tid] stops progress by the invariant et[tid]'==0. In this way, the preemption mechanism of scheduling systems is implemented.

UPPAAL [3] is a model checking tool suite which we apply to our analysis. UPPAAL MC is an exhaustive analysis technique that explores every state and transition to see if the system satisfies properties. Meanwhile, UPPAAL SMC [8] is a simulation-based technique that runs a given tasks numerous times for a given simulation time, generates traces, and computes out of the traces a probability that the system satisfies properties with a specific certainty. In this paper, UPPAAL MC is the primary analysis technique for the computation of a minimal ratio of frequency and UPPAAL SMC is used to simulate a given task set to validate the results.

In order to model the preemption between tasks, we use SWA. However, it has the limitation in obtaining analysis accuracy in a sense that UPPAAL may return a false positive answer. Hence, even though the system in SWA models satisfies a given property, UPPAAL may say that "property may not be satisfied."

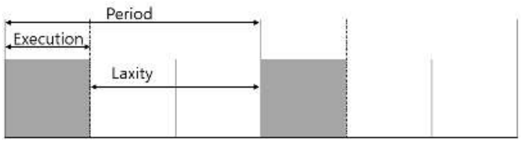
IV. COMPOSITIONAL FRAMEWORK FOR ENERGY-AWARENESS

This section describes the underlying idea that is the basis of our analysis framework for the computation of a minimal frequency ratio of processors.

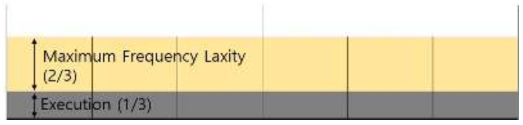
A. Selection of A Minimal Frequency

The frequency of processors refers to clock cycles per second. Basically, we view the frequency of a processor as a periodic resource model that provides tasks of components with a specific quantity of resources, e.g., execution cycles per period. The reason why the frequency can be viewed as a periodic resource is as follows: In principle, the execution of a periodic task has a laxity interval where the task can delay the execution as long as it does not miss the deadline, as shown in Fig. 2(a). Namely, the execution of a periodic task can be delayed the same as the laxity interval of the task. The laxity interval of a task can be interpreted by a frequency laxity which can throttle the processor frequency ratio, and such a lowered frequency ratio inflates the execution of a task as much as the laxity interval of a task, as shown in Fig. 2(b).

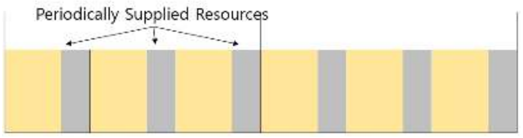
The compositional framework [19] allows computing the amount of resources that a component requires every specific time unit. The component may have one or more tasks running under specific scheduling policy. That is, the compositional framework computes an interface, a collective resource requirement of a given component, by checking whether the resource supply of a resource model for the component satisfies the schedulability of all tasks in the component. For



(a) An execution of $T_1 = (3, 1)$



(b) Inflated execution of T_1 by the frequency ratio $\frac{1}{3}$



(c) Regularly split execution of T_1 by a periodic resource model PRM=(1,0.33)

Fig. 2. Job executions

instance, Fig. 2(c) shows that T_1 is satisfied by a periodic resource model, $PRM = (1, 0.333)$. The execution time of the resource model, 0.333, can also be used as a frequency ratio for running T_1 that satisfies its deadline. Similarly with the compositional framework, the laxity interval of a periodic task in Fig. 2(a) can be split, where the split laxity and the execution of the task can interleave, as shown Fig. 2(c). That is, the execution of tasks can be regularly divided by uniformly split laxity periods, as shown Fig.r̃effig:PeriodicExecution. For instance, the execution of $T_1 = (3, 1)$ can be split by $(1, 0.333)$ which executes for 0.333 time units every time unit. In other words, the execution of the task can be delayed at least for 0.666 every time unit. The execution time 0.333 of the task is exactly the same as the number of necessary cycles for the task.

The previous work, such as [18], [19], [20], discuss various ways to find an optimal resource model for scheduling units. In this paper, we apply the compositional framework to figure out a minimal frequency ratio of processors by providing a new resource model from the frequency perspective.

B. Supply Bound Function of Processor Frequency

To compute a minimal frequency ratio of processors for a given component, the compositional framework requires a resource model that simulates the worst-case supply of CPU cycles for a given frequency ratio. Using the resource model, it can be checked if all tasks of a given component are scheduled by the CPU cycles per period supplied by the resource model. A minimal frequency ratio can be found by identifying the smallest frequency ratio that satisfies the schedulability of components.

In the compositional framework, the interface of a component captures resources needed by the component. The interface should be compatible with a resource model in that the resource supply pattern of a resource model satisfies the resource requirement of an interface. In the following, we thus introduce a new resource model that can be used to compute a minimal frequency ratio of processor that leads to schedule every task of a component.

