Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 815230, 12 pages
http://dx.doi.org/10.1155/2015/815230

Research Article

A Formal Approach for RT-DVS Algorithms Evaluation Based on Statistical Model Checking

Shengxin Dai, Mei Hong, Bing Guo, Yang He, Qiongyu Zhang, Lin Sun, and Yi Du

College of Computer Science, Sichuan University, Chengdu 610065, China

Correspondence should be addressed to Mei Hong; hongmei@scu.edu.cn

Received 1 June 2015; Accepted 12 July 2015

Academic Editor: Xiaoyu Song

Energy saving is a crucial concern in embedded real time systems. Many RT-DVS algorithms have been proposed to save energy while preserving deadline guarantees. This paper presents a novel approach to evaluate RT-DVS algorithms using statistical model checking. A scalable framework is proposed for RT-DVS algorithms evaluation, in which the relevant components are modeled as stochastic timed automata, and the evaluation metrics including utilization bound, energy efficiency, battery awareness, and temperature awareness are expressed as statistical queries. Evaluation of these metrics is performed by verifying the corresponding queries using UPPAAL-SMC and analyzing the statistical information provided by the tool. We demonstrate the applicability of our framework via a case study of five classical RT-DVS algorithms.

1. Introduction

Energy saving has become a crucial concern for embedded real time systems, especially for the battery-powered systems. High energy consumption not only shortens the lifespan of computing system but also leads to high processor temperature that degrades system performance, robustness, and reliability. Therefore, system level power management has gained significant attention in recent years.

Dynamic voltage scaling (DVS) is a way to save energy in processor, which adjusts the supply voltage and operating frequency dynamically. The energy consumption scales quadratically to the supply voltage ($E \propto V^2$), and the operating frequency almost linearly depends on the supply voltage. DVS permits trading system performance for energy saving through frequency and voltage scaling [1]. However, changing the operating frequency will affect the execution time of the tasks and may violate the timing constraints. Real time DVS (RT-DVS) [1] is proposed to achieve energy saving while ensuring temporal correctness.

A number of RT-DVS algorithms [1–4] have been developed in the last two decades. Many of these algorithms focus on periodic tasks, assuming that the tasks timing attributes including period, worst-case execution time (*WCET*, at the maximum available processor frequency), and deadline are known in advance. These algorithms have been experimentally studied using simulation based approaches both on software simulators [5, 6] and on hardware platforms [7, 8]. However, simulation based evaluation is time consuming and often not completely reliable. Hence, we propose evaluating DVS algorithms with statistical model checking approach which is based on mathematical models and logics.

In this paper, we illustrate the applicability of statistical model checking [9], an automatic formal verification technique, to the area of RT-DVS algorithms evaluation. The relevant components involved in RT-DVS algorithms evaluation are modeled as stochastic timed automata [10] and implemented in statistical model checker UPPAAL-SMC [10]. Subsequently, the evaluation metrics including utilization bound, energy efficiency, battery awareness, and temperature awareness are expressed as statistical queries and written in metric interval temporal logic (MITL) [10]. Finally, the system model is verified with respect to statistical queries using UPPAAL-SMC. The model checker estimates the probability of the system to satisfy the queries. What is more, the tool provides statistical information, for example, probability densities and means, which is used for further evaluation. We propose a scalable framework for

RT-DVS algorithms evaluation. In the sequel, a case study is conducted in order to demonstrate the applicability of our framework.

To the best of our knowledge, this is the first research effort that proposes evaluating RT-DVS algorithms using statistical model checking. Our conclusions are partly observed by the simulation based approaches. However, we obtain these conclusions in a more automatic and reliable manner. What is more, we consider the impact of temperature on the energy efficiency of RT-DVS algorithms. In addition, it should be emphasized that our approach is not limited to the algorithms described in this paper, but all the dynamic voltage scaling algorithms can be modeled and then evaluated with our framework. Furthermore, although multicore systems are increasingly used, single core systems are still of interest since the approaches can be applied to multicore systems with some modification.

The rest of the paper is structured as follows. Section 2 introduces the fundamental theories of RT-DVS algorithms. Section 3 introduces statistical model checking based approach for RT-DVS algorithms evaluation. Section 4 describes the system model in detail. Section 5 describes the statistical queries for evaluation. Section 6 presents a case study to evaluate five classical RT-DVS algorithms. Section 7 gives an overview of the related work. Finally, Section 8 concludes the paper.

2. Power Consumption and RT-DVS Algorithms

This section introduces some fundamental theories of RT-DVS algorithms, which are the foundation of the models presented later.

2.1. Processor Power Distribution. The power consumption of the processor can be divided into three components: dynamic, static, and short circuit power [4]. Short circuit power is comparatively insignificant to dynamic and static power; thus it is negligible [4], so we do not discuss it in this paper.

A DVS enabled processor has a discrete set of operating frequencies $F = \{f_1, f_2, \ldots, f_n\}$, and each clock frequency is associated with a supply voltage level in set $V = \{v_1, v_2, \ldots, v_n\}$. Meanwhile processor has two modes of operation: (1) idle mode when it does not have any job in the system and thus dissipates only static power and (2) active mode when the processor executes some jobs in the system and dissipates both dynamic power and static power [11]. For many years, dynamic power dissipation was the dominant factor in total power consumption which can be approximated with the following formula:

$$P_{\text{dyn}} = C_l \cdot N_{\text{sw}} \cdot V_{dd}^2 \cdot f, \tag{1}$$

where C_l is the load capacitance, N_{sw} is the average number of circuit switches per clock, V_{dd} is the supply voltage, and f is the operating frequency. The operating frequency almost linearly depends on the supply voltage [4]:

$$f = k \cdot \frac{(V_{dd} - V_t)^2}{V_{dd}}, \tag{2}$$

where k is a constant and V_t is the threshold voltage. By substituting (2) in (1), the dynamic power at supply voltage v_k will be formulated as [12]

$$P_{\text{dyn}}(v_k) = C_0 \cdot v_k^3, \tag{3}$$

where C_0 is a processor specific constants and $v_k \in V$ is kth voltage level.

Static power or leakage power increases its dominance of total power consumption in recent years, which can be formulated as [11]

$$P_{\text{sta}} = N_{\text{gate}} \cdot I_{\text{leak}} \cdot V_{dd}, \tag{4}$$

where N_{gate} represents the number of gates, I_{leak} is the leakage current, and I_{leak} has a nonlinear exponential relationship with both temperature and supply voltage. Fortunately, it has been shown in [13] that the static power can be approximatively calculated with

$$P_{\text{sta}}(v_k) = (C_1(k) + C_2(k) \cdot T) \cdot v_k, \tag{5}$$

where $C_1(k)$ and $C_2(k)$ are some processor specific voltage sensitive constants and T is the processor temperature. As reported in [13], using (5) can accurately estimate the static power, with relative error less than 1.3%.

