Combining Optimistic and Pessimistic DVS Scheduling: An Adaptive Scheme and Analysis

Simon Perathoner, Kai Lampka, Nikolay Stoimenov, Lothar Thiele
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH)
Zurich, Switzerland
Email: perathoner@tik.ee.ethz.ch

Jian-Jia Chen
Institute for Process Control and Robotics (IPR)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: jian-jia.chen@kit.edu

***Abstract*—Performance boosting of modern computing systems is constrained by the chip/circuit power dissipation. Dynamic voltage scaling (DVS) has been applied for reducing the energy consumption by dynamically changing the supply voltage. One can optimistically apply greedy online DVS scheduling algorithms by considering only the events that have arrived in the system. However, this might require a speed that is beyond a system's capability. Alternatively, one can pessimistically use a conservative speed to ensure timing guarantees, which might consume an excessive amount of energy as events might be processed faster than necessary. This paper presents an adaptive scheme that combines these two strategies for the scheduling of arbitrary event streams. The proposed adaptive DVS scheduler chooses the execution speed dynamically as long as it is below a certain threshold. Once the speed exceeds this threshold, the proposed scheduler operates at a constant (pessimistic) speed for guaranteeing the feasibility. The computation of the threshold speed is, however, not straight-forward. For deriving it, we make use of a framework based on timed model checking because the scheduler is strongly state-dependent. The resulting analysis framework allows to obtain the threshold speed for the proposed adaptive DVS scheduling algorithm such that both timing and speed constraints are guaranteed to be met and at the same time an energy-efficient execution is ensured.**

I. Introduction

Power-aware design in both hardware and software has become a significant aspect for the design of modern computing systems. Energy-efficient devices reduce the electricity bills, or extend the battery lifetime of autonomous embedded systems. The dynamic energy consumption of a system can be reduced by means of dynamic voltage scaling (DVS), a technique that permits to trade system performance for energy savings. In general, a lower supply voltage for a processor leads not only to a lower execution speed, but also to a lower power consumption. As a result, most DVS scheduling algorithms, e.g., [18], tend to execute events as slowly as possible, without any violation of timing constraints. On the other hand, to consume less leakage (static) power, we can apply dynamic power management (DPM) to change the system state to a sleep mode when the system has no jobs to execute [7], [9].

Energy-efficient scheduling has been widely studied in the literature, coping with the minimization of energy consumption under real-time constraints of events/tasks, see e. g., [7], [9], [18]. However, most methods for energy-efficient scheduling in real-time systems assume that event streams are very regular, e.g., events arrive periodically or at most sporadically. Another common assumption is to have irregular arrival patterns, but full a-priori knowledge of event arrival times. Unfortunately, both assumptions are often not realistic, as in practice event arrivals are neither regular nor fully predictable. For instance, in many real streams events arrivals are affected by non-deterministic jitters which may lead to bursts, i.e. short time intervals with accumulated arrivals. Such more complex event streams can be characterized by *arrival curves* as used in Real-Time Calculus (RTC) [14], [17] and Network Calculus [11]. An arrival curve bounds the maximum number of events that may arrive in any time interval of a specified length.

One possibility to minimize the energy consumption of a system is to find the minimum (or most effective) *constant* speed to schedule an event stream bounded by an arrival curve. This is the approach considered in [13]. However, as the arrival curve itself does not reveal how events *actually* arrive, the system could run most of the time at an unnecessarily high speed. For instance, if bursts of events happen only very rarely in a stream, the above scheduling approach would, nevertheless, always run at a high speed in order to guarantee the schedulability of the system. In other words, an arrival curve solely defines an upper bound on the event arrivals but, in general, the actually experienced event arrivals may be far away from that bound.

Instead of a static speed assignment, one can apply on-line DVS scheduling algorithms in order to reduce the energy consumption. These algorithms decide the execution speed based on the events that actually arrive to the system. Different on-line DVS algorithms have been proposed in the literature [3], [18]. However, to guarantee the satisfaction of timing constraints, it is possible that the execution speed required by an on-line DVS scheduling algorithm is higher than the maximum speed available in the system. This imposes a careful verification of the required maximum speed before actually adopting an on-line DVS scheduling algorithm [6]. However, again, an on-line DVS algorithm might exceed the maximum available speed only for a short time interval, depending on the actual event arrivals. Falling back to a static speed assignment in such a case, as suggested by Chen et al. [6], might be overly pessimistic.

This paper proposes an adaptive DVS scheduling scheme, in which an on-line DVS algorithm is applied when the system is *light-loaded* and a pessimistic speed is assigned when the system is *heavy-loaded*. On one hand the proposed adaptive DVS scheduling algorithm reduces the energy consumption by being as optimistic as possible. On the other hand, it guarantees the feasibility of the resulting solution by switching to a pessimistic mode once the system is heavily loaded. The key issue for the adaptive DVS scheduling algorithm is to decide when to be pessimistic and when to be optimistic. This paper presents how to design and analyze such an adaptive DVS scheme by applying timed automata (TA) [1]. In order to perform the analysis by TA, we first derive an event generator based on the arrival curve of a stream by applying the approach proposed by Lampka et al. [10]. Then, we couple the event generator with a discretized and time-triggered TA model of the adaptive DVS processor. As a result, an adaptive DVS scheduler is confirmed to be safe if it passes the verification of our proposed TA scheme. By means of experiments we evaluate the performance of the adaptive DVS scheme in comparison to purely pessimistic and purely optimistic DVS approaches.

978-1-4244-8194-1/10/$26.00 ©2010 IEEE

The rest of this paper is organized as follows: Section II describes the system models. Section III presents a motivational example and the concept for the adaptive DVS scheme. How to design and analyze such a scheme is discussed in Section IV. The performance of the adaptive scheme is evaluated in Section V, and the paper is concluded in Section VI.

II. System Models and Background

In this section we briefly describe the considered system models and provide the theoretical background for the models.