First, we define a resource model, called Periodic Laxity Resource Model (PLRM), that has the worst resource supply pattern for a given frequency ratio. Periodic Laxity Resource Model (PLRM) is a periodic resource model for processor frequency ratio, and it is denoted by $\Gamma_{PLRM} = (\Pi, \Theta, \Lambda)$ where Π is a period, Θ is a number of resources, and $\Lambda = \Pi - \Theta$ is a laxity, where $\Lambda > 0$. PLRM $\Gamma = (\Pi, \Theta, \Lambda)$ provides the amount Θ of CPU cycles every Π time units under Λ time unit of laxity.

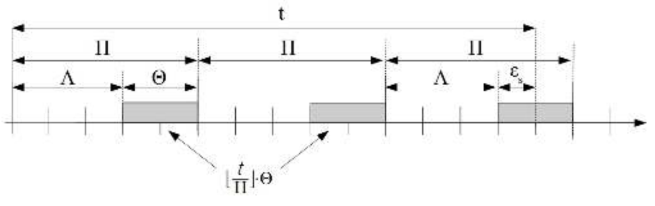


Fig. 3. The worst-case resource supply of PLRM

Similary with PRM resource model of the compositional framework, as shown in Fig. 2, the PLRM has the worst-case resource supply pattern shown in Fig. 3. In this worst-case, the resource supply Θ never starts before the laxity Λ elapses. The worst-case resource supply happens when a new resource demand occurs as soon as a new period begins.

For a given PLRM $\Gamma = (\Pi, \Theta, \Lambda)$, the supply bound function $\mathsf{sbf}_{\Gamma_{\text{PLRM}}}(t)$ is defined by:

$$\mathsf{sbf}_{\Gamma_{\text{PLRM}}}(t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta + \epsilon_s \tag{4}$$

$$\epsilon_s = \max\left(t - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi - \Lambda, 0\right) \tag{5}$$

where $\Lambda = \Pi - \Theta$. For the conservative analysis, PLRM serves the worst-case resource supply, where the least amount of resource is computed depending on a PLRM Γ for any given time interval t.

Lemma 1: A linear lsbf lower-bounds $\mathsf{sbf}_{\Gamma_{\text{PLRM}}}$ and it is defined by:

$$\mathsf{lsbf}_{\Gamma_{\text{PLRM}}} = \frac{(t-\Lambda)}{\Pi} \cdot \Theta \leq \mathsf{sbf}_{\Gamma_{\text{PLRM}}}$$

Proof: We consider two cases: 1) $\epsilon_s = 0$ and 2) $\epsilon_s > 0$.

Case 1) $\epsilon_s = 0$. In this case,

$$t - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi - \Lambda \leq 0 \tag{6}$$

From Eq. 6, we have

$$\frac{(t-\Lambda)}{\Pi} \cdot \Theta \leq \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta \tag{7}$$

Then,

$$\mathsf{sbf}_{\Gamma_{\text{PLRM}}} - \mathsf{lsbf}_{\Gamma_{\text{PLRM}}} = \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta - \frac{(t-\Lambda)}{\Pi} \cdot \Theta \geq 0 \tag{8}$$

by Eq. 7

Case 2) $\epsilon_s > 0$. In this case,

$$t - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi - \Lambda > 0 \tag{9}$$

From Eq. 9, we have

$$\frac{t-\Lambda}{\Pi} > \left\lfloor \frac{t}{\Pi} \right\rfloor \tag{10}$$

Then,

$$\begin{aligned}
&\mathsf{sbf}_{\Gamma_{\text{PLRM}}} - \mathsf{lsbf}_{\Gamma_{\text{PLRM}}} \\
&= \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta + t - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi - \Lambda - \frac{(t-\Lambda)}{\Pi} \cdot \Theta \\
&= \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi + t - \Lambda - \frac{(t-\Lambda)}{\Pi} \cdot \Theta \\
&= \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot (\Theta - \Pi) + (t-\Lambda) \cdot \left(1 - \frac{\Theta}{\Pi}\right) \\
&= \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot (\Theta - \Pi) + (t-\Lambda) \cdot \left(\frac{\Pi - \Theta}{\Pi}\right) \\
&= -\left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Lambda + (t-\Lambda) \cdot \frac{\Lambda}{\Pi} \\
&= \Lambda \cdot \left(\frac{t-\Lambda}{\Pi} - \left\lfloor \frac{t}{\Pi} \right\rfloor\right) > 0
\end{aligned} \tag{11}$$

by Eq. 10 and $\Lambda > 0$ from its definition. ■

For a given PLRM $\Gamma = (\Pi, \Theta, \Lambda)$, the schedulability conditions for EDF and RM [9] are as follows:

Theorem 1: A component $C = \langle W = \{T_1 = (p_1, e_1, d_1), ..., T_n = (p_n, e_n, d_n)\}, EDF\rangle$ is schedulable with respect to the PLRM resource model Γ *iff*

$$\forall 0 < t \leq LCM_W + \max_{1 \leq i \leq n} d_i \quad \mathsf{dbf}_{EDF}(W, t) \leq \mathsf{sbf}_{\Gamma_{\text{PLRM}}}(t) \tag{12}$$

where t is a time interval, and LCM_W is the least common multiple of the periods of all the tasks $T_i \in W$.