In general, the total power of the processor at active model is formulated as

$$\text{Pow}(v_k) = C_0 \cdot v_k^3 + (C_1(k) + C_2(k) \cdot T) \cdot v_k. \tag{6}$$

Lastly, we discuss the processor temperature in detail. We adopt the well-known lumped RC model [12] to describe the transient behavior of the processor temperature as

$$\frac{\partial T(t)}{\partial t} = \rho \cdot \text{Pow}(t) - \beta \cdot T(t), \tag{7}$$

where ρ and β are processor specific thermal coefficients and Pow(t) and $T(t)$ are the instantaneous power and temperature, respectively. Without loss of generality, we have scaled the temperature such that the ambient temperature is considered zero. Otherwise, an offset equals the ambient temperature which is substituted in (7).

2.2. RT-DVS Algorithms. RT-DVS algorithms exploit two types of slack time [3]: (1) static slack, when the total utilization is less than 1, there exists time interval that can be used for voltage scaling and (2) dynamic slack, when the actual execution time of the task is smaller than its *WCET*, a time interval is generated to further reduce the supply voltage. RT-DVS algorithms differ in the way they estimate and utilize

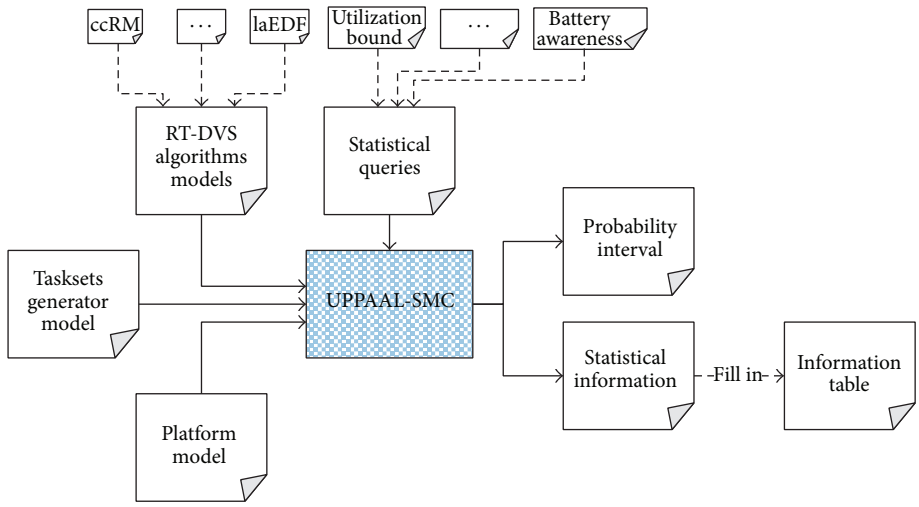


Figure 1: Illustration of our approach.

the static and dynamic slacks; accordingly, they can be further divided into three types as follows:

(1) We have static voltage scaling [1], for example, static Earliest Deadline First (*staEDF*) and static Rate Monotonic (*staRM*), which only exploits the static slacks by setting the operating frequency to the lowest possible one that guarantees the taskset feasibility. Note that the frequency is set at the starting and never changed during execution.

(2) We have cycle conserving scaling [1], for example, cycle conserving EDF (*ccEDF*) and cycle conserving RM (*ccRM*), which exploits the dynamic slacks by updating the utilization whenever a task is released or terminated; thus when a task is terminated early, the remaining time can be used to scale the frequency down.

(3) We have look-ahead scaling [1], for example, look-ahead EDF (*laEDF*), which exploits both static and dynamic slacks by pushing tasks as close as possible to their deadlines and sets the operating frequency to the slowest possible one that ensures that all future deadlines are met.

3. RT-DVS Algorithms Evaluation Based on Statistical Model Checking

3.1. Statistical Model Checking and UPPAAL-SMC. Statistical model checking [9] is one of the quantitative verification [14] approaches, which efficiently addresses verification problem by combining the Monte Carlo method with temporal logic model checking [15]. It automatically samples the traces (or simulations) of the system model, checks whether the samples satisfy or violate the quantitative property, and finally applies a statistical estimation technique to compute an approximate value for the probability that the property is satisfied [15]. The estimation is correct up to some confidence level that can be parameterized by user.

UPPAAL-SMC [10] is a stochastic and statistical model checking extension of UPPAAL toolbox for verification of stochastic hybrid systems. The modeling formalism of UPPAAL is an extension of classical timed automata with additional features such as integer variables, data structures, user-defined functions, and synchronization. And the modeling formalism of UPPAAL-SMC is based on a stochastic interpretation and extension of the timed automata used in UPPAAL; it has been extended to handle stochastic and nonlinear dynamic behaviors, discrete probabilities, a stochastic interpretation for timed delays, and even dynamic process creation [10]. In UPPAAL-SMC, a system is represented as a network of interacting stochastic timed automata (STAs), which communicate via broadcast channels and shared variables [10]. Three types of queries are supported by UPPAAL-SMC, that is, probability estimation, hypothesis testing, and probability comparison. The probability confidence interval can be estimated by the query $Pr[bound](\varphi)$, the hypothesis testing is achieved by the query $Pr[bound](\varphi) >= p$ (where p is a specified threshold), and probability comparison is achieved by the query $Pr[bound_1](\varphi_1) >= Pr[bound_2](\varphi_2)$, where *bound* defines how to bound the runs. There are three ways to bound the runs, they are by time (specifying $<= N$ where N is positive integer), by cost (specifying $x <= N$ where x is a specific clock), and by number of steps (specifying $\# <= N$) [10]. In addition to the three classical queries, UPPAAL-SMC supports the evaluation of expected values of min or max of an expression that evaluates to a clock or an integer value; the queries are $E[bound; C](min : expr)$ or $E[bound; C](max : expr)$, respectively, where *bound* is as previously explained, C accounts for the number of runs explicitly, and *expr* is the expression to be evaluated.

3.2. Statistical Model Checking Based Approach for RT-DVS Algorithms. Our approach is illustrated in Figure 1. Each RT-DVS algorithm is modeled as timed automata. Tasksets generator model consists of tasksets attributes generator model and task template. It is used to generate the tasksets to which the algorithms can be applied. Platform model consists of processor model and battery model, in which the variation of energy consumption, processor temperature, and the electronic charges in battery is modeled according

to the analytical models proposed in existing papers, such as [3, 12, 13, 16]. UPPAAL-SMC automatically composites the models in advance; then the entire system model for evaluation can be obtained.

There are four evaluation metrics in our approach, and they are expressed as statistical queries and written in MITL as follows:

(1) The utilization bound U_b of an algorithm A is the maximum utilization that any tasksets whose utilization is smaller than U_b are schedulable according to A [17]. Bigger utilization bound indicates that the corresponding algorithm has better scalability for tasksets scheduling.

(2) Energy efficiency of an algorithm is represented by the total energy consumption of the system in a bounded time. Less energy consumption indicates that the corresponding algorithm is more energy efficient.

(3) Battery awareness is represented by battery utilization which is the ratio of the energy dissipated until the battery is empty to the total capacity of the battery. Higher battery utilization indicates that the corresponding algorithm is more battery aware; that is, more charges have been used before the battery cannot provide energy for the first time.

(4) Temperature awareness is represented by the expected value of the maximum processor temperature. Higher expected temperature indicates that the corresponding algorithm is less temperature aware; that is, it is more likely to cause a hotspot.

