Feasibility Analysis of On-Line DVS Algorithms for Scheduling Arbitrary Event Streams

Jian-Jia Chen, Nikolay Stoimenov, Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology, ETH Zurich
8092 Zurich, Switzerland
Email: {jchen, stoimenov, thiele}@tik.ee.ethz.ch

***Abstract*—Performance boosting of modern computing systems has been constrained by the significant chip/circuit power dissipation. Dynamic voltage scaling (DVS) has been applied in the past decade for reducing the energy consumption by dynamically changing the supply voltage. On-line scheduling algorithms for DVS systems usually guarantee the real-time constraints of the system based on the condition that they can select any system speed that is sufficiently high to allow processing of all events within their deadlines. However, practical systems have a maximum available system speed and the feasibility of using on-line DVS algorithms needs to be verified during design time, i.e., they will never require during runtime a speed higher than the maximum available. This paper presents feasibility analysis of two on-line DVS algorithms that can compute in advance an upper bound on the system speed that these algorithms may require given that there is a single input event stream described by the worst-case event arrivals in interval domain. Moreover, we also present new results on the competitive ratios of the resulting schedules for energy consumption minimization with comparison to the off-line optimal solutions to show the effectiveness of the two algorithms. At the end, the performance of the different algorithms is evaluated.**

***Keywords*-applicability and schedulability analysis; arbitrary event streams; energy-efficient scheduling; on-line DVS; real-time calculus; real-time systems**

I. Introduction

Power dissipation has been considered as one of the most important design issues for modern computing systems in both hardware and software aspects. The reduction of power consumption is not only useful for embedded systems to prolong battery lifetimes but also important for server systems to cut power bills. As shown in the literature, e.g., [14], the power consumption of modern CMOS circuits mainly comes from dynamic power consumption due to switching activities and leakage power consumption due to the leakage current. The dynamic power consumption can be reduced by applying dynamic voltage scaling (DVS), while the leakage power consumption can be reduced by changing the system mode dynamically with dynamic power management (DPM).

Specifically, the dynamic voltage scaling (DVS) technique was introduced to reduce the dynamic energy consumption by trading performance for energy savings. For DVS processors, a higher supply voltage, generally, leads not only to a higher execution speed but also to higher power consumption. As a result, DVS scheduling algorithms, e.g., [2], [22], [23], tend to execute events as slowly as possible, without any violation of timing constraints. On the other hand, to consume less leakage 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. It has been shown that there is a critical speed, such that executing at any speed lower than the critical one consumes more energy than at the critical speed [8], [14], [24].

Most studies for energy-efficient scheduling in real-time systems assume that input tasks are periodic or sporadic. For example, periodic real-time events are assumed in [7], [8], [12]–[14], [16], while aperiodic real-time events with known arrival times are assumed in [1], [6]. However, in practice, the precise information of event arrival times might not be known in advance since the arrival times depend on many factors. More complex input event streams can be characterized by arrival curves as used in Real-Time Calculus [20] and Network Calculus [9], [15]. Real-time events are characterized by evaluating how often system functions will be called, how much data is provided as input to the system, and how much data is generated by the system back to its environment. As a result, arrival curves constrain the number of events arriving in the system for a specified interval. The system designer can then perform schedulability analysis based on these curves.

However, even though each trace of events is constrained by the arrival curve, the curve itself does not reveal how events *actually* arrive at the system. It might be possible that (1) bursts of event arrivals only happen rarely, or (2) there is no burst at all. However, relying only on off-line static analysis, we have to always provide higher computation power to supply the expected demand for the worst cases. Therefore, applying scheduling decisions based only on the arrival curve could be very pessimistic. For example, if

The work presented in this paper was partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322, and the European Commission's Seventh Framework Programme FP7/2007-2013 project Predator under grant number 216008.

we find the minimum *constant* speed to schedule an event stream as proposed in [18], it is possible that most of the time we run at an unnecessarily high speed because our scheduling decision would be based entirely on the worst-case event arrivals that may not happen often. Instead, we would like to apply on-line DVS scheduling algorithms that make scheduling decisions only when events *really* arrive. As most systems do not have event bursts all the time, the on-line algorithms can help reduce the energy consumption.