A. Event Model

This paper considers a general model for event arrivals. Events could come irregularly, periodically, sporadically, or with bounded non-deterministic jitters. For example, events in real-time embedded systems are often triggered by the physical environment, which can in many cases not be predicted very accurately. It is, however, often possible to constrain the number of event arrivals in a certain interval length. This abstraction forms the basis of *arrival curves*, introduced in Network and Real-Time Calculus (RTC) [11], [14].

An event stream can be abstractly characterized by an arrival curve $\bar{\alpha}(\Delta)$, for which

$$R(t+\Delta) - R(t) \leq \bar{\alpha}(\Delta) \tag{1}$$

holds for all $0 \leq t, 0 \leq \Delta$, where $R(t)$ is the cumulative event counting function of any event trace from time 0 to time t.

Arrival curves can be seen as a generalization of classical event stream models such us periodic event arrivals or periodic events with jitters. For instance, the arrival curve for an event stream with periodicity and jitter (characterized by a period p, a maximum jitter J, and a minimum event inter-arrival time d) is computed as

$$\bar{\alpha}(\Delta) = \min\left\{ \left\lceil \frac{\Delta + J}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil \right\}. \tag{2}$$

An arrival curve can be derived from system specifications or by profiling a set of representative event traces; see [14], [15] for details. Note that by means of arrival curves one can specify arbitrary event arrival patterns. As a result, the analysis framework proposed in this paper can be directly applied to many simpler models.

For the events of an event stream, we suppose that C is their worst-case execution time at the maximum speed $s_{\max}$ and D is their relative deadline D. We can use a curve $\alpha(\Delta) = C \cdot \bar{\alpha}(\Delta)$ to denote an upper bound on the processing demand of the event stream for any time interval of size Δ, where the processing demand is expressed in units of execution time at speed $s_{\max}$.

In analogy with arrival curves, the availability of processing resources is characterized by *service curves*. In particular, we use a service curve $\beta(\Delta)$ to denote a *lower* bound on the available execution time in *any* time interval of length Δ with $\Delta \geq 0$. For arrival curves, service curves, and a given scheduling policy, RTC can be used to analyze the worst-case response time (or delay) experienced by an event or the required buffer size [14], [15].

B. Processor and DVS Model

This work assumes the power consumption that is manageable by the system as a convex and increasing function of the supply voltage/speed. We adopt the power model of [19] in which the power consumption of the system at execution speed s is specified by $P(s) = P_{sta} + \hbar(P_{ind} + P_d) = P_{sta} + \hbar(P_{ind} + C_{ef} s^\gamma)$, where P_{sta}, P_{ind}, and P_d are *static*, *speed-independent* active, and *speed-dependent* active power, respectively. If the system is in sleep mode, $\hbar$ is 0, whereas $\hbar$ is set to 1 when the system is in active mode. The mode switch between the active mode and the sleep mode can be achieved by applying gated supply voltage with negligible overhead in our scheduling scale. Moreover, C_{ef} and $2 \leq \gamma \leq 3$ are system-dependent constants for representing the effective switching capacitance and the dynamic power exponent, respectively. Due to the excessive time/energy overhead of turning on/off a system, the static power P_{sta} cannot be removed, and hence power consumption that is manageable by the system is $\hbar(P_{ind} + C_{ef} s^\gamma)$.

In the described setting, the energy consumption for execution that is manageable by the system is merely a convex function of the execution speed. There is a *critical speed* s_{crit}, e.g., $\sqrt[\gamma]{\frac{P_{ind}}{C_{ef}(\gamma-1)}}$ in [19], such that executing at s_{crit} is more energy-efficient than executing at any other (also lower) speed [9], [19]. We assume that the system can operate at any speed in the range of $[s_{\min}, s_{\max}]$. By the definition of the critical speed, only operating speeds in the range $[s^*_{\min}, s_{\max}]$ should be used, where $s^*_{\min} = \min\{\max\{s_{\min}, s_{crit}\}, s_{\max}\}$.

For systems with only discrete speeds, we can use voltage hopping [12] to achieve a spectrum of continuous speeds. Without loss of generality, we assume that $s_{\max}$ is normalized to 1, and all the other related metrics are scaled accordingly. We consider systems with negligible overhead to switch between sleep and active modes. More specifically, we assume that the system turns to the sleep mode when there is no event to process, and that it turns back to the active mode as soon as an event to be processed arrives. Note that, our method does not focus on a specific power model. We just require monotonicity and convexity of the energy consumption in the interval $[s^*_{\min}, s_{\max}]$.

C. Problem Definition

In this paper, we consider power-aware DVS scheduling for one event stream, in which an event arriving at time t must be finished no later than its *absolute deadline* $t + D$. The objective is to minimize the energy consumption for the processing of the stream while satisfying all the deadlines. For multiple event streams, our approach can be easily adopted, but requires longer verification time. Events are prioritized by their absolute deadlines with earliest-deadline-first (EDF) scheduling. For notational brevity, for any two events e_j and e_k, we use $e_j \preceq e_k$ to denote that event e_j has higher priority than or equal priority to e_k.

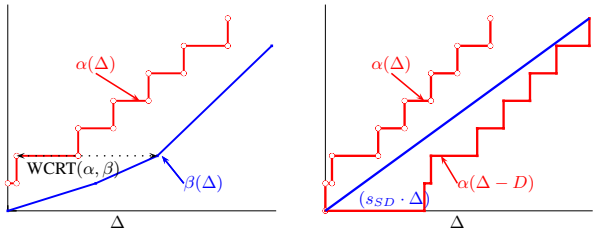
Moreover, we assume that processing an event at speed s takes C/s time units. Hence, the curve $\alpha(\Delta)/s$ denotes an upper bound on the execution time requested by the event stream in any time interval of size Δ under the assumption that the events are processed at speed s.