Theorem 2: A component $C = \langle W = \{T_1 = (p_1, e_1, d_1), ..., T_n = (p_n, e_n, d_n)\}, RM\rangle$ is schedulable with respect to the PLRM resource model Γ *iff*

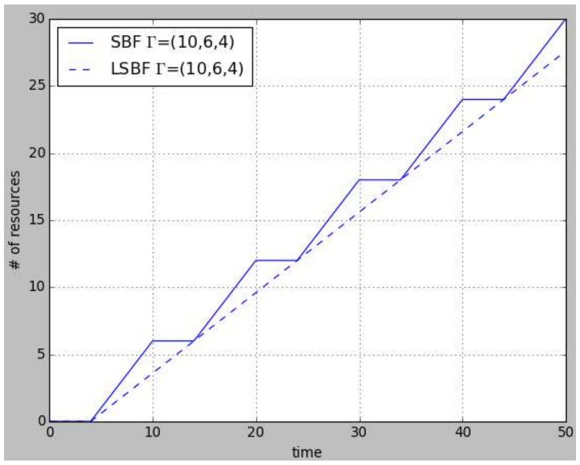
$$\forall T_i, \exists t_i \in [0, d_i] \quad s.t \quad \mathsf{dbf}_{RM}(W, t_i, i) \leq \mathsf{sbf}_{\Gamma_{\text{PLRM}}}(t) \tag{13}$$

A minimal frequency ratio for a component can be computed by finding a PLRM satisfying the set of tasks in the component.

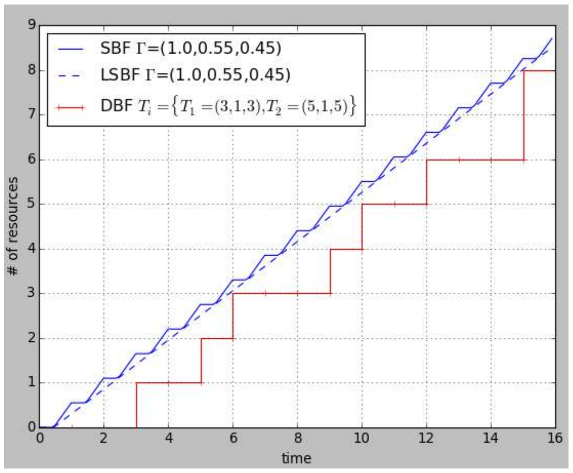
Example 1: As shown in Fig. 4(b), the set of tasks $T = \{T_1 = (3, 1), T_2 = (5, 1)\}$ under EDF is scheduled by the PLRM $\Gamma_{\text{PLRM}} = (1.0, 0.55, 0.45)$. Conclusively, the satisfying frequency ratio is 0.55. If the frequency of a given processor is 1 MHz, a minimal frequency ratio satisfying the task set is 550 kHz.

C. Real-Time Calculus based Approach

This paper compares our technique with a Real-Time Calculus-based technique. Real-Time Calculus (RTC) [22] is a well known compositional analysis technique for embedded systems that has its roots in Network Calculus [11]. In RTC, the workload of a task T_i is captured in terms of the arrival curves $[\overline{\alpha}_i^u(\Delta), \overline{\alpha}_i^l(\Delta)]$. These represent the upper $(\overline{\alpha}_i^u(\Delta))$ and lower bound $(\overline{\alpha}_i^l(\Delta))$ of the workload requirement (in



(a) The sbf and lsbf for $\Gamma_{\text{PLRM}} = (10, 6, 4)$



(b) The sbf and lsbf for $\Gamma_{\text{PLRM}} = (1.0, 0.55, 0.45)$ and the resources demand of a task set $T = \{T_1 = (3, 1), T_2 = (5, 1)\}$

Fig. 4. The sbf and lsbf of PLRM

terms of processor cycles) for the task in any time interval Δ. Similarly, service curves $[\overline{\beta}_i^u(\Delta), \overline{\beta}_i^l(\Delta)]$ are used in RTC to capture the upper and lower bound on the service provided to task T_i.

In order to compute a minimal frequency ratio, we only need one quantity for each task in the workload set namely $\overline{\alpha}_i^u(\Delta)$. This can be used to compute the lower bound on the service requirement ($\overline{\beta}_i^l(\Delta)$) for the task. The lower bound of the frequency ratio that ensures schedulability can be found using $\overline{\beta}_i^l(\Delta)$ for each task T_i. This is described in detail below.