UPPAAL-SMC is employed to verify the system model with respect to the statistical queries. We alter the parameters in tasksets attributes and observe how each parameter impacts the RT-DVS algorithms. Tasksets generator model is involved in every evaluation in order to generate the tasksets with respect to specified taskset characteristics. Thus, all the algorithms are applied to the tasksets with the same attributes instead of prior generated tasksets. We record the statistical information provided by the model checker in the information tables, and each item of the tables is obtained by an individual verification process. Finally, we visualize the information tables in the form of line chart, so that intuitive results can be obtained.

4. System Modeling with UPPAAL-SMC

In this section, we describe the framework for RT-DVS algorithms evaluation in detail. Each model of the framework is discussed separately; all the models are composited via channels and shared variables so as to obtain the entire model for evaluation.

We consider that a system consists of a taskset composed of n independent periodic tasks, that is, $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and it is executing upon a single DVS enabled processor. Each task is characterized by a period P, a worst-case execution time $WCET$, and a relative deadline D that equals the task period. Each task generates infinite jobs, with the first job arriving at time zero and subsequent arrives every P time units. We assume that the voltage scaling and the task preemption overheads are negligible, in both the time and the energy consumed.

4.1. Global Data Structures. In order to store the attributes of the system, some data structures are defined, and the UPPAAL-SMC declaration of global data structures is shown as follows:

const int $N = \cdots$;

typedef int $[0, N-1]$ t_id;

double period $[N]$;

double wcec $[N]$;

const double $BCEF = \cdots$

const int $FRES = \cdots$

typedef int $[0, FRES - 1]$ p_id;

const double fres $[FRES] = \{\cdots\}$;

const double voltages $[FRES] = \{\cdots\}$;

const double $c0 = \cdots$;

const double $c1$ $[FRES] = \{\cdots\}$;

const double $c2$ $[FRES] = \{\cdots\}$;

const double $p = \cdots$;

const double $b = \cdots$,

where the number of tasks is encoded with constant N and t_id type declaration indicates that task identifiers are integers ranging from 0 to $N - 1$. The task period and worst-case execution cycles ($WCEC$, i.e., the value obtained by multiplying $WCET$ by the maximum frequency) are stored in two arrays named $period[N]$ and $wcec[N]$, respectively, and task identifiers are used to index these two arrays. The constant $BCEF$ defines the best case execution cycles fraction of $WCEC$; hence, the actual execution cycles will be in range of $[BCEF * WCEC, WCEC]$. The number of scaling levels represents how many discrete operating frequencies that the processor supports is encoded with constant $FRES$, and f_id type declares frequency identifiers. All the discrete operating frequencies are normalized with respect to the maximum frequency, which are stored in $fres[FRES]$ array and the corresponding supply voltages are stored in *voltages[FRES]*; both of the two arrays are indexed by frequency identifiers. The identifier of the current frequency is encoded with Fre, which is initially set to 1. And the remaining variables are the processor specific parameters which have been described in formulas (6) and (7).

4.2. Tasksets Generator Modeling

4.2.1. Tasksets Attributes Generator Modeling. To evaluate the RT-DVS algorithms requires a method of generating tasksets to which the algorithms can be applied. Two important taskset parameters for tasksets generation are the taskset cardinality n and the taskset utilization u. The tasksets generation algorithm sets the utilization of each task; meanwhile, it guarantees the total utilization $\sum_{i=1}^{n} U_i = u$, where $U_i = WCET_i/T_i$ is the utilization of τ_i. Once task period has been

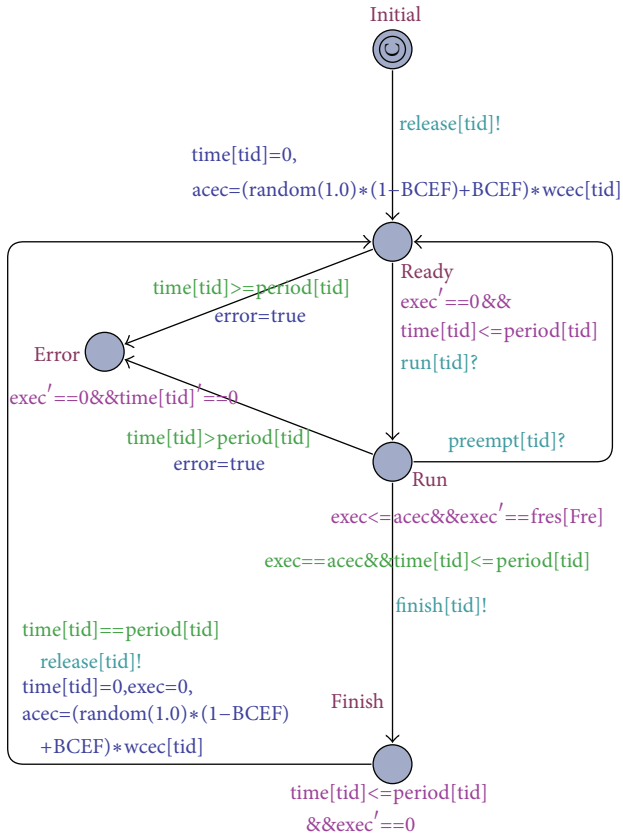


Figure 2: Task template.

chosen, its *WCET* and *WCEC* can be calculated in the sequel. *UUniFast* [18] algorithm is an efficient algorithm for task utilization generation, which generates unbiased utilization values and supports parameter independence.

The tasksets attributes generator is modeled as timed automata in form of UPPAAL template, which has two locations that linked with one transition in which *UUniFast* algorithm is implemented, and the initial location is a committed location. This model takes three parameters: the taskset cardinality n, the taskset utilization u, and the task cardinality period p. The first two parameters have been described in advance, and the task cardinality period p is used to bound the tasks period in range of $[5 * p, 10 * p]$. Task attributes can be obtained with combining all the parameters with *UUniFast* algorithm.

4.2.2. Real Time Task Modeling. The task attributes are associated with a timed automata template that captures the states of tasks. The task template is depicted in Figure 2, which takes a single parameter, namely, the task identifier *tid*, which is used to index the task attributes arrays as well. Each task is represented by an instance of the task template parameterized with the task attributes. And all the jobs (instances of the task) of one task share the same task identifier.

Whenever a job is released, it signals the scheduler that it is ready for execution using *release*[*tid*]. Meanwhile, two clock variables called *time*[*tid*] and *exec* are initialized, where *time*[*tid*] represents the time since the beginning of the current period and *exec* is a stopwatch used for counting the execution cycles in current period, respectively. Moreover, the actual execution cycles *acec* of this job is calculated with formula $acec = (random(1.0) * (1 - BCEF) + BCEF) * wcec[tid]$; that is, it is a value in range of $[BCEF * WCEC, WCEC]$.

The invariant on location *Ready* indicates that *exec* does not progress ($exec == 0$); that is, the job is not running, and *time*[*tid*] cannot be larger than the task period. The location *Run* corresponds to busy executing, in which the growth rate of *exec* is *fres[Fre]*, that is, the current normalized frequency. When *exec* is equal to *acec*, the template moves to *Finish* location and stays in this location until the next period starts. When the job is running, it may be preempted by a higher priority job; this behaviour is implemented by *preempt*[*tid*]. Whenever the job misses its deadline, that is, $time[tid] > period[tid]$, the template moves to *Error* location and sets the value of global variable *error* to TRUE.