D. Pessimistic DVS Scheduling

To guarantee the timing constraints, one needs to prepare for the worst-case event arrivals, which occur in case of bursts. As the energy consumption is a convex and increasing function of the execution speed, the goal is to execute as slow as possible in the range $[s^*_{\min}, s_{\max}]$, see e.g. [18], [19]. Therefore, we can execute events at a *static speed* such that the timing constraints are satisfied. By using results from RTC [11], [14], the worst-case response time (WCRT) of the event stream is a function of α and β:

$$\text{WCRT}(\alpha, \beta) \equiv \sup_{\lambda \geq 0} \{\inf\{\tau \geq 0 : \alpha(\lambda) \leq \beta(\lambda + \tau)\}\} \tag{3}$$

$\text{WCRT}(\alpha, \beta)$ can be interpreted as the maximum horizontal distance between the curves α and β as shown in Figure 1a. The event stream is *schedulable* if $\text{WCRT}(\alpha, \beta) \leq D$, i.e. the worst-case response time



(a) WCRT (b) Pessimistic DVS scheduling

Fig. 1: Graphical interpretations for analysis based on Real-Time Calculus.

is not larger than the relative deadline. This can also be expressed as the inequality

$$\alpha(\Delta - D) \leq \beta(\Delta) \quad \forall \Delta \geq 0. \tag{4}$$

Based on the above analysis, the static DVS schedule with one constant speed is to find the minimum speed s_{SD} such that

$$\alpha(\Delta - D) \leq s_{SD} \cdot \Delta \quad \forall \Delta \geq 0, \tag{5}$$

and to execute at the constant speed $\max\{s^*_{\min}, s_{SD}\}$. For brevity, we denote this approach as Algorithm SD (stands for static DVS).

Therefore, executing events at a speed s_{SD} that satisfies (5) and by applying EDF, we can guarantee the satisfaction of the timing constraints. However, as the arrival curve bounds the worst-case event trace, the *actual* event arrivals might be far below the arrival curve in many intervals. For instance, it could be that a burst of event arrivals happens only once, but we have to always provide a high computation speed s_{SD} in order to guarantee the timing requirements. In other words, by a static speed assignment one tends to be too pessimistic and ignores the opportunity of reducing the energy consumption.

E. Optimistic On-Line DVS Algorithms

In contrast to a pessimistic scheduling scheme, an optimistic DVS scheduling algorithm takes a new scheduling decision each time when an event *really* arrives. As most systems do not have event bursts all the time, the on-line algorithms can help to reduce the energy consumption. Different on-line DVS algorithms have been proposed in the literature [3], [18]. The *Algorithm OPT* proposed by Yao, Demers, and Shenker [18] is known to be the best on-line competitive algorithm for the minimization of the energy consumption.

For an event e_j that is incomplete at time t, suppose that $C_j(t)$ is its worst-case remaining execution time at time t and speed $s_{\max}$, a_j is its arrival time, and d_j is its absolute deadline. Algorithm OPT makes the scheduling decision at time t by executing the highest-priority event at speed

$$s(t) = \max\left\{ s^*_{\min}, \max_{e_j}\left\{ \sum_{e_i : a_i \leq t, e_i \preceq e_j} \frac{C_i(t)}{d_j - t} \right\} \right\}. \tag{6}$$

In other words, Algorithm OPT selects the execution speed on-line in a greedy fashion and guarantees a low energy consumption by only considering those events that have arrived so far and are not yet completely processed. Note that if speed switching requires timing overhead bounded by χ, we just have to modify $d_j - t$ in (6) to $d_j - t - \chi$, and introduce one additional state in the TA discussed in Section IV.

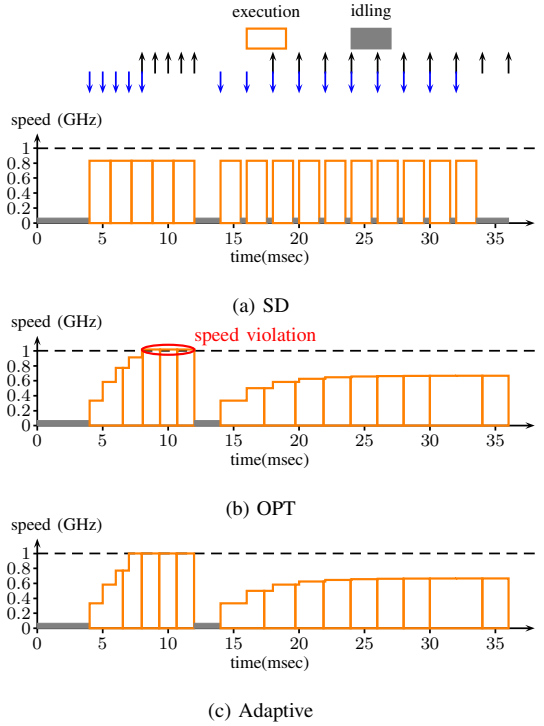


Fig. 2: An example for different scheduling algorithms.

When EDF is applied, Algorithm OPT has a competitive factor of γ^γ, compared to optimal offline DVS scheduling with known event arrival times [2]. Note that Algorithm OPT can always derive feasible schedules to meet the timing constraints of the real-time events, if the system has no maximum speed constraint (i.e., $s_{\max} = \infty$). However, if $s_{\max}$ is constrained, applying Algorithm OPT might lead to a schedule that violates the timing constraints since the speed required at some time instant t might be larger than $s_{\max}$.

III. Adaptive Scheduling Scheme

As suggested by Section II-E, before applying Algorithm OPT, we have to ascertain that it is *feasible* in the particular system. That is, we have to verify that in all possible event traces constrained by the arrival curve the events can meet their deadlines without violating $s_{\max}$. The feasibility test of Algorithm OPT for one event stream has been studied by Chen et al. [6]. If Algorithm OPT is not feasible, we can fall back to the pessimistic DVS scheduling by running at speed s_{SD}, as suggested in [6]. But this fall-back might be too harsh since Algorithm OPT is possibly applicable as long as the system is *light-loaded*. In order to prevent deadline violations, we have to be pessimistic when the system is *heavy-loaded* only. This section presents a motivational example for an adaptive DVS scheduling scheme, followed by the definition of such a scheme.