For an implicit deadline periodic task T_i with period p_i and worst-case execution time (WCET) e_i, the upper bound on arrival curve $\overline{\alpha}_i^u(\Delta)$ can be found as follows.

$$\overline{\alpha}_i^u(\Delta) = \left\lceil \frac{\Delta}{p_i} \right\rceil \times e_i \tag{14}$$

In order for the task to be schedulable, i.e., for execution of each job of the task to finish before the deadline d_i, the service required in case of EDF scheduling is given by $\overline{\beta}_i^l(\Delta) \geq \overline{\alpha}_i^u(\Delta - d_i)$. Hence, the lower bound on the service required for schedulability of a single task is $\overline{\beta}_i^l(\Delta) = \overline{\alpha}_i^u(\Delta - d_i)$. The lower bound on the service required for schedulability of all the tasks is given by $\overline{\beta}^l(\Delta) = \sum_{\forall i} \overline{\alpha}_i^u(\Delta - d_i)$. The lower bound on frequency ratio can then be computed for EDF as $\max_{\forall \Delta} \left(\frac{\overline{\beta}^l(\Delta)}{\Delta} \right)$.

Similarly, the lower bound on the service required for task T_i in case of RM scheduling is given by $\overline{\beta}_i^l(\Delta) = \sum_{\forall k \in hp(i)} \overline{\alpha}_k^u(\Delta) + \overline{\alpha}_i^u(\Delta)$, where $hp(i)$ is the set of tasks with higher priority than task T_i. After finding $\overline{\beta}_i^l(\Delta)$ for all the tasks, the lower bound on frequency ratio can be computed for RM as $\max_{\forall i \forall \Delta} \left(\frac{(\overline{\beta}_i^l(\Delta))}{\Delta} \right)$.

V. Model-based Compositional Framework

In the previous sections, we explained how to compute a minimal processor frequency ratio by checking schedulability of a component, a scheduling unit, against the resource model, PLRM. However, this framework is limited to trivial settings of components, so real-time tasks are generally characterized by a period, an execution time, and a deadline under EDF and RM. To make it more extensible, i.e., adaptable to more complicated scheduling mechanisms, we capture scheduling units and resource models as formal models and analyze schedulablity of scheduling units using model checking techniques. This section presents a formal component model (C_{SWA}) corresponding a scheduling unit under EDF and RM of the compositional framework, and a formal resource model of PLRM (PLRM_{SWA}). So that a minimal frequency ratio is identified by checking the component model against the resource model. In order to analyze the pessimism of PLRM resource model, we compare minimal frequencies computed by using our formal models with the ones computed by a similar RTC-based technique.

In this paper, the features of the scheduling unit in our model-based approach are limited to the features of the compositional framework so as to fairly compare the gap between our model-based technique and RTC-based technique. For this reason, our formal model does not incorporate any complicated feature of scheduling units in this paper.

The model-based compositional framework to check the schedulability and compute a minimal frequency ratio of processors is shown in Fig. 5, where the $\text{sbf}_{\Gamma_{\text{PLRM}}}$ and dbf_A for

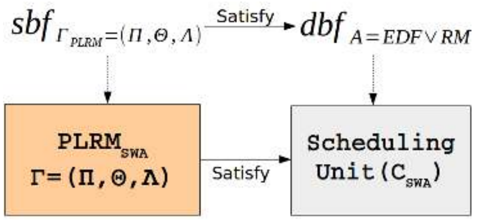
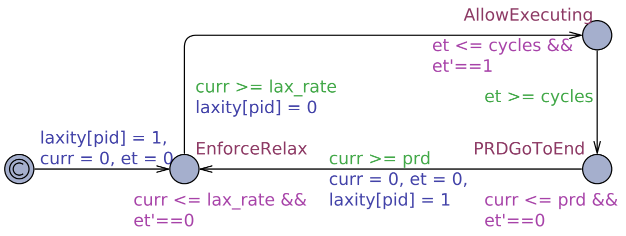
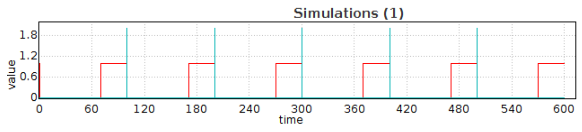

$sbf_{\Gamma_{PLRM}=(\Pi,\Theta,\Lambda)}$
Satisfy
$dbf_{A=EDF \vee RM}$
PLRM_{SWA}
$\Gamma=(\Pi,\Theta,\Lambda)$
Satisfy
Scheduling Unit (C_{SWA})

Fig. 5. Our model-based framework

RM and EDF are modeled, respectively, by a PLRM resource model and a scheduling unit model using SWA. Fig. 6(a) shows a formal model of PLRM resource model in SWA. The PLRM resource model controls the execution of a job by a bool variable laxity[pid], which represents the enforcement of laxity over a running job; a running job stops if laxity[pid] is 1, and otherwise, the running job keep executing. When laxity[pid] = 1, the current job stops for a given laxity laxity_rate time units (Λ). After the laxity time, the PLRM resource model reaches the location AllowExecuting, and lets a running job to execute for cycles time units (Θ). Then, it joins the location PRDGoToEnd and does nothing until the end of the current period. Fig. 6(b) describes the execution pattern of a PLRM for $\Gamma = (100, 30, 70)$ and shows the same execution pattern with that of the worst-case execution pattern of PLRM in Fig. 3.