4.3. RT-DVS Algorithms Scheduler Modeling. In order to dispatch the tasks and implement the RT-DVS algorithms the scheduler template is introduced. Each RT-DVS algorithm is implemented with an individual template; however, all the templates are similar. Besides, some extra data structures should be appended in order to provide the information as required, for example, last *acec* of the task for *ccEDF*. Although it is not a trivial work to implement the templates, we will not describe all the templates but detailedly explain one template instead. We use *ccEDF* algorithm template as an example; it is shown in Figure 3.

The scheduler template is equipped with a task identifier array named *queue*, and all the elements in *queue* are initialized to −1 with function *Initial*(). It stays in *Idle* location unless a job is released or finished execution. When *release*[*t*] arrives, the scheduler inserts the task identifier into *queue* based on EDF policy using the function *schEDF*(*t*) and then adjusts the operating frequency using the function *ccEDFR*(*t*), in which the corresponding function *task_release*() [1] is implemented. After that, the scheduler alters the state of the relevant jobs; if the incoming job has higher priority than the running job, the scheduler preempts the running job and dispatches the incoming job to run; else if *queue* is empty, the incoming job is dispatched to execute; otherwise, no operation is performed.

Upon receiving *finish*[*t*], the scheduler pops the first task identifier out of *queue* and then adjusts the operating frequency using the function *ccEDFF*(*t*), in which the corresponding function *task_completion*() [1] is implemented. Finally, the scheduler selects the next highest priority job in *queue* (if it is not empty) to execute. Committed location (encircled with C) is employed to achieve atomic transition; thus *Idle* is the only location where the time can elapse.

4.4. Platform Modeling

4.4.1. Processor Modeling. The template of processor alternates between *Idle* and *Active* locations, which represent the two modes of processor operation, as shown in Figure 4. Two variables named *TotalEnergy* and *T* are defined in order

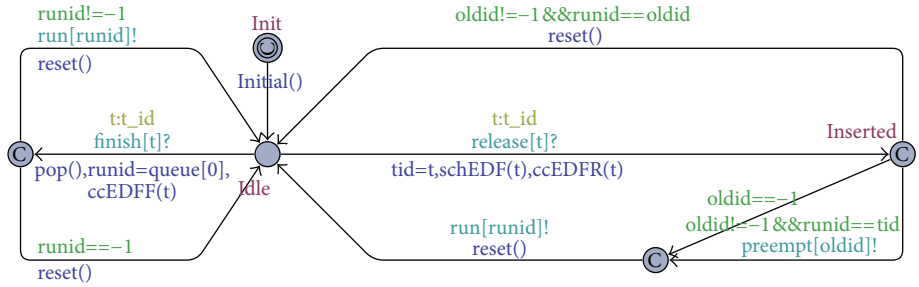


Figure 3: *ccEDF* template.

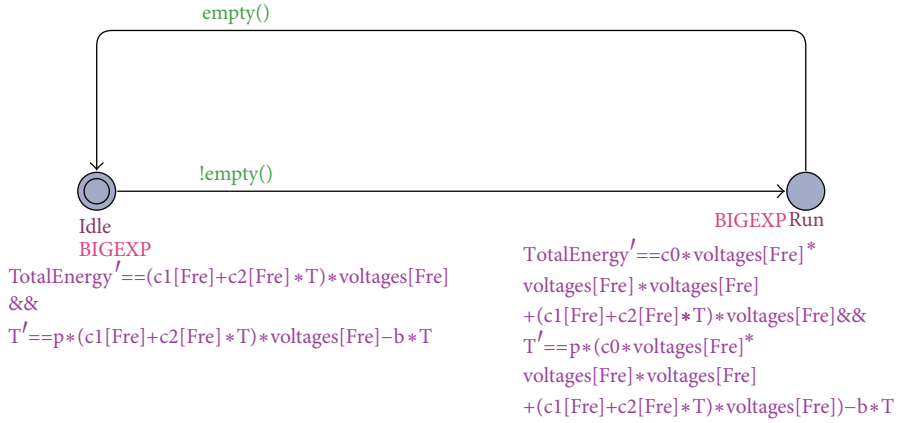


Figure 4: Processor template.

to capture the energy consumption and the instantaneous processor temperature, respectively. The invariants of the two locations describe the variation of the two variables based on formulas (6) and (7).

4.4.2. Battery Modeling. The kinetic battery model (KiBaM) [16] has proven to be the best suited model for performance modeling and cooperate with possible workload model. The model partitions the battery charge into two wells: available charge well $ava(t)$ and bound charge well $bou(t)$. The total capacity C is partitioned into the two wells with a factor of c; hence, initially the charges in the two wells are $ava(0) = c * C$ and $bou(0) = (1 - c) * C$. The available charge well provides energy to the current load $l(t)$, and the bound charge well provides electrons to the available charge well. The charge flows from the bound charge well to the available charge well at a rate proportional to a conduction parameter k and the height difference between the two wells. The battery is considered empty when there is no charge left in the available charge well; that is, $ava(t) = 0$. The variation of the charge in both wells is described as a system of ordinary differential equations:

$$\begin{aligned} \frac{\partial ava(t)}{\partial t} &= -l(t) + k \cdot \left(\frac{bou(t)}{1-c} - \frac{ava(t)}{c} \right), \\ \frac{\partial bou(t)}{\partial t} &= -k \cdot \left(\frac{bou(t)}{1-c} - \frac{ava(t)}{c} \right). \end{aligned} \tag{8}$$

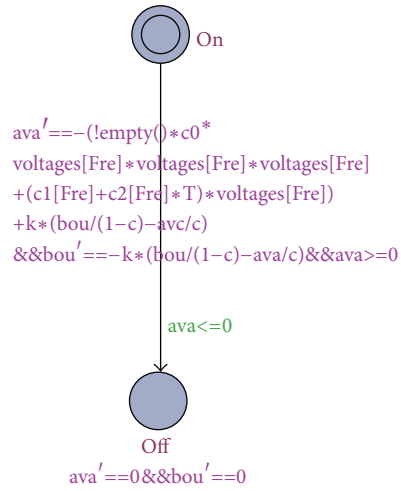


Figure 5: Battery template.

The template of the battery mainly consists of two locations On and Off, which indicate that the battery is available and empty, as shown in Figure 5. The invariant on location On represents the variation of the charge in both wells based on formula (8). When there is no charge left in the available charge well, the template moves to Off location.

5. Statistical Queries for Evaluation

In this section, we list out the statistical queries of the corresponding evaluation metric.

Seeking the utilization bound of an algorithm A is regarded as a hypothesis problem. For a given taskset utilization u, is the probability of the tasksets are not schedulable is smaller than or equal to a threshold q? And the verification is limited up to B time units. We increase u until the query is not satisfied for the first time, and the value of u is the utilization bound of A. The corresponding query for model checking is

$$Pr\left[<= B\right]\left(error\right) <= q. \quad (9)$$

If UPPAAL-SMC reports that the query is satisfied, the tasksets are deemed schedulable.