A. Motivational Example

Consider an upper arrival curve specified as in (2) with $p = 2$ msec, $J = 4$ msec, and $d = 1$ msec. For simplicity, let the power consumption function be $P(s) = \hbar(\frac{s}{1GHz})^3$ Watt, where $s_{\min}$ is assumed 0 and the maximum speed is 1 GHz. The execution time at speed $s_{\max}$ is $\frac{4}{3}$ msec and the relative deadline D is 4 msec.

Suppose that we release 15 events bounded by the arrival curve at times $(4, 5, 6, 7, 8, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32)$ msec. By

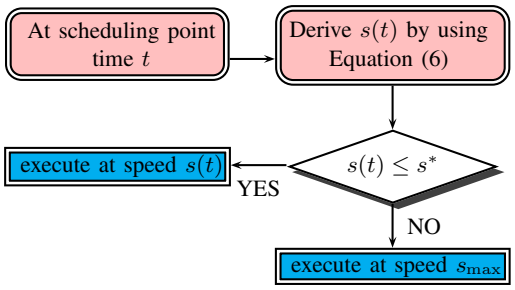


Fig. 3: Flowchart of Adaptive DVS Scheduling Scheme.

applying (5), for the given arrival curve, we know that speed s_{SD} is $\frac{5}{6}$ GHz. The resulting schedule for the static speed assignment for this input trace is shown in Figure 2a with 13.89 mJoule energy consumption. Applying Algorithm OPT leads to a schedule in Figure 2b with 10.91 mJoule energy consumption, but the required speed from in time interval $[8, 12]$ msec is 1.017 GHz, which slightly violates the maximal speed. If we adapt Algorithm OPT by greedily running at the maximum speed $s_{\max}$ when $s(t)$ derived from (6) is more than 0.85 GHz, we can switch to speed $s_{\max}$ at time 7 msec and have a feasible schedule as shown in Figure 2c with energy consumption 10.92 mJoule, which is an energy reduction of 21%, compared to running at speed s_{SD}.

B. Definition of the Adaptive Scheme

To achieve energy saving while satisfying the timing constraints, we propose the adaptive DVS scheduling scheme shown in Figure 3. Specifically, at a scheduling point at time t, we first estimate the speed $s(t)$ by applying Algorithm OPT, as shown in (6). If $s(t)$ is less than or equal to a speed threshold s^*, we greedily execute the event with the highest priority by using speed $s(t)$. Otherwise, we execute at speed $s_{\max}$ to resolve the burst of event arrivals. We say that an adaptive DVS scheduling scheme is *feasible* with speed s^* if the timing constraints of the events are satisfied for all traces constrained by the arrival curve. As we would like to be as optimistic as possible to reduce the energy consumption, the objective is to derive the *maximal* speed s^* such that the adaptive DVS scheduling scheme is feasible with s^*.

Algorithm OPT is an *event-driven* scheduling algorithm, which means that the instant when a job arrives or completes is a scheduling point for speed determination. Note that the speed computation in (6) requires precise knowledge of the amount of completed processing demand at the given time t, as well as the time left until the deadline of the individual events. Taking into account these quantities is, however, not trivial in a state-based model. This requires not only that the modeling formalism has a notion of continuous time, but also that computations on time variables can be performed. For instance, in Uppaal [4], [5], the verification tool based on Timed Automata (TA) that we employ in this paper, computations on clock variables are not supported. This means that the event-driven scheduling policies that are based on elapsed/remaining time such as OPT or EDF can only be approximately represented by means of TA models. Moreover, coming up with a conservative TA approximation for the described adaptive scheduling scheme based on Algorithm OPT is not trivial. In particular, any approximation that overestimates the actual speed $s(t)$ selected by the scheme at time t is not safe as it might ignore deadline violations. But at the same time, any approximation that underestimates the actual speed $s(t)$ is not safe as it might result in premature changes to $s_{\max}$ compared to the actual system, and hence again jeopardize the verification of deadlines.

Therefore, in order to avoid ambiguous results for the safety of an actual system architecture, in this paper we devise a formal model based on TA that restricts the scheduling scheme to *time-driven* scheduling. More precisely, we discretize time by introducing artificial clock ticks with period T. These ticks are counted in order to keep track of elapsed/remaining computation times. In our model, an event that arrives between two clock ticks will be buffered, and will affect the system only at the following tick. That is, an event that arrives at time t' will be released to the scheduler at time $\left\lceil \frac{t'}{T} \right\rceil T$ and we also anticipate its deadline from $t'+D$ to $\left\lfloor \frac{t'+D}{T} \right\rfloor T$. The resulting algorithm is an adaptive and time-driven variant of the original OPT algorithm.

IV. Analysis of Adaptive Scheme

The proposed DVS scheduler is state-based, as it chooses the processing speed depending on the actual history of event arrivals. For this reason, the traditional RTC analysis for state-less components cannot be applied to examine its behavior and derive the optimal threshold speed s^*. As shown in [6], it is also not trivial to employ RTC for analyzing the maximum speed that Algorithm OPT might experience. This section presents a method for the analysis of the proposed DVS scheme. The method uses RTC curves for abstractly describing event streams (task activation patterns) and exploits TA [1] for modeling the DVS scheduler. It is based on the hybrid analysis approach introduced in [10]. Figure 4 depicts our analysis framework. It consists of the following components:

1) *Simplified arrival curves*
   For reducing the complexity of the TA-based analysis framework we use conservatively simplified event arrival curves. The simplification is an over-approximation which converts a general curve $\bar{\alpha}$ into a staircase function $\bar{\alpha}'$ with non-decreasing step-widths.
2) *Event generator*
   We automatically derive a network of TA which produces event streams that are bounded by the simplified staircase curve $\bar{\alpha}'$. This event generator, which we denote $\mathcal{G}(\bar{\alpha}')$, is fully equivalent to $\bar{\alpha}'$ in the sense that it is able to produce *all* event traces that are constrained by $\bar{\alpha}'$.
3) *TA-based abstraction of the DVS-scheduler*
   We use a TA-based model of the adaptive DVS scheduling algorithm in which we employ clock discretization in order to keep track of the completed/remaing processing demands of events. The interaction among the event generator and the state-based DVS scheduler is modeled by means of a shared variable, which represents the number of buffered input events. We employ the timed model checker Uppaal [4] in order to derive the maximum threshold speed s^*. In particular, we follow a binary search strategy in which the timing requirements of the input event stream are verified for different values of s^*. If for a given value of s^* Uppaal reports a deadline violation, we have to be more conservative and reduce s^*. In contrast, if Uppaal confirms that for the given s^* all events meat their deadline, we try to use a more optimistic threshold speed. Note that the model checker asserts the schedulability for *all* timed event traces bounded by $\bar{\alpha}'$.

It is important to note that the analysis framework described above, which is based on expensive formal verification of TA models, is used at design time only. In particular, it is employed offline to parameterize and validate the adaptive DVS scheduler for a given

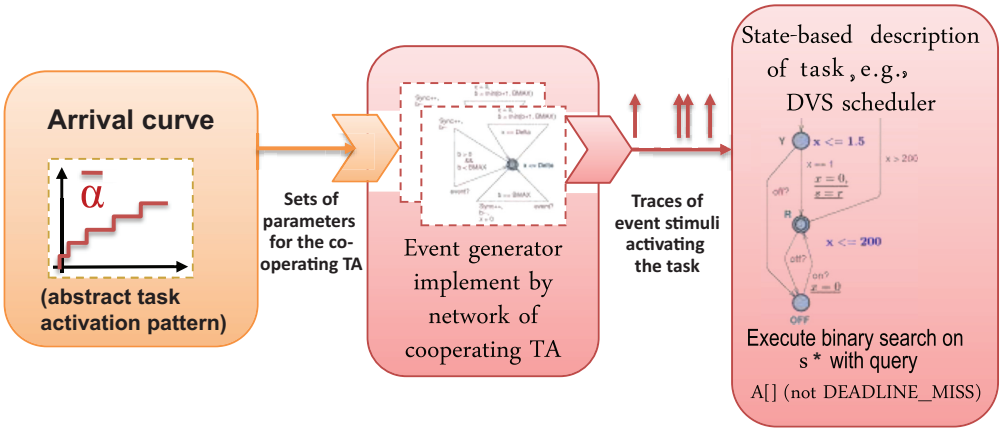


Fig. 4: Framework for analyzing adaptive DVS scheduling schemes

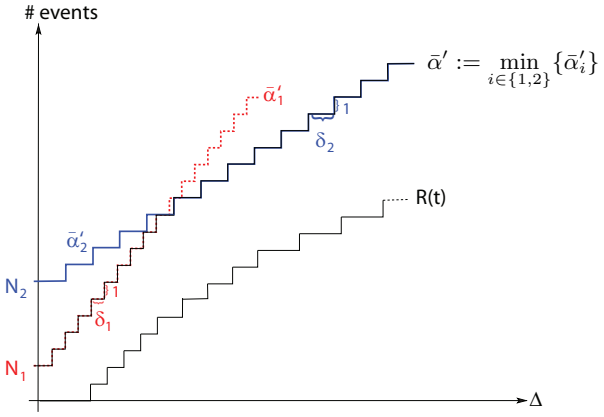
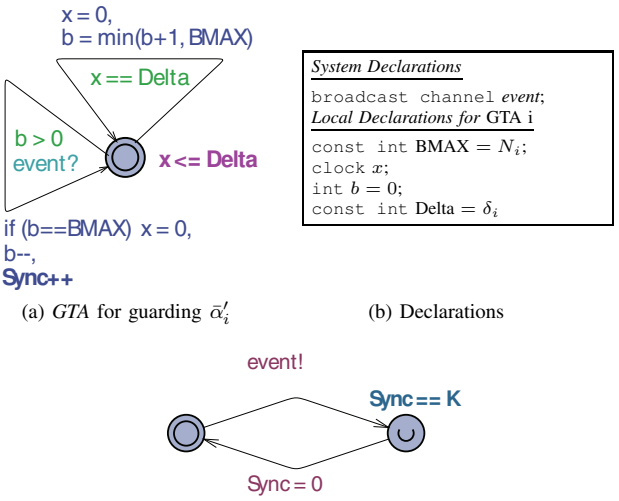


Fig. 5: Stair-case arrival curve



(a) *GTA* for guarding $\bar{\alpha}'_i$ (b) Declarations

(c) TA for enforcing full synchronization

Fig. 6: Event generator $\mathcal{G}(\bar{\alpha}')$ for a concave arrival curve $\bar{\alpha}'$.

(potentially infinite) set of real-time event streams. In the proposed scheme there is, however, no additional overhead for the online DVS algorithm itself, which is just a simple variant of Algorithm OPT.

In the following we detail on the parts of the analysis framework. Due to space limitation, we will implicitly use the syntax and semantic of Uppaal for describing our TA-based approach. For details about Uppaal, please refer to [4], [5].

A. Conservative Simplification of Input Curves

A general arrival curve $\bar{\alpha}$ can by approximated by a set of simple staircase functions $\bar{\alpha}'_i$ composed by minimum operation.

$$\begin{aligned} \bar{\alpha}'(\Delta) &:= \min_i\{\bar{\alpha}'_i(\Delta)\} \text{ with} \\ \bar{\alpha}'_i(\Delta) &:= N_i + \left\lfloor \frac{\Delta}{\delta_i} \right\rfloor . \end{aligned} \tag{7}$$

Figure 5 shows an example of a staircase function $\bar{\alpha}'$ that is obtained by the minimum of two simple staircase functions $\bar{\alpha}'_1$ and $\bar{\alpha}'_2$. Each simple staircase function $\bar{\alpha}'_i$ is defined by means of two parameters N_i and δ_i which represent the maximum burst and the maximum long-term rate of the considered event stream, respectively. Note that a staircase curve $\bar{\alpha}'$ in (7) has non-decreasing step widths, which we denote as pseudo-concaveness of the arrival curve. The approximation of an arrival curve is safe as long as $\bar{\alpha} \leq \bar{\alpha}'$ holds for all $\Delta \in [0, \infty)$, where $\bar{\alpha}$ denotes the original arrival curve and $\bar{\alpha}'$ is the approximated one. Intuitively speaking, for the sake of efficiency we test the DVS scheduler with more activation patterns than necessary, which does not harm the safety of the analysis.