Recently, Real-Time Calculus has been adopted by Huang et al. [10], [11] for exploiting DPM to turn off devices dynamically and Maxiaguine et al. [18] for DVS systems. Specifically, in [11], off-line algorithms are proposed to derive periodic power management schemes for energy reduction. In [10], on-line algorithms are developed to find the adaptation points which vary according to the actual event arrivals. There is a class of on-line scheduling algorithms that make predictions about the future arrivals of events, for example see [18]. They make scheduling decisions based on the history of already arrived events and the predicted possible arrivals in the future. However, all the above algorithms are different from the ones considered in this paper, in which we do not make any predictions about the future.

Specifically, in this paper, we explore two existing on-line DVS algorithms: Algorithm AVR and Algorithm OPT [4], [22]. Algorithm AVR executes events at the accumulated average speed of tasks, whereas Algorithm OPT greedily makes on-line speed scheduling by only considering those events that have arrived at the system so far and schedule them with the optimal energy consumption. Suppose that the power consumption of the system at speed s is proportional to s^γ, where γ is a hardware constant. As shown in [4], [22], Algorithm AVR (Algorithm OPT, respectively) has a $2^{\gamma-1}\gamma^\gamma$-competitive ratio ($\gamma^\gamma$-competitive ratio), in which a ρ-competitive ratio of an algorithm derives solutions at most ρ times of the energy consumption of the optimal off-line solutions.

With bounded speed systems, there have been some researches for on-line DVS scheduling for maximizing the throughput and optimizing the trade-off between the total flow time incurred and the energy consumed by jobs, e.g., [3]. However, these algorithms cannot be adopted for systems with hard real-time constraints. To the best of our knowledge, this is the first paper that deals with feasibility analysis of on-line DVS scheduling algorithms for an arbitrary event stream in hard real-time systems.

Contributions. By considering systems with one event stream that is constrained by an arrival curve, this paper presents how to analyze whether Algorithm AVR and Algorithm OPT are able to serve events within a specified range of available speeds. On-line DVS scheduling algorithms guarantee the real-time constraints of a system assuming that they can select any system speed that is sufficiently high to allow all events to be processed within their deadlines. Therefore, we would like to verify in advance that such an algorithm will never require a speed that is higher than the maximum system speed and real-time constraints can always be met. If the maximum execution speeds of all resulting schedules for an event stream do not violate the hardware speed constraint, we know that the on-line DVS algorithm is applicable for scheduling; otherwise, we have to pessimistically run at a constant speed without violating the timing constraints but being a less energy-efficient. Here, we propose an analysis framework for algorithms AVR and OPT that can compute upper bounds on the maximum speeds that these algorithms may require for scheduling all possible traces of an event stream that is characterized with an arrival curve. Moreover, for scheduling one event stream, we also show the 2^γ-competitive ratio of the resulting schedules for algorithms AVR and OPT. For example, if $\gamma = 3$, the competitive ratio is significantly reduced from 27 to 8 for Algorithm OPT and from 108 to 8 for Algorithm AVR.

The rest of this paper is organized as follows: Section II describes the system models used in the paper. Section III presents the algorithms considered in this paper and the proposed scheduling framework. Section IV introduces a motivational example. Section V presents the feasibility analysis of algorithms AVR and OPT. The competitive analysis with respect to energy minimization is shown in Section VI. The performance of different scheduling algorithms is evaluated in Section VII. We conclude this paper in Section VIII.

II. System Models

A. Hardware Model

This work considers the system-level power model in [24], where the power consumption at speed s is as follows:

$$P(s) = P_{sta} + \hbar(P_{ind} + P_d) = P_{sta} + \hbar(P_{ind} + C_{ef}s^\gamma), \quad (1)$$

where P_{sta}, P_{ind}, and P_d are *static power*, *speed-independent* active power, 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. 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. However, due to the excessive time/energy overhead of turning on/off a system, we consider systems that cannot be turned off dynamically, and, hence, the static power P_{sta} cannot be removed. Therefore, P_{sta} is not manageable, and we focus on how to manage the speed-independent and speed-dependent active power, i.e., P_{ind} and P_d. And more specifically, here we analyze algorithms that can control s online given information about the currently unprocessed events in the system. When there are no events in the system, we switch to sleep mode, i.e., $\hbar = 0$.