The key for energy efficiency evaluation is to estimate the energy consumption under each algorithm within a time bound B. The corresponding query is

$$Pr\left[TotalEnergy <= MAX\right]\left(gTime >= B\&\&!error\right), \quad (10)$$

where $TotalEnergy$ records the energy consumption, MAX is a constant larger than the reachable range of the energy consumption of all runs, and $gTime$ is a clock that indicates the time since the beginning. Thus, this query computes the probability of reaching time beyond B and the system is not error within a large energy bound. One thing should be emphasized here is that $!error$ is necessary in order to eliminate the impact of unschedulable samples; UPPAAL-SMC discards the unschedulable samples and generates more traces to meet the defined confidence level.

Battery awareness is represented by the battery utilization. With a known battery capability, the total energy dissipated until the battery is empty is required to calculate the utilization. The corresponding query is

$$Pr\left[TotoalEnergy <= MAX\right]\left(gTime > 0\&\&ava <= 0\&\&!error\right), \quad (11)$$

in which ava indicates the charge left in available well, and $gTime > 0$ is used to eliminate the impact of the initial state.

The expected value of the maximum processor temperature is required for temperature awareness evaluation. The corresponding query is

$$E\left[<= 10000 : 500\right]\left(max : T\right), \quad (12)$$

in which T indicates the processor temperature which has been scaled down to the ambient temperature. Thus, this query performs 500 runs and returns the expected value of the maximum processor temperature.

6. Case Study

In this section, in order to illustrate the applicability of our framework for RT-DVS algorithms evaluation, we perform a case study using our framework to evaluate five classical RT-DVS algorithms.

Table 1: Parameters for the linearized leakage power.

f_k	v_k	C_1	C_2
1.0	1.5	1.4038	0.0641
0.8	1.2	0.7439	0.0308
0.6	0.9	0.390	0.0077
0.4	0.6	0.204	0.0081

6.1. Experimental Setting. We choose RT-DVS algorithms proposed in [1], because these algorithms are the most cited ones in the literature, and they cover all the three types of RT-DVS algorithms as introduced in the previous section.

In our experiments, tasksets are generated with tasksets generator model, and taskset utilization is varied with a step size of 0.05 within the range of $[0.5, 1]$. The cardinality number and period are 4 and 100, respectively. The default processor characteristics are obtained from [12], whose normalized frequencies are varied with a step size of 0.2 within the range of $[0.4, 1]$; the processor specified consent $C_0 = 6.68$ and parameters for the linearized leakage power are listed in Table 1, and the thermal coefficients $\rho = 0.0071^{\circ}$C/W and $\beta = 0.0041$.

The parameters in statistical queries are set as follows: the time bound $B = 10000$ and the threshold $q = 0.001$. The statistical parameters for verification are set as follows: the probability of false negatives $\alpha = 0.001$, the probability uncertainty $\varepsilon = 0.01$, and both lower and upper bound of indifference region $\delta = 0.005$.

6.2. Utilization Bound Evaluation. In these experiments, we seek the utilization bound of the RT-DVS algorithms. It is shown that all tasksets are schedulable according to EDF based algorithms if the taskset utilization is not greater than 1. And according to both RM based algorithms, tasksets are schedulable if the taskset utilization is 0.75 but not schedulable if the taskset utilization is 0.8. In order to get more precise bound, we change the step size to 0.01. Finally, we observed that, according to *staRM*, tasksets are schedulable if the taskset utilization is 0.79 but not schedulable if the taskset utilization is 0.8, and according to *ccRM* tasksets are schedulable if the taskset utilization is 0.78 but not schedulable if the taskset utilization is 0.79. Thus the utilization bound of *staRM* and *ccRM* is 0.79 and 0.78, respectively.

6.3. Energy Efficiency Evaluation. In these experiments, we investigate the energy efficiency of all RT-DVS algorithms. The original EDF algorithm is included as a baseline case in evaluation similar to [1]. The energy consumption of each RT-DVS algorithm is normalized with respect to the energy consumption of the original EDF algorithm.

6.3.1. Taskset Utilization. To investigate the impact of the taskset utilization on the energy efficiency of RT-DVS algorithms, experiments with various utilization were performed. And *BCEF* is set to 1 so that the tasks do consume their *WCEC*; thus *staEDF* and *ccEDF* have the same behavior.

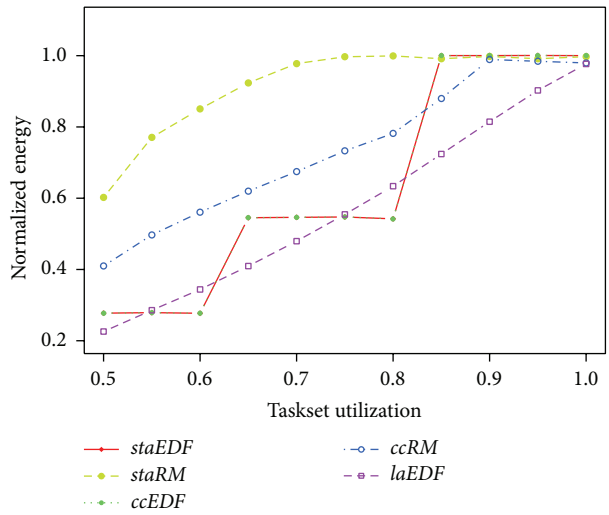
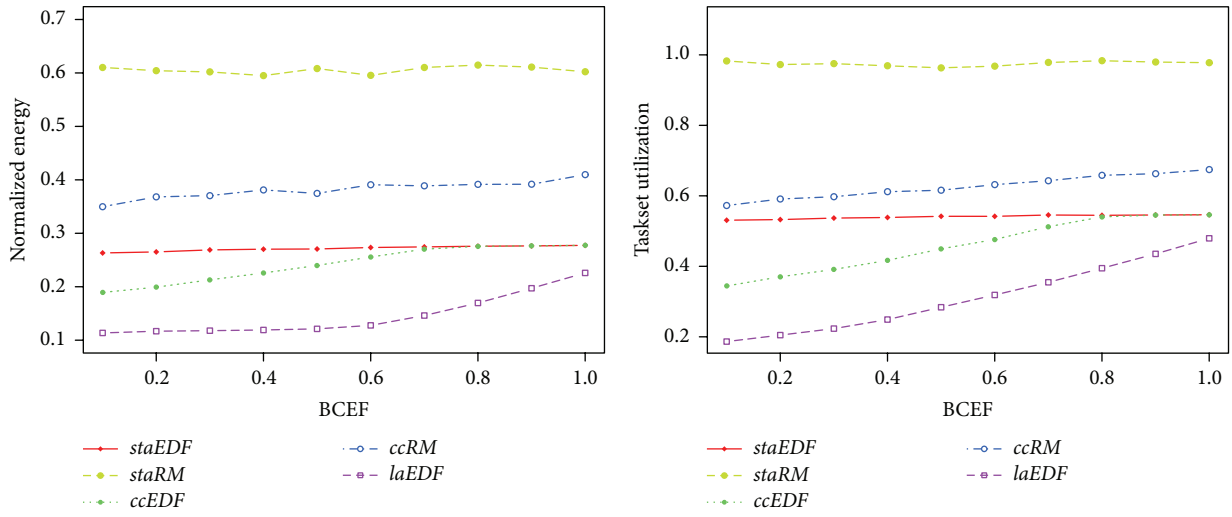


Figure 6: Utilization impact.