(a) SWA PLRM model



(b) Simulation of PLRM ($\lambda = (100, 30, , 70)$)

Fig. 6. SWA PLRM model and simulation

Given a C_{SWA} and a $PLRM_{SWA}$, the schedulability is checked by checking the property **A[] not error**. The variable error in the temporal logic happens when a running task misses its deadline, i.e., the task model Fig. 8 stops on the location DLMissed). If the model checker cannot find out any counterexample against the property, a given task set is proved schedulable. In other words, by checking the following property:

$$(PLRM_{SWA}||C_{SWA}) \vdash \text{A[] not error}$$

Then we prove the following property:

$$\forall t_{\mathbb{N}_{\geq 0}} \; \mathtt{dbf}_A(W, t) \leq \mathtt{sbf}_\Gamma(t)$$

We use this model to check the schedulability and identify a minimal frequency ratio of a processor. Basically, this computation consists in two steps: First, any frequency ratio is set to the PLRM resource model for a given component; Second it is proved that the component is schedulable by the cycle supply of the resource model. We repeat the above steps until the minimum frequency ratio is identified.

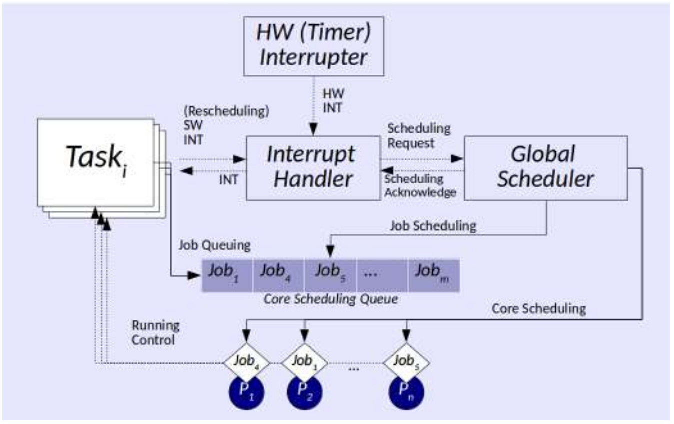


Fig. 7. Architecture of SWA models

A. SWA Scheduling Unit Model

Fig. 7 shows the structure of our SWA scheduling unit model for multi-processor, however we limit the number of core running tasks to 1 in our experiments. It is structured in a similar way with a real-time operating system. `Interrupt Handler` in Fig. 7 handles interrupts from HW/SW components and requests a scheduler to instantiate and schedule jobs. Once `Interrupt Handler` is invoked by HW Interrupter, it inserts the relevant task identity (id) to `Core Scheduling Queue` so that a job of the id is created, and then requests a scheduling for the inserted job. `Global Scheduler` schedules jobs in `Core Scheduling Queue` according to a scheduling policy and sets the highest priority job to an available processor. If the running job has a lower priority than a new job, it is switched out but resides in the ready queue. Once `Global Scheduler` finishes the scheduling, it acknowledges `Interrupt Handler` to enable interrupts. A job occupying a processor can execute as long as it is the highest priority task in `Core Scheduling Queue`. If a job of `Task` completes, it releases the processor and executes scheduling again to assign the next highest-priority job to the released processor.

B. Task Model

A task we consider in this paper is characterized by the parameters shown in Listing 1. This parameters are instantiated by the structure variable tcb[tid] where tid is a task identity (id).

The SWA `Task` model of Fig. 8 captures the behavior of a preemptive job. It starts with synchronizing the event channel dispatch_job[tid] with `Interrupt Handler`. The progress of a running job stops when it is preempted by a higher priority task, and stops when the cycle of the associated processor are not provided. The progress of a running job is captured by a stopwatch clock t_et[tid], of which progress stops when it is set to 0 and resumes when it set to 1. If a job is preempted by a higher priority task, the task model moves from the location Executing to the location Ready, then the clock t_et[tid] is set to 0 and the progress of job's execution stops. Even if a job is occupying a CPU, its progress can stops at the location