There is a *critical speed* $s_{crit} = \sqrt[\gamma]{\frac{P_{ind}}{C_{ef}(\gamma-1)}}$ such that executing at critical speed s_{crit} is more energy-efficient than executing at a speed lower than s_{crit} [14], [24]. 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 from $[s^*_{\min}, s_{\max}]$ should be used, where

$s^*_{\min} = \min\{\max\{s_{\min}, s_{crit}\}, s_{\max}\}$. We assume that the mode switch between the active mode and the sleep mode can be achieved by applying gated supply voltage with negligible overhead [24]. Without loss of generality, we will assume that $s_{\max}$ is normalized to 1, and all the other related metrics are also normalized.

B. Event Model

This paper focuses on events that can arrive in the system irregularly. To model such events, we use the concept of arrival curves successfully used in Network and Real-Time Calculus [9], [15], [20]. Specifically, a trace of an event stream can conveniently be described by means of a cumulative function $\bar{R}(t)$, defined as the number of events seen on the event stream in the time interval $[0, t)$. While any $\bar{R}$ always describes one concrete trace of an event stream, a tuple $\bar{\alpha}(\Delta) = [\bar{\alpha}^u(\Delta), \bar{\alpha}^l(\Delta)]$ of upper and lower arrival curves provides an abstract event stream model, providing bounds on admissible traces of an event stream. The upper arrival curve $\bar{\alpha}^u(\Delta)$ provides an upper bound on the number of events that are seen in any time interval of length $\Delta \geq 0$, while the lower arrival curve $\bar{\alpha}^l(\Delta)$ analogously provides a lower bound. Please refer to [20] for detailed discussion. The concept of arrival curves unifies many other common timing models of event streams, such as periodic, sporadic, periodic with jitter, etc. For example, the arrival curves for a stream that is described by period p, jitter j and minimum interarrival distance d can be computed with

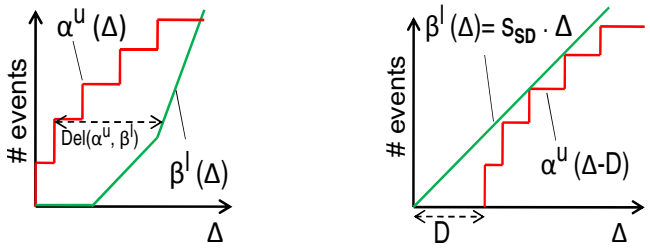
$$\bar{\alpha}^u(\Delta) = \min\left\{\left\lceil\frac{\Delta + j}{p}\right\rceil, \left\lceil\frac{\Delta}{d}\right\rceil\right\}, \; \bar{\alpha}^l(\Delta) = \left\lfloor\frac{\Delta - j}{p}\right\rfloor. \tag{2}$$

Throughout this paper, we implicitly focus our study on systems processing one event stream. For all events in the event stream, we assume that they have the same execution time C at the maximum speed $s_{\max}$ and the same relative deadline D. Moreover, we assume that executing at speed s takes $\frac{C}{s}$ amount of execution time. We use a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)] = [C \cdot \bar{\alpha}^u(\Delta), C \cdot \bar{\alpha}^l(\Delta)]$ to denote the computation demand of the event stream. Note that, as we normalized $s_{\max}$ to 1, $\alpha(\Delta)$ will represent the maximum number of *normalized computation cycles* that can arrive in any time interval with length $\Delta \geq 0$. Similarly, $R(t)$ is the accumulative normalized computation cycles arriving at the system from time 0 to time t, where $R(t) = C \cdot \bar{R}(t)$.

An event e_i is associated with its arrival time a_i and its absolute deadline $d_i = a_i + D$, and C is the execution time at speed $s_{\max}$. Note that, on-line scheduling algorithms do not know when an event e_i will come before it arrives at time a_i. The scheduling objective is to minimize the energy consumption without violating the timing constraints of events.