(a) *Utilization* = 0.5

(b) *Utilization* = 0.7

Figure 7: *BCEF* impact.

As shown in Figure 6, the energy efficiency of all algorithms decreases dramatically as the utilization increases. And EDF based algorithms are more efficient than RM based algorithms. One interesting observation is that the curve of *staEDF* is in the likeness of stairs, which have salient points when the task utilization is equal to the discrete frequencies; that is, $u = 0.6$ and $u = 0.8$. This is because of the fact that *staEDF* uses the taskset utilization to scale the operating frequency. What is more interesting is that at the salient points *staEDF* is more efficient than the aggressive algorithm *laEDF*. This is because *laEDF* defers works in order to take advantage of early termination. However, in these experiments we assume that all tasks do consume their *WCEC*; this requires operating at high frequencies later in order to meet all the deadlines that may dissipate more energy.

6.3.2. BCEF. In this set of experiments, we investigate how each RT-DVS algorithm takes advantage of slack times. We vary the *BCEF* in range of [0.1, 1] with a step size of 0.1, and the taskset utilization is equal to 0.5 and 0.7, respectively.

As shown in Figure 7, the energy efficiency of static RT-DVS algorithms is not highly affected by *BCEF*, and for other algorithms their efficiency decreases as *BCEF* increases. This conforms to the results in [1] that the static algorithms do not take advantage of early termination, and cycle conserving algorithms and look-ahead algorithm make use of these slacks. We must emphasize that the curves of the algorithms do not indicate the actual energy consumption but its ration to the energy consumed by EDF algorithm. Therefore, the flat curves do not denote the actual energy consumption being equal at every point but indicates that the actual energy consumption is linearly increased

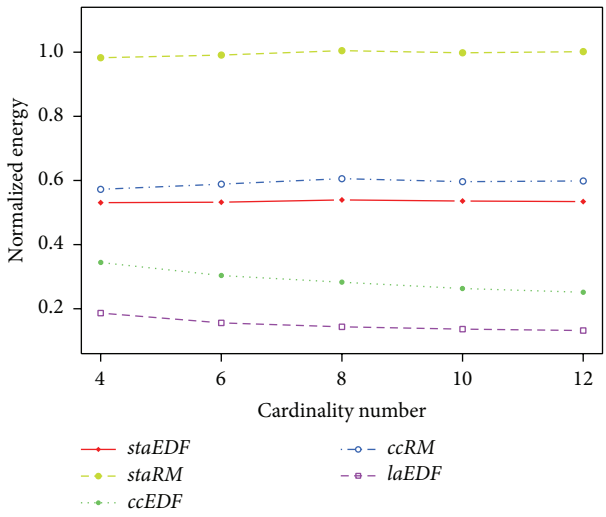


Figure 8: Cardinality impact.

Table 2: Parameters for the linearized leakage power of processor 2.

f_k	v_k	C_1	C_2
1.0	1.10	18.497	0.2149
0.9027	1.05	14.998	0.2043
0.8797	1.00	12.315	0.1942
0.8553	0.95	10.238	0.1846
0.8291	0.90	8.6126	0.1754
0.8010	0.85	7.3249	0.1666

with *BCEF* and the increase rate is almost equal to EDF algorithm.

6.3.3. Cardinality Number. In these experiments, we investigate the impact of the number of tasks on the energy consumption. Experiments with various numbers of tasks in range of $[4, 12]$ with step size of 2 were performed; meanwhile, they keep the taskset utilization $U = 0.7$ and $BCEF = 0.1$.

As shown in Figure 8, we observed that as the cardinality number increases, the normalized energy consumption of all the algorithms is not highly affected.

6.3.4. Processor Characteristics. In these experiments, we investigate the impact of varying the processor characteristics. We obtained other processor characteristics from [13] (we named the default processor as processor 1 and this processor as processor 2), where the processor specified consent $C_0 = 15$ and parameters for the linearized leakage power are listed in Table 2, and the thermal coefficients $\rho = 0.002941^{\circ}\text{C/W}$ and $\beta = 0.003676$. In addition, we investigate the algorithms upon processors with cooling systems, cool fan for processor 1 (thus $\beta = 0.0071$) [12], water spray for processor 2 (thus $\beta = 0.043898$) [13]. Other system parameters are set as follows: taskset utilization $U = 0.7$, cardinality number $n = 4$, and $BCEF = 0.1$.

As shown in Figure 9, both RM based algorithms have a better performance upon processor 2; this is because the leakage power accounts for a larger proportion of the total power. This conforms to the results in [1] that RM based algorithms have better performance as the idle level energy consumption increases. For EDF based algorithms, when the taskset utilization is smaller than the lowest frequency available in processor 2, that is, $f = 0.801$, there is no difference among these algorithms, and they have worse performance upon processor 2; it is clear that this is because of the lowest available frequency limitation. However, when the taskset utilization is bigger than the lowest frequency, all algorithms have better performance upon processor 2; thus the curves grow much gently. This is because more frequencies are available so that the algorithms can select the most suitable operating frequency that results in better energy efficiency.

In addition, it is shown that although cooling systems highly decrease actual energy consumption, the normalized energy is not highly affected. This is because all the algorithms in our case study focus on dynamic power reducing; they do not take any advantage of the static power reducing derived from cooling systems.

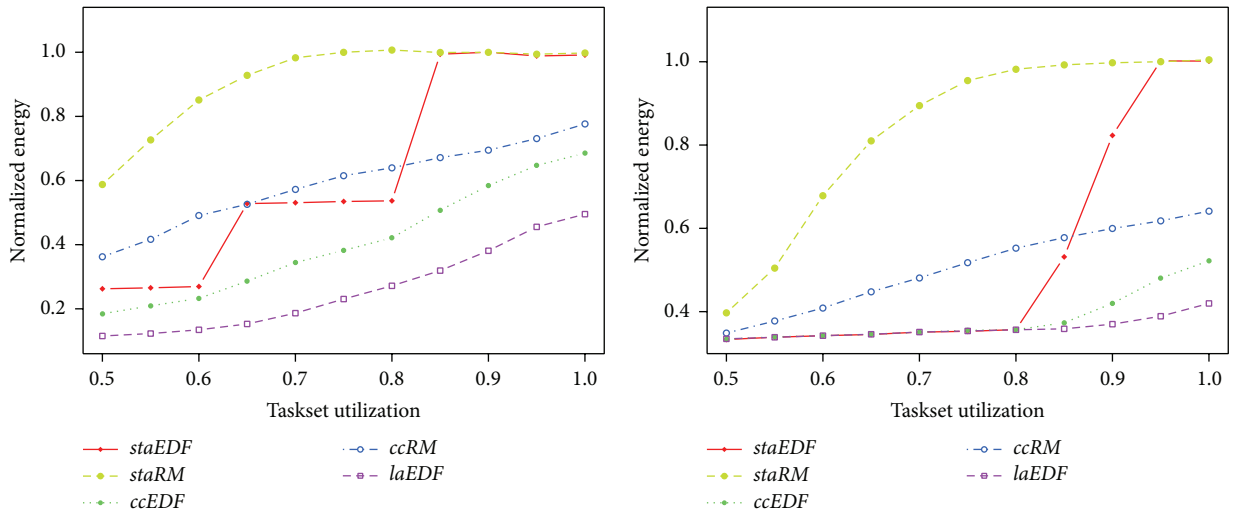
6.4. Battery Awareness Evaluation. In these experiments, we investigate how each RT-DVS algorithm takes advantage of the capacity of battery. The parameters of the battery are as follows: the total capacity $C = 60000$, the factor $c = 1.0/6$, and the conduction parameter $k = 2.324 \times 10^{-4} s^{-1}$, similar to the ones used in [19]. All the results are normalized with respect to the total capacity of the battery, so that the battery utilization can be obtained.