Listing 1. Task Cotrol Block

```
typedef  struct   {
   pid_a          pid;                    //  Processor ID assigned to job,
   prio_t         prio ;                  //  Priority  ,
   time_t        ioffset   ;              //  Initial    offset  ,
   time_t         prd;                    //   Period,
   time_t         dl ;                    //  Deadline,
   time_t         bcet;                   //  Best−case execution time,
   time_t         wcet;                   //  Worst−case execution time,
   bool          preemptive;              //  Preemptability
task_stat_t   stat ;                      //  Job status
} os_tcb_t ;

os_tcb_t       tcb [ tid_t  ];            //  Task Control  Block
```

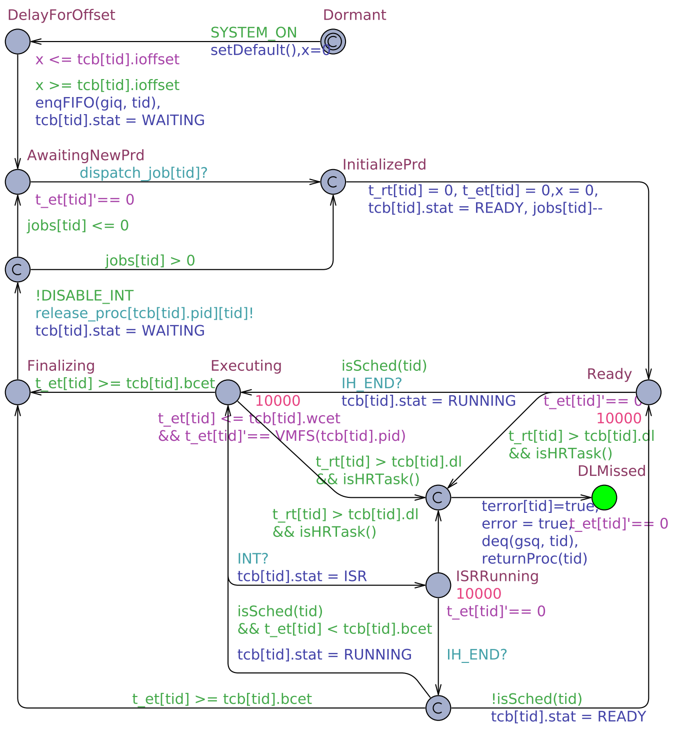


Fig. 8. SWA model of task

Executing when the stopwatch t_et[tid] is set to 0 by the function VMFS(tcb[tid].pid) which returns 0 if a cycle supplier is not running. Otherwise, the stopwatch is set to 1 and the progress of the job runs.

VI. Comparison of Minimal Frequency Ratios

Now, we compare minimal frequencies computed by the model-based technique with the ones computed by RTC-based technique.

Table I shows 3 analysis cases, where the configuration of individual experiments has the same task set but depends on a scheduling policy, EDF or RM. Checking each sample repeats by reducing the amount of cycles, the second parameter of Γ, until we identify a minimal frequency that satisfies the schedulability; In those experiments, the CPU utilization is set to 56%. As results, the frequency ratios which satisfy the set of tasks are 0.626, 0.667, and 0.57, respectively, for RM, RM with non-preemption, and EDF.

TABLE I
PLRM Interfaces and Frequency.

C_{id}	A	Tasks	PLRM	Freq. Ratio
C_1	RM	T_1(25,5)	$\Gamma(1, 0.626, 0.374)$	0.626
		T_2(45,10)		
		T_3(75,10)		
C_1	RM*	T_1(25,5)	$\Gamma(1, 0.667, 0.333)$	0.667
		T_2(45,10)		
		T_3(75,10)		
C_1	EDF	T_1(25,5)	$\Gamma(1, 0.57, 0.43)$	0.57
		T_2(45,10)		
		T_3(75,10)		

*a non-preemptive scheduling unit

TABLE II
Avionics software real-time components [13]

Ref.	Comp. ID	(P,E)	CPU Util.	New (RM)	New (EDF)	RTC (RM)	RTC (EDF)
1	Comp3	(40,1)	0.085	0.088	0.086	0.124	0.085
		(80,2)					
		(200,2)					
2	Comp4	(40,1)	0.18	0.181	0.181	0.342	0.18
		(40,2)					
		(40,4)					
		(200,1)					
3	Comp5	(200,1)	0.015	0.016	0.016	0.025	0.015
		(200,1)					
		(400,2)					
4	Comp6	(100,7)	0.085	0.086	0.086	0.139	0.085
		(400,6)					
5	Comp8	(52,6)	0.231	0.241	0.241	0.453	0.231
		(52,6)					
		-					
6	Comp9	(52,8)	0.164	0.171	0.171	0.302	0.1649
		(200,1)					
		(200,1)					
7	Comp11	(200,1)	0.007	0.008	0.008	0.01	0.007
		(1000,2)					
8	Comp12	(40,1)	0.060	0.063	0.061	0.098	0.060
		(40,1)					
		(100,1)					
9	Comp14	(100,1)	0.015	0.016	0.016	0.023	0.015
		(400,2)					
10	Comp15	(100,3)	0.040	0.041	0.041	0.065	0.040
		(200,2)					
11	Comp16	(200,1)	0.023	0.024	0.023	0.036	0.023
		(800,10)					
		(1000,5)					