Analogously to the cumulative function $R(t)$, the concrete availability of the system can be described by a cumulative function $F(t)$, that is defined as the number of available resources, i.e., normalized computation cycles in this paper, in the time interval $[0, t)$. Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves provides an abstract resource model. The upper and lower service curves provide upper and lower bounds on the available resources in any time interval of length $\Delta \geq 0$.

Any arbitrary resource availability can be represented with the model of service curves. For example, for a schedule which executes events at a constant speed s, both the upper and the lower service curves are equal and are represented by straight lines $\beta^u(\Delta) = \beta^l(\Delta) = s \cdot \Delta$.



(a) Graphical interpretation for (3)

(b) Graphical interpretation for Algorithm SD

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

III. Scheduling Algorithms and Framework

A. Static DVS Scheduling Based on Arrival and Service Curves

Arrival curves bound the arrivals of events from an event stream. Using results from Network and Real-Time Calculus [9], [15], [20], it is possible to bound the maximum delay that events can experience given a certain system availability characterized by a service curve. Specifically, the maximum delay experienced by an event from the stream can be defined as $\text{Del}(\alpha^u, \beta^l)$ as follows:

$$\text{Del}(\alpha^u, \beta^l) \equiv \sup_{\lambda \geq 0}\left\{\inf\{\tau \geq 0 : \; \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\}\right\}, \tag{3}$$

which can be interpreted as the maximum horizontal distance between the curves α^u and β^l as shown in Figure 1a. The stream is *schedulable* if the maximum delay is smaller than the relative deadline, i.e., $\text{Del}(\alpha^u, \beta^l) \leq D$ which can also be expressed as the inequality

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

Since the speed-dependent active power consumption is a convex and increasing function of the processor speed, to reduce the energy consumption, an optimal solution would like to execute an event at a constant speed. Therefore, for periodic real-time tasks, it has been shown that running at a constant speed is optimal [2]. Suppose that a static DVS schedule enforces the system to run at a constant speed. Based on the above analysis, the best static DVS schedule

is to find a minimum speed s_{SD} such that

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

and execute at a constant speed $\max\{s^*_{\min}, s_{SD}\}$. Figure 1b illustrates an example of the above strategy. Based on Real-Time Calculus, executing constantly at the above speed guarantees the schedulability of the given event stream, even in the case of worst-case arrivals of events, and saves energy consumption, compared to executing events greedily at the maximum speed $s_{\max}$. For brevity, we denote the above approach as Algorithm SD (stands for static DVS).

As it is not necessary to force the system to be active when there is no event to serve, for the rest of this paper, we will simply assume that the system is in sleep mode when there is no unfinished event in the ready queue. Note that, this does not affect the schedulability of events by applying Algorithm SD, since we can turn the system to active mode instantly when an event comes.

B. On-Line DVS Algorithms

Even though arrival curves constrain the possible traces of events, they do not reveal how events *actually* arrive at the system. It might be possible that the burst of event arrivals only happens once, but we have to always provide higher computation power in Algorithm SD as if events are arriving with worst-case arrivals. Therefore, applying scheduling decisions based on the upper arrival curve could be very pessimistic. Instead, we would like to apply on-line DVS scheduling algorithms to make scheduling decisions only when events *really* arrive. As most systems do not have event bursts all the time, the on-line algorithms can help reduce the energy consumption.

There have been some on-line DVS algorithms proposed in the literature [5], [19], [22]. To our best knowledge, most algorithms are variants of *Algorithm AVR* and *Algorithm OPT* proposed by Yao, Demers, and Shenker [22]. Therefore, in this paper, we will study these two algorithms. They make scheduling decisions at time t as follows:

- *Algorithm AVR*: It selects the event with the earliest deadline and executes it at speed $\max\{s^*_{\min}, \sum_{e_i:a_i \leq t < d_i} \frac{C}{D}\}$, where an event e_i is associated with arrival time a_i and absolute deadline $d_i = a_i + D$, and C and D are the worst-case execution time (at $s_{\max}$) and the relative deadline, respectively. In other words, Algorithm AVR executes at the accumulated average speed of events.
- *Algorithm OPT*: It executes the event with the earliest deadline at speed $\max\left\{s^*_{\min}, \max_{e_j}\left\{\sum_{e_i:a_i \leq t, d_i \leq d_j} \frac{C_i(t)}{d_j - t}\right\}\right\}$, where an event e_i is associated with arrival time a_i and absolute deadline $d_i = a_i + D$, and the remaining (unfinished) computation of event e_i at time t is $C_i(t)$. In other words, Algorithm OPT greedily makes on-line speed scheduling by only considering those events that have arrived in the system so far, and selecting the minimum necessary speed to process them within their deadlines, i.e., with the optimal energy consumption.

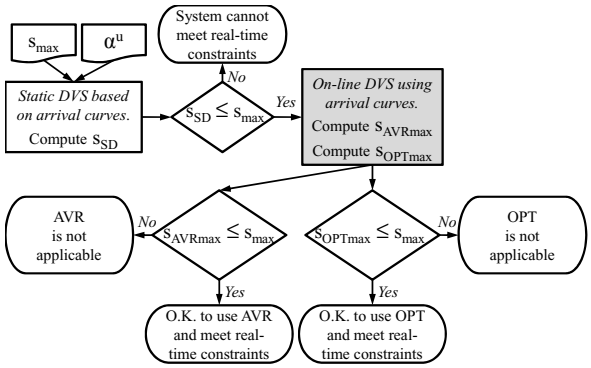


Figure 2: Structure of the proposed framework.

If there is no speed constraint in the system (i.e., $s_{\max} = \infty$), these two on-line DVS algorithms can derive feasible schedules to meet the timing constraints of the real-time events. However, if the system is constrained by some speed, it is possible that the required execution speed, at some moment t, of the resulting schedule might be higher than speed $s_{\max}$, and some events might miss their deadlines even if we execute at the maximum speed $s_{\max}$. Therefore, the highest speed used in a schedule derived from Algorithm AVR or Algorithm OPT must be verified in advance by considering the arrival curve of the input event stream. We say that a scheduling algorithm is *feasible* to use for scheduling an event stream if any feasible trace of the stream is schedulable by applying the scheduling algorithm without any deadline misses or speed violations.

C. Framework for Feasibility Analysis

In this paper, we propose a framework for feasibility analysis shown in Figure 2 that will be able to answer the following questions given information about the maximum speed supported by the system $s_{\max}$ and an upper bound α^u on the input event stream:

- Can the system meet real-time constraints (e.g., do not miss any deadlines) under a constant speed, i.e., the minimum necessary constant speed s_{SD} is smaller than or equal to the maximum system speed $s_{\max}$?
- Can the system meet real-time constraints when the speed is controlled by an online DVS algorithm such as AVR or OPT, i.e., the maximum speed that such an algorithm can require $s_{AVR\max}$ or $s_{OPT\max}$ is smaller than or equal to $s_{\max}$?

If an on-line scheduling algorithm can guarantee the satisfaction of the speed constraint, we would prefer to adopt the on-line algorithm for speed determination. In general, Algorithm OPT has better performance in energy minimization than Algorithm AVR [17]. In Section VII, we will compare the performance of algorithms SD, AVR, and OPT.

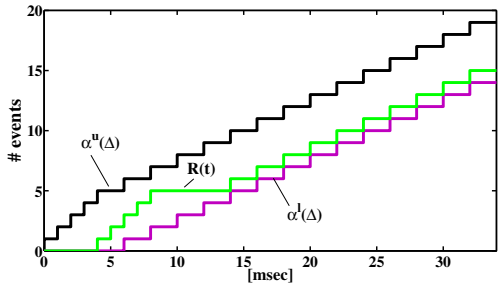


Figure 3: Arrival curves $[\alpha^u, \alpha^l]$ in interval domain and a trace R in time domain for the motivational example

IV. Motivational Example

This section will present a motivational example for demonstrating the benefit of on-line algorithms. Suppose that we are given a periodic event stream with specified jitters, where the arrival curve is specified in (2) with $p = 2$ msec, $d = 1$ msec, and $j = 4$ msec. Let's suppose that the maximum speed is 1 GHz, in which C (at speed $s_{\max}$) is 1 msec and the relative deadline D is 4 msec. The power consumption function in this example is $P(s) = \hbar(\frac{s}{1GHz})^3$ Watt, where $s_{\min}$ is assumed 0. The arrival curve of the above case is illustrated in Figure 3.