As shown in Figure 10, the utilization of battery decreases as the taskset utilization increases. This is because as the taskset utilization increases less time is left for the battery to recover charges from bound well; this is also the reason why various RT-DVS algorithms utilize different percent of the total battery capacity. It can be observed that the utilization of the battery under look-ahead RT-DVS algorithm (93% of the total energy) is over twice as that under original EDF algorithm (42% of the total energy) when the taskset utilization is low.

6.5. Temperature Awareness Evaluation. In these experiments, we investigate the expected value of the maximum processor temperature at every taskset utilization level under each RT-DVS algorithm.

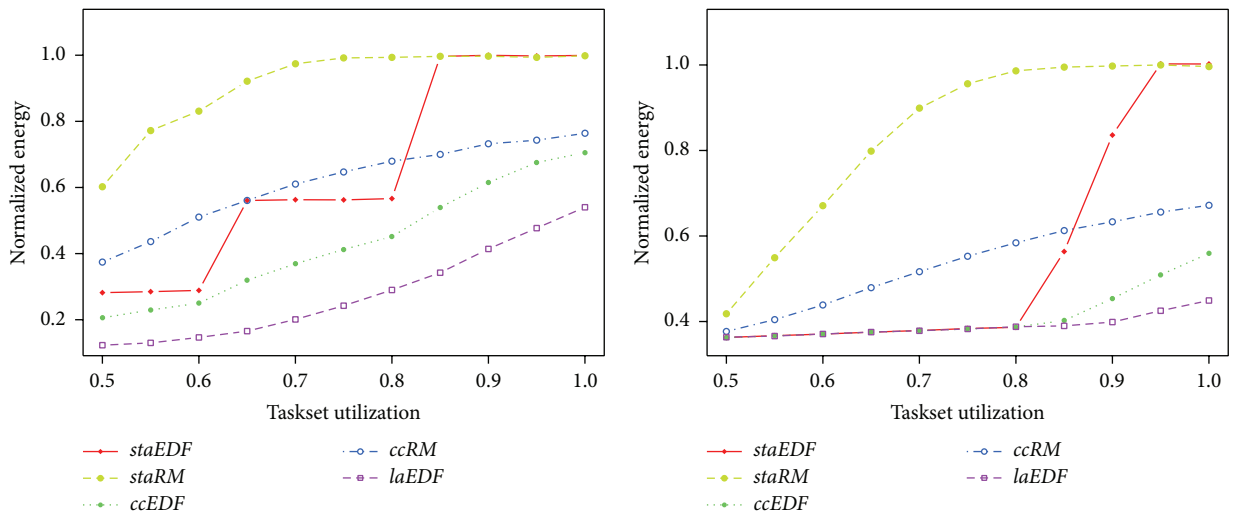
As shown in Figure 11, the maximum processor temperature grows rapidly as the utilization increases, and the growth rate is almost proportional to the energy efficiency of the corresponding algorithm.

6.6. Summary. We have evaluated how each parameter, for example, the taskset attributes and the processor characteristics, impacts the energy efficiency of the RT-DVS algorithms. In the sequel, evaluation of other metrics is also performed. Our experiments have shown that the most impact parameters on the energy efficiency of RT-DVS algorithms are



(a) Processor 1, $\beta = 0.41$

(b) Processor 2, $\beta = 0.003676$



(c) Processor 1, $\beta = 0.71$

(d) Processor 2, $\beta = 0.043898$

Figure 9: Processor impact.

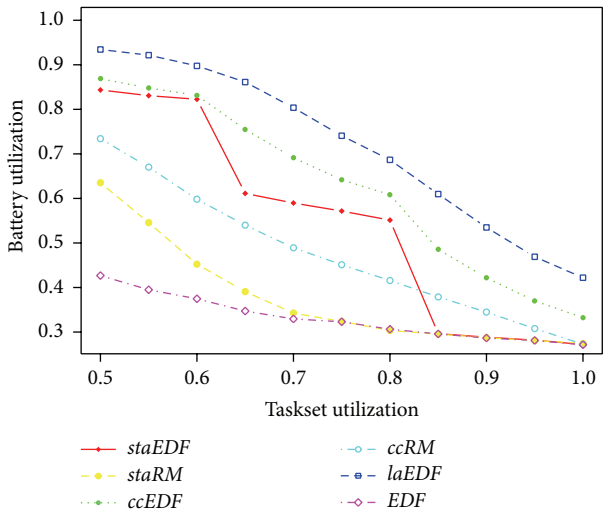


Figure 10: Battery awareness.

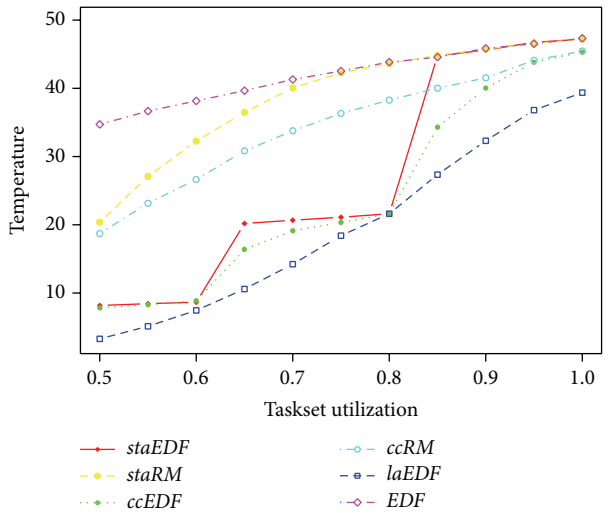


Figure 11: Temperature awareness.

the taskset utilization and processor characteristics, especially the lowest available frequency and the density of the frequencies. Moreover, we observed that the battery awareness and temperature awareness are almost proportional to the energy efficiency.

The conclusions obtained in these experiments have been partly observed by simulation based approaches. Thus, these experiments demonstrated the correctness of our approach. Furthermore, we obtained correct results while considering the impact of processor temperature.

7. Related Work

Shin et al. [6] have proposed a simulator for performance compression of DVS algorithms called SimDVS. In addition, they have performed three case studies to demonstrate the effectiveness of the simulator, including performance evaluation of InterDVS, IntraDVS, and HybridDVS algorithms, as well as overhead measurement of InterDVS algorithms; that is, the more efficient DVS algorithms may suffer more system overhead.