B. TA-Based Modeling of Arrival Curves

The proposed modelling pattern for arrival curves follows the ideas of [10]. A pseudo-concave arrival curve $\bar{\alpha}'$ is implemented by an event generator $\mathcal{G}(\bar{\alpha}')$, which is a set of cooperating TA that produces all event traces bounded by $\bar{\alpha}'$. Figure 6 illustrates the basic components of the TA network. For each simple staircase function $\bar{\alpha}'_i$ defined in (7), an instance of the TA shown in Figure 6(a) is used. This TA, denoted as *GTA*, constrains event emissions such that the event trace conforms to $\bar{\alpha}'_i$. The event emissions are triggered by the TA of Figure 6(c). The synchronization with the different *GTA* enforces that the event generator can produce only timed event traces that are bounded by $\bar{\alpha}'$. In particular, the minimum operation of (7) is implemented by *full synchronization* of all *GTA*. Within Uppaal this behavior can be implemented by making use of a broadcast channel and a global variable $SYNC$ as depicted in Figure 6. In the TA of Figure 6(c) the constant K corresponds to the number of employed *GTA*. Note that $\mathcal{G}(\bar{\alpha}')$ allows to produce *all* event traces that are constrained by $\bar{\alpha}'$. The corresponding proof is provided in [10].

An example is provided by Figures 5 and 6. For translating the arrival curve $\bar{\alpha}'$ of Figure 5, one needs 2 *GTA* instantiated with $BMAX := N_{\{1,2\}}$ and $Delta := \delta_{\{1,2\}}$. In the event emitter (Figure 6(c)) the constant K is set to 2 such that events can be generated only if both $\bar{\alpha}'_1$ and $\bar{\alpha}'_2$ are respected.

Note that the described model for representing arrival curves by means of TA is very general in the sense that it allows to combine an arbitrary number of simple staircase functions $\bar{\alpha}'_1$ in order obtain an accurate approximation of an arrival curve. There are, however, also more specific models that are tailored to particular types of arrival curves. For instance, in [8] a TA model for periodic event streams with jitter is presented. While such dedicated models have a very restricted application scope, in terms of verification they are at times more efficient than the described general representation.

C. Analysis Method

In this work we consider TA extended with integer data types, as employed by the model checker Uppaal [4]. Uppaal adopts a continuous representation of time (clocks are real-valued), however it does not permit arithmetic operations on clock values. Hence, the behavior of the event-driven OPT Algorithm as described by (6) cannot be directly translated to TA. In order to circumvent this problem, in the following we introduce a TA model for a time-driven variant of OPT that makes use of a discrete time representation.

The TA model for a CPU processing one event stream and implementing the proposed scheduling scheme is shown in Figure 7. The figure depicts a network of TA that cooperate by means of channels and global variables. The model assumes that all the events of the input stream have a constant execution demand *C* expressed in processing units, and a constant relative deadline *D* expressed in time units, where both *C* and *D* are integers. The model relies on an explicit representation of time, where the advancing of a clock is tracked by periodic clock ticks. In particular, automaton (a) broadcasts a signal *tick* every *T* time units (clock period). Automaton (b) handles the arrival of new events, as it synchronizes with the event generator over the broadcast signal $event$. Every time a new event arrives, the automaton increments the counter variable *b* which represents the current number of buffered events that need to be processed. Note that event arrivals are independent of clock ticks, meaning that new events can arrive at any point in time. However, in the proposed model an event arrival will not affect the CPU until the next clock tick, when the automaton will move to the *BUSY* location if the CPU was idle or compute a new processing speed when it was busy. The processing speed is represented by a global integer variable *s*. The value of *s* represents the number of processing units that the CPU provides per time unit. The model uses an array of integers *deadlines* to keep track of the number of time units remaining for each individual event to its absolute deadline. At an event arrival the corresponding position in the array is initialized with *D*, see 7(b). At every clock tick, all elements in the deadline array are updated, that is, for each buffered event the counter of time units remaining to the deadline is decreased by *T*. Note that at each clock tick the deadline array is updated *before* the new processing speed is computed. This makes sure that a sufficiently high speed will be chosen, such that the deadline of an event is met, even though it might have been released with delay, i.e. only at the next clock tick following its actual arrival time.

Automaton 7(c) models the CPU itself. It manipulates a global integer variable *rct* that represents the remaining execution demand, expressed in processing units, for the currently processed event. At each clock tick *rct* is decreased according to the selected processing speed. If the deadline of an event is contained within the next clock tick but the processing cannot be finished in time, the automaton immediately reports a deadline violation by moving to an appropriate state.[1] On the other hand, if the remaining execution demand for an event is less or equal than the processing service available in a clock period, the location *BUSY* is immediately left and one of the following three cases applies. (1) The number of time units left to the deadline is not sufficient to complete the processing of the event and, hence, a deadline violation is reported. (2) The completion of the event processing happens in time and there are no other events in the queue. Hence, the transition to the location *IDLE* is taken. (3) The completion of the event processing happens in time and there are other events in the queue. Hence the direct transition to the location *BUSY* is taken. In both cases (2) and (3) the elements of the array *deadlines* are shifted by one position and the variable *b* is decremented.