Suppose that we release 15 events bounded by the arrival curve, where the cumulative function R of the arrivals (from time 0) is also shown in Figure 3. These 15 events arrive at times $(4, 5, 6, 7, 8, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32)$ msec. We compare the energy needed by the three algorithms to process all the events within their deadlines. By applying (5), for the given arrival curve, we know that speed s_{SD} is 0.625 GHz. The schedule by applying Algorithm SD for scheduling these 15 events at speed s_{SD} is illustrated in Figure 4a, where the energy consumption for serving these events is 5.8594 mJoule. The disadvantage of Algorithm SD is that it might complete an event earlier than its deadline, and such early completion makes the system consume more energy. The advantage of Algorithm SD is that we do not have to dynamically calculate the execution speeds for timing guarantees.

Figure 4b (Figure 4c, respectively) demonstrates the schedule by applying Algorithm OPT (Algorithm AVR, respectively) for scheduling. Both algorithms utilize the event deadlines so that events complete earlier than their deadlines only when there is interference from other events. The energy consumption by applying Algorithm OPT (Algorithm AVR, respectively) for this trace is 4.601 (5.4375, respectively) mJoule. For scheduling these 15 events, the speeds used by Algorithm OPT (AVR, respectively) are no more than 0.7627 GHz, (1.0 GHz, respectively). Moreover, the improvement of energy consumption of the on-line algorithms could be more if the burst of events does not happen, or the gap between the trace and the upper arrival curve is larger.

As shown above, even though running at speed s_{SD} is

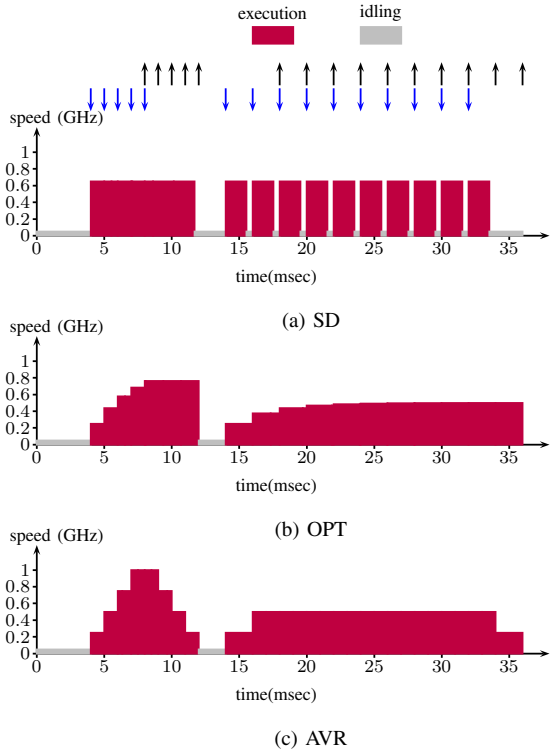


Figure 4: An example for different scheduling algorithms.

safe, it might be too conservative in energy minimization since we can vary the speed dynamically to reflect the events that actually arrive at the system. The speed scheduling of Algorithm AVR or Algorithm OPT is highly dependent on the input trace, while Algorithm SD depends only on the worst-case arrival. However, if the maximum speed of the system is less than 0.75 GHz, Algorithm OPT or Algorithm AVR are not applicable.[1] Therefore, to apply an on-line algorithm for energy reduction, it needs to be verified that the speed needed by the on-line DVS algorithm is never higher than the maximum speed supported by the system; otherwise, events may miss deadlines.

V. Feasibility Analysis of On-Line Algorithms

This section will present how to analyze the feasibility of Algorithm AVR and Algorithm OPT to guarantee that the derived solution can meet the timing constraints without using any speed higher than $s_{\max}$. Since the number of traces that one may need to consider could be infinite, it is not possible or not practical to test this on all traces. Therefore, we use in our analysis the arrival curves of an event stream.