A. Observations from Use-cases

We applied our method to a use-case of avionics software components [13], where 34 tasks are grouped into 17 components, and each component consists at most 4 tasks scheduled by RM or EDF. 11 components of 17 components, excluding

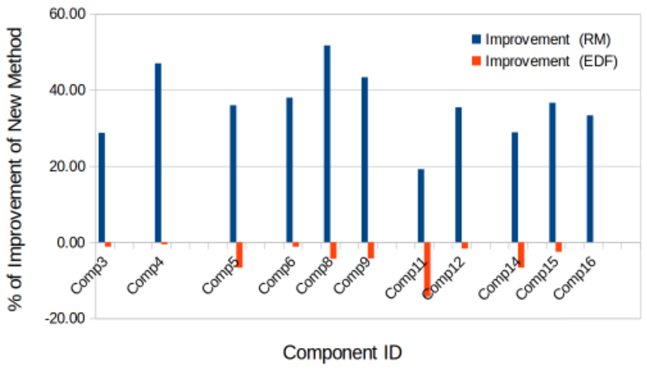
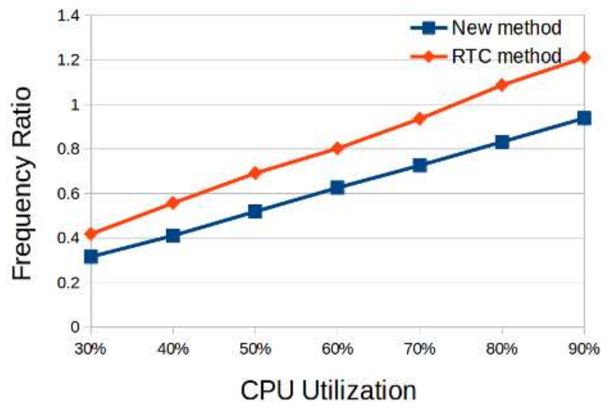
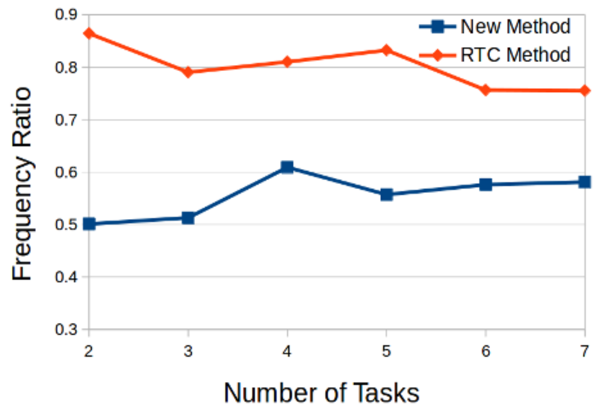


Fig. 9. Improvements of new methods against the RTC method



(a) Frequency variation by variation of CPU utiliation



(b) Frequency variation by the number of tasks

Fig. 10. Variation of minimal frequency ratios

single-task components, are analyzed by the model-based technique and RTC-based technique in Section IV-C. Fig. 9 shows that our new method outperforms the classical RTC technique, requiring the average 36.21% less frequency ratio for scheduling units under RM. However, the RTC technique outperforms the new method in the case of EDF.

Notice that the RTC-based technique returns the minimal frequencies of the task sets using EDF that are the same as the corresponding CPU utilizations, that is consistent with [4]. Meanwhile, the minimal frequencies of the same cases computed by the model-based method are bigger than the corresponding CPU utilizations. We used UPPAAL SMC to check those task sets again and could see that the task sets using EDF are scheduled with the frequency ratio that is the same as its CPU utilization too. We believe that the reason is due to the over-approximation computation by UPPAAL MC when handling stopwatch clock mechanism. As a conclusion, for the task sets using RM, our new method returns a more optimized frequency ratio than the RTC-based technique. The RTC-based technique shows that the task set using EDF is schedulable with the frequency ratio that is the same as the corresponding CPU utilization. The over-approximation analysis of UPPAAL does not always impose pessimism upon the analysis results, and it depends on complexity of a given model. For this reason, the results of RM show lower frequencies than the RTC-based method.

For the RM cases, Fig 10(a) compares the deviations of minimal frequencies from the CPU utilizations of given task sets computed by our model-based technique and the RTC-based technique. Fig. 10(b) the variation of frequencies by the number of tasks (from 2 to 7). Using the same CPU utilization (50%), the frequency ratio satisfying the the components increases by the number of tasks in our method, but the RTC-based technique is sensitive to the number of tasks.