Kim et al. [5] have compared several DVS algorithms for hard real time periodic tasks. Their experiments were performed using a simulator environment called SimDVS [6]. They have evaluated the impact of several parameters, for example, number of tasks, utilization, scaling levels, and speed bound, on the energy efficiency of the DVS algorithms. They conclude that the existing EDF based DVS algorithms are close to optimal bound, and more research should be done for better RM based DVS algorithms.

Saha and Ravindran [20] have implemented fourteen DVS algorithms in the ChronOS real time Linux kernel and evaluated their energy efficiency on two hardware platforms, including an Intel i5 processor and an AMD Zacate processor. Workload is generated with a synthetic application and measures the energy consumption by accounting for energy consumption in the active and idle states and the system energy using a multimeter. They concluded that energy efficiency of DVS algorithms highly depends on the processor characteristics, that is, the relative power of the active and the idle states and the number of frequency steps available.

Lin et al. [8] have designed a framework including hardware settings and software implementation to evaluate power aware scheduling algorithms on Intel PXA255 XScale processor, namely, real energy. In addition, this framework was also used for testing the implementation of DVS algorithms. They have evaluated several classical DVS algorithms and concluded that these algorithms are effective for reducing the processors energy consumption; however, they also concluded that these algorithms do not effectively reduce the systems energy consumption.

Grosu et al. [21] have explored the usability of statistical model checking for measuring quantitative properties of systems. A specification formalism for measurement computation is proposed, in which a set of measurement variables are associated with each state in the state space. In the sequel, they developed a Monte Carlo technique for generating a suitable set of samples in order to satisfy the desired confidence bound. Finally, they illustrated their method for measuring the aggregate behavior of flock of bird-like agents.

Zhang et al. [22] have presented a formal framework for symbolic analysis of programmable logic controllers (PLC) systems. The framework provides automatic analysis of reliability calculations and performance measurements. They translate the embedded control program into a logic expression and embedded the hardware uncertainty into the expression so that the reliability of the system can be obtained. The PLC system is abstracted as Hidden Markov Model, and it is combined with the reliability of system to form a regular Markov Model. Finally, the performance measurement is obtained with probabilistic model checking tools.

David et al. [23] have presented a process that combines control synthesis, model checking, and statistical model checking in order to examine the consequences of adopting the control strategy. Strategy is synthesized for a timed game and a given control objective with control synthesis techniques. Then, model checking is employed to verify correctness properties of the timed game under the synthesized strategy. Lastly, statistical model checking is applied to evaluate the performance aspects of the synthesized strategy.

David et al. [24] have presented a study of statistical model checking for analyzing performance properties of energy aware building. A framework for modeling and analysing energy aware building is presented, which allows evaluating the performance of the control strategies. Comfort time and energy consumption are obtained under various environmental settings. The framework has been applied to the Hybrid Systems Verification Benchmark.

8. Conclusion

We have shown that statistical model checking supports automatic evaluation of RT-DVS algorithms. In contrast to simulation based evaluation, the approach proposed in this paper inherit the advantages of the statistical model checking. It performs a reliable result with respect to a user-defined confidence level. Moreover, it automatically generates the tasksets and eliminates the impact of the unschedulable samples. In addition, the framework proposed in this paper can be used for verification of the algorithms. We are presently extending this work by adding more RT-DVS algorithms, battery aware algorithms, and temperature aware algorithms into this framework meanwhile capturing more processor configurations.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant no. 61332001, the National Natural Science Foundation of China under Grant no. 61272104, and the Basic Research for Application Foundation of Sichuan Province no. 2014JY0112.

References

[1] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 89–102, 2001.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.

[3] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, Article ID 1206038, p. 39, 2007.

[4] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[5] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '02)*, pp. 219–228, IEEE, September 2002.

[6] D. K. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min, "SimDVS: an integrated simulation environment for performance evaluation of dynamic voltage scaling algorithms," in *Power-Aware Computer Systems*, vol. 2325 of *Lecture Notes in Computer Science*, pp. 141–156, Springer, Berlin, Germany, 2003.

[7] J. Lin, W. Song, and A. K. Cheng, "Real-energy: a new framework and a case study to evaluate power-aware real-time scheduling algorithms," in *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED '10)*, pp. 153–158, IEEE, Austin, Tex, USA, August 2010.

[8] J. D. Lin, A. M. K. Cheng, and W. Song, "A practical framework to study low-power scheduling algorithms on real-time and embedded systems," *Journal of Low Power Electronics and Applications*, vol. 4, no. 2, pp. 90–109, 2014.

[9] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: an overview," in *Runtime Verification*, H. Barringer, Y. Falcone, B. Finkbeiner et al., Eds., Lecture Notes in Computer Science, pp. 122–135, Springer, 2010.

[10] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal SMC tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.

[11] W. P. Liao, L. He, and K. M. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1042–1053, 2005.

[12] M. Mohaqeqi, M. Kargahi, and A. Movaghar, "Analytical leakage-aware thermal modeling of a real-time system," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1377–1391, 2014.

[13] G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 329–339, 2010.

[14] M. Kwiatkowska, "From software verification to everyware verification," *Computer Science—Research and Development*, vol. 28, no. 4, pp. 295–310, 2013.

[15] E. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems," in *Automated Technology for Verification and Analysis*, T. Bultan and P.-A. Hsiung, Eds., vol. 6996 of *Lecture Notes in Computer Science*, pp. 1–12, Springer, Berlin, Germany, 2011.

[16] M. R. Jongerden and B. R. Haverkort, "Which battery model to use?" *IET Software*, vol. 3, no. 6, pp. 445–457, 2009.

[17] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, article 35, 2011.

[18] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[19] E. R. Wognsen, R. R. Hansen, and K. G. Larsen, "Battery-aware scheduling of mixed criticality systems," in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, T. Margaria and B. Steffen, Eds., vol. 8803 of *Lecture Notes in Computer Science*, pp. 208–222, Springer, Berlin, Germany, 2014.

[20] S. Saha and B. Ravindran, "An experimental evaluation of real-time DVFS scheduling algorithms," in *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR '12)*, pp. 1–12, ACM, Haifa, Israel, June 2012.

[21] R. Grosu, D. Peled, C. R. Ramakrishnan, S. Smolka, S. Stoller, and J. Yang, "Using statistical model checking for measuring systems," in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, T. Margaria and B. Steffen, Eds., vol. 8803 of *Lecture Notes in Computer Science*, pp. 223–238, Springer, Berlin, Germany, 2014.

[22] H. H. Zhang, Y. Jiang, W. N. Hung, X. Song, M. Gu, and J. Sun, "Symbolic analysis of programmable logic controllers," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2563–2575, 2014.

[23] A. David, H. Fang, K. G. Larsen, and Z. Zhang, "Verification and performance evaluation of timed game strategies," in *Formal Modeling and Analysis of Timed Systems: Proceedings of the 12th International Conference (FORMATS '14)*, vol. 8711 of *Lecture Notes in Computer Science*, pp. 100–114, Springer, 2014.

[24] A. David, D. Du, K. G. Larsen, M. Mikučionis, and A. Skou, "An evaluation framework for energy aware buildings using statistical model checking," *Science China Information Sciences*, vol. 55, no. 12, pp. 2694–2707, 2012.