Algorithm 1 `Compute speed`

```
1: function SPEED
2:     int s ← 0,  int s'
3:     for i ← 0, b − 1 do
4:         s' = (rct + i * C)/deadlines[i]
5:         s ← max(s, s')
6:     end for
7:     if s > s* then
8:         s ← s_max
9:     end if
10:    return s
11: end function
```

The function adopted in the model to compute the processing speed is reported in Algorithm 1. As can be seen, it implements the described adaptive scheduling scheme by setting the processing speed to s_{max} if a speed above the threshold s^* is requested.

By means of the model checker Uppaal we can now verify whether an event stream is schedulable by the proposed adaptive scheduling scheme with threshold speed s^*. The corresponding query is specified as follows:[2]

```
A[] (not CPU.DEADLINE_MISS)
```

By performing a simple binary search on s^* it is possible to determine the *maximal* threshold speed that permits to guarantee the timing constraints.

Note that the discrete time approximation of the scheduling scheme can be made arbitrarily precise by reducing the length of the clock period *T*. However, a more fine-grained time representation will result in longer verification times. Hence, a system designer adopting the proposed approximated model can trade analysis accuracy for verification time.

D. Generalization to Multiple Event Streams

In this section we briefly explain how the TA model introduced above can be extended to the case of multiple input event streams. In particular, we sketch a model of a CPU that processes the events of several input streams according to the Earliest Deadline First (EDF) scheduling policy. For the sake of conciseness, we do not report the entire TA model, but only point out the differences to the model of Section IV-C.

We assume that the CPU processes n input event streams with execution demands C_i and relative deadlines D_i, $i \in n$. The extended model still uses a single counter variable b and a single array

[1] Immediate reaction is guaranteed by the urgent channel *hurry* which is always ready to synchronize.

[2] In Uppaal `A[]` stands for 'always invariantly'.

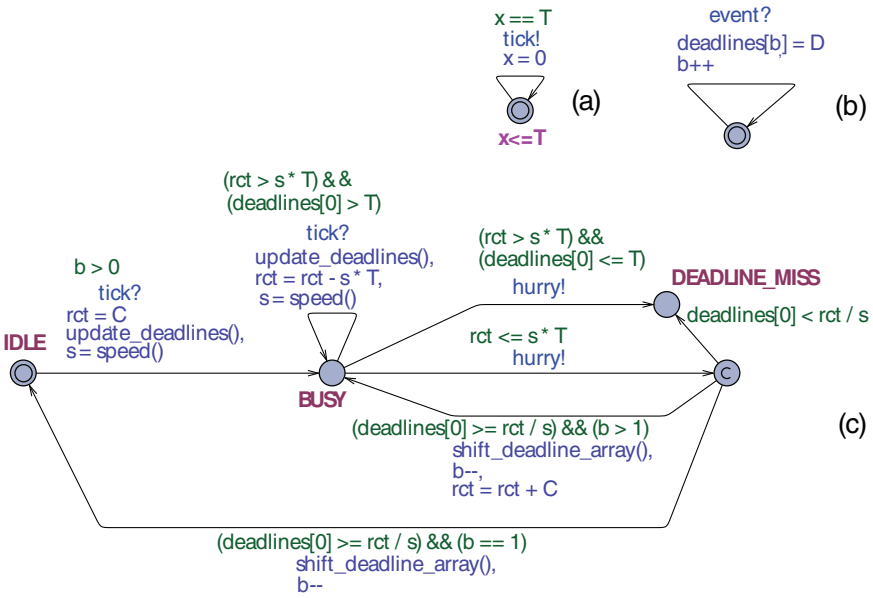


Fig. 7: TA model of the discretized adaptive scheduling scheme

deadlines to handle all the input events now originating from different input streams. However, unlike in the previous case, the last arrived event does not necessarily have the farthest absolute deadline. Hence, we have to explicitly keep the elements in the array *deadlines* sorted, meaning that new elements have to be inserted at the right position. Moreover, under EDF scheduling it can happen that a currently processed event is preempted by a newly arrived event with earlier deadline. In our discrete time representation of the CPU, this case is conservatively approximated by newly selecting the next event to process at each clock tick. Preemptions also imply that we have to explicitly store the remaining execution demand for each buffered event. This is done by replacing the variable *rct* by an appropriate array of integers.

While in principle the described modeling method permits to consider an arbitrary number of input streams, it is clear that the complexity of the verification will increase drastically with the number of considered input streams.

V. Performance Evaluation

In this section, we evaluate the performance of Algorithm SD, Algorithm OPT, and the proposed adaptive algorithm (denoted AD) in terms of energy usage.

Experimental Setup: For comparison of the performance of the three algorithms, we use a set of 6 different event streams adapted from [16]. The considered streams are periodic event streams with jitter that are specified by the following parameters: period p, jitter J, minimum time between consecutive event arrivals d, worst-case execution time C (at $s_{\max}$), and relative deadline D. Table I summarizes the parameter values for the 6 event streams. The streams have been selected such that if they are scheduled with Algorithm OPT, they violate the maximum system speed which is 0.5GHz. In other words, there will be deadline violations if Algorithm OPT is constrained to $s_{\max} = 0.5$GHz. The maximum speeds required by Algorithm OPT for the different streams are calculated as proposed in [6]. In particular, we use approximative traces of length $3 \cdot D$ ms (where D is the relative deadline of a stream), which was shown to be a tight approximation. The speeds required by Algorithm SD are calculated with (5). The corresponding speeds are shown in Table II (first two rows).

	I	II	III	IV	V	VI
p	198	102	283	239	148	114
J	387	70	269	222	91	13
d	48	45	58	65	78	0
C	30	35	77	69	53	52
D	110	140	310	280	200	120

TABLE I: Parameters for the 8 input event streams in [ms].