B. Scalability of Model-Based Method

Compared with RTC-technique, the model-based technique is not scalable in that model-checking technique is often subject to the state-explosion since it explores all possible states exhaustively. Our model-based framework is also prone to the state-explosion issue.

Fig. 11 shows the variation of UPPAAL MC verification time using our models by the number of tasks. The scalability of our model-based framework is impacted by two major factors: the number of tasks and the complexity of task properties. As the number of underlying tasks increases, the verification time increases exponentially. In addition, the complexity of task properties, such as execution times and periods, impacts on the scalability. For instance, task properties in the prime numbers impacts on the verification time. Our experiment environment is as follows:

- Processor: Intel Core i7-3520M CPU @ 2.90GHz $\times 4$
- Memory: 7.5 GiB
- OS: Ubuntu 64-Bit

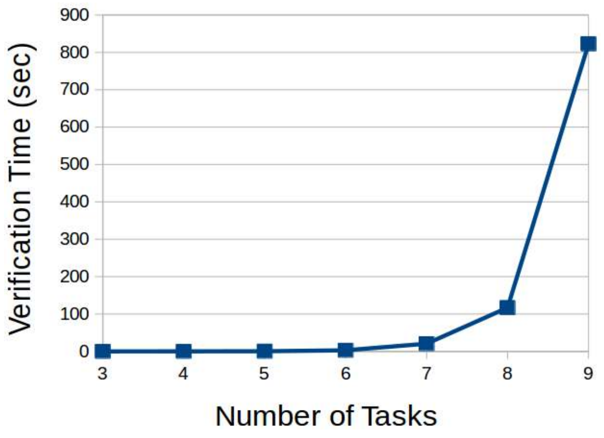


Fig. 11. Verification times of UPPAAL MC

VII. CONCLUSION

The static power dissipation is more important than ever as dynamic power dissipation reduce by the advance of CMOS technologies. Moreover, parameters of real-time components to consider for real-time guarantee are so diverse that the classical energy management and efficiency analysis techniques shows a lot of limitations.

This paper presents a novel model-based technique based on the schedulability theory of the compositional framework to compute a static minimal frequency ratio of processors. To the end, this paper presented a new supply bound function depending on a periodic resource model, PLRM, that provides a minimal CPU cycles to a given task set in the worst-case. Using the periodic model, a real-time component is verified schedulable for a given frequency ratio.

In addition, using the new periodic resource model for frequency ratio, we presented schedulability conditions for preemptive task set limited by a minimal frequency ratio of a processor. To enlarge the extensibility of our framework, we captured scheduling units under EDF and RM and the PLRM resource model in formal models of UPPAAL so that more various resource demand can be represented by the scheduling unit model in automaton models and analyzed against the resource model given in the same formalism. In the comparison with a RTC-based method, we showed that our method requires the average 36.21% less frequency ratio for the task sets under RM than the comparable RTC method does.

In the future work, we will study an automated computation technique for a frequency interface for a given component based on our new schedulability conditions and apply this approach to energy-aware composition of hierarchical scheduling systems.

REFERENCES

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 95–105, Dec 2001.

[2] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. Energy-aware scheduling for real-time systems: a survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):7, 2016.

[3] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[4] E. Bini, G. Buttazzo, and G. Lipari. Minimizing cpu energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(4):31, 2009.

[5] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Widening the schedulability of hierarchical scheduling systems. In I. Lanese and E. Madelaine, editors, *11th FACS 2014, Bertinoro, Italy, September 10-12, 2014, Revised Selected Papers*, volume 8997 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2014.

[6] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. A reconfigurable framework for compositional schedulability and power analysis of hierarchical scheduling systems with frequency scaling. *Science of Computer Programming*, 113:236–260, 2015.

[7] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[8] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[9] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 129–138, Dec 2007.

[10] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, pages 197–202, Aug 1998.

[11] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

[12] Y. Liu, A. Maxiaguine, S. Chakraborty, and W. T. Ooi. Processor frequency selection for soc platforms for multimedia applications. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 336–345, Dec 2004.

[13] D. Locke, L. Lucas, and J. Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[14] D. Mosse, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *In Workshop on Compilers and Operating Systems for Low Power*, 2000.

[15] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 89–102. ACM, 2001.

[16] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority rt-systems. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 106–114. IEEE, 2003.

[17] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(1):200–224, 2006.

[18] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.

[19] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13, 2003.

[20] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

[21] G. M. Tchamgoue, J. Seo, K. H. Kim, and Y.-K. Jun. Compositional power-aware real-time scheduling with discrete frequency levels. *Journal of Systems Architecture*, 61(7):269–281, 2015.

[22] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of IEEE International Symposium on Circuits and Systems*, volume 4, pages 101–104. IEEE, 2000.

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[24] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382, Oct 1995.

[25] D. Zhu, R. Melhem, and B. R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):686–700, July 2003.