For Algorithm AD, the speed thresholds s^* have been computed with the model checker Uppaal based on the TA models described in Sec. IV and using a discretization granularity of $T = 2$ ms. Note that in order to improve the efficiency of the verification, we take advantage of the periodicity of the considered streams by using the dedicated event generator described in [8] to trigger the TA-based adaptive DVS model. The resulting threshold speeds are shown in Table II (last row). The run-times for computing the speed thresholds for Algorithm AD on a 64-bit Sun Fire X2200 M2 with 8GB RAM are shown in Table III.

	I	II	III	IV	V	VI
s_{SD}	0.44	0.38	0.42	0.4	0.39	0.47
s_{OPT}^{max}	0.513	0.505	0.501	0.506	0.506	0.506
s^* (T=2ms)	0.38	0.36	0.29	0.39	0.37	0.39

TABLE II: Maximum speeds for algorithms SD and OPT (first two rows), threshold speeds s^* determined for Algorithm AD, assuming $s_{max} = 0.5$ (last row) in [GHz].

In order to evaluate the energy-efficiency of the different algorithms, we first use the RTC Toolbox [15] to produce 10 random event traces for each of the streams. These traces are generated so that they are as close to the arrival curve as possible, each with a length of 20000 ms. The execution of these traces is then simulated

	I	II	III	IV	V	VI
T=2ms	210	262	16679	2973	459	2

TABLE III: Run-times for computing the threshold speeds s^* for clock period value $T = 2$ ms in [sec].

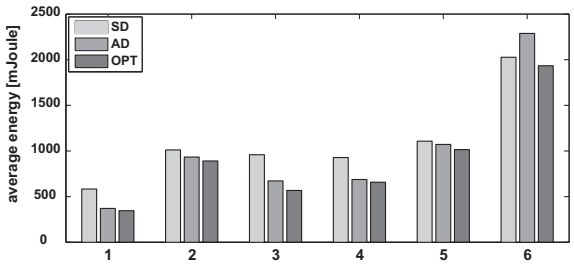


Fig. 8: Average energy required by the evaluated algorithms to process the traces of each stream.

by means of a simple discrete event simulator that implements the described scheduling schemes and monitors the energy consumption of the system. The simulator adopts the power consumption function $P(s) = 0.04 + \hbar\big(1.56(\frac{s}{0.5GHz})^3\big)$Watt, which is an approximation for the power consumption of the Intel XScale architecture. Finally, we calculate the average energy needed by each algorithm for scheduling the different traces from a particular stream.

Results: Figure 8 shows the average energy required by the algorithms to process the traces of each event stream. Since for the considered streams it is not feasible to use Algorithm OPT on a system with a maximum speed of 0.5GHz, the average energy is given only as a reference to see how pessimistic Algorithm AD is. As can be observed in the figure, Algorithm AD is not much worse than Algorithm OPT, on average 10%. For five of the streams, it performs better than Algorithm SD with 22% on average. Note the results for the last stream that show that Algorithm SD is not always worse than Algorithm AD. The reason for this result is that stream VI has a very small jitter, i.e., is almost a periodic event stream and hence executing at a constant speed is more energy-efficient than adapting the execution speed.

VI. Conclusion

This paper explores an adaptive DVS scheduling scheme that decides the adoption of pessimistic and optimistic DVS scheduling dynamically. Specifically, the scheme tries to be as optimistic as possible when the system is light-loaded in order to reduce the energy consumption. However, when the required processing speed reaches some threshold, the scheme becomes pessimistic in order to ensure the observation of deadlines. The presented approach models event streams by means of arrival curves, an abstraction used in Real-Time Calculus (RTC), and exploits Timed Automata with clock discretization for verifying whether a given threshold speed is safe in terms of timing guarantees. In particular, the framework applies a binary search to find the maximal threshold such that the scheduling scheme is as optimistic as possible. The experiments confirm the beneficial properties of the adaptive DVS scheme.

Note that the proposed analysis method can be easily extended to systems with discrete speeds, without falling back to voltage hopping and (expensive) speed-scaling. One just needs to modify the speed selection in (6) and the models in Figure 7 accordingly. As the restriction to discrete speeds reduces the search space, lower verification times are expected. Moreover, we considered that the scheme runs at speed $s_{\max}$ in the pessimistic mode, but any other speed can be used instead. However, how to best choose a suitable speed for the pessimistic mode in terms of energy is unclear. Further, note that even though we focussed most of our discussions on systems with one event stream, the approach can be easily extended to multiple streams. However, when scaling TA models one often encounters state space explosion. Hence, we expect the verification time to increase drastically for systems with multiple inputs.

Acknowledgment

This work is funded by the European Union projects PREDATOR and COMBEST under grants number 216008 and 215543, respectively, as well as by the Swiss National Science Foundation under grant number 200020-116594.

References

[1] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. In M. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, pages 322–335, 1990.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 520–529, 2004.

[3] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *STACS*, pages 460–471, 2005.

[4] 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: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, pages 200–236, 2004.

[5] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124, 2004.

[6] J.-J. Chen, N. Stoimenov, and L. Thiele. Feasibility analysis of on-line dvs algorithms for scheduling arbitrary event streams. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 261–270, 2009.

[7] V. Devadas and H. Aydin. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *EMSOFT*, pages 99–108, 2008.

[8] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *IPDPS*, 2006.

[9] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*, pages 275–280, 2004.

[10] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems. In *International Conference on Embedded Software (EMSOFT)*, pages 127–136, 2009.

[11] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[12] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *DAC*, pages 806–809, 2000.

[13] A. Maxiaguine, S. Chakraborty, and L. Thiele. Dvs for buffer-constrained architectures with predictable qos-energy tradeoffs. In *CODES+ISSS*, pages 111–116, 2005.

[14] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. *ISCAS*, 4:101–104, 2000.

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

[16] E. Wandeler and L. Thiele. Optimal tdma time slot and cycle length allocation for hard real-time systems. In *ASP-DAC*, pages 479–484, 2006.

[17] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis - a case study. *Software Tools for Technology Transfer (STTT)*, 8(6):649 – 667, Oct. 2006.

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

[19] D. Zhu, R. G. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *ICCAD*, pages 35–40, 2004.