For any of Algorithm OPT and Algorithm AVR, if we ignore the speed constraint $s^*_{\min}$ when making decisions, the maximal speed used in derived schedule is less than or equal to the maximal speed used by constraining the algorithm to use speeds greater or equal to speed $s^*_{\min}$. Therefore, for the rest of this section, we will focus on the case that $s^*_{\min}$

[1]For showing the feasibility of Algorithm AVR and Algorithm OPT, we do not use the normalized $s_{\max}$ here.

is 0, and the results here still hold when $s^*_{\min}$ is not 0. We will first analyze the feasibility of Algorithm AVR and then present how to approximately analyze that of Algorithm OPT.

For a trace R, suppose that $s_{AVR}(t, R)$ is the speed at time t by applying Algorithm AVR, whereas $s_{OPT}(t, R)$ is the speed at time t by applying Algorithm OPT. Without loss of generality, we implicitly assume that a trace starts at time 0, unless it is specified.

A. Feasibility Analysis of Algorithm AVR

The following theorem shows the maximal speed of the schedule derived from Algorithm AVR for any trace R constrained by the upper arrival curve $\alpha^u()$.

Theorem 1: For any trace R constrained by the upper arrival curve $\alpha^u()$, the maximal speed $s_{AVR\max}$ by applying Algorithm AVR for scheduling is at most $\frac{\alpha^u(D)}{D}$.

Proof: For brevity, let s^* be the maximal speed in the speed schedule $s_{AVR}(t, R)$, i.e., $s^* = \sup_{0\le t\le\infty}\{s_{AVR}(t, R)\}$. Suppose that at time t^*, the schedule uses the maximal speed s^* for execution, i.e., $s_{AVR}(t^*, R) = s^*$ for trace R. Let $E(t^*)$ be the set of events that could be executed at time t^*, i.e., $E(t^*) = \{e_i \mid e_i \text{ is in trace } R \text{ and } a_i \le t^* < a_i + D\}$. By the definition of Algorithm AVR, we know that

$$s^* = \frac{C}{D}|E(t^*)|,$$

where $|E(t^*)|$ is the number of events in set $E(t^*)$. If an event e_i arrives earlier than $t^* - D$, the feasible execution interval of the event does not cover t^* since the deadline is less than t^*. Moreover, of course, before time t^*, Algorithm AVR does not execute any event arriving later than t^*.

Therefore, all the events in $E(t^*)$ must arrive in $[t^* - D, t^*)$. By the definition that R is constrained by $\alpha^u(\Delta)$, we know that $R(t^*) - R(t^* - D) \le \alpha^u(D)$. As a result, $C \cdot |E^*(t)| \le \alpha^u(D)$, and, hence,

$$s^* \le \frac{\alpha^u(D)}{D}.$$

■

As a result, we have the following corollary for the feasibility of Algorithm AVR.

Corollary 1: Algorithm AVR guarantees to derive feasible schedules without violating the timing constraint or speed constraint if $\frac{\alpha^u(D)}{D} \le s_{\max}$.

The above analysis in Theorem 1 is tight, which means that there exists some input trace constrained by the arrival curve with the maximal speed in the AVR speed scheduling equal to $\frac{\alpha^u(D)}{D}$.

Example 5.1: Consider the example in Section IV. The analytically determined maximal speed of Algorithm AVR is $\frac{4}{4} = 1$ GHz for the example. As shown in Figure 4c, the maximal speed of the trace by applying Algorithm AVR is also 1 GHz in time interval $[7, 9)$ msec.

Moreover, the following theorem shows that the maximal speed of the resulting speed scheduling by applying Algorithm AVR is bounded by any feasible speed scheduling for a given trace R.

Theorem 2: Suppose that s^* is the maximal speed for scheduling a trace R by applying off-line scheduling. The maximal speed by applying Algorithm AVR for scheduling is at most $2s^*$.

Proof: Let us define $E(t^*)$, s^*, and t^* by the same terminologies as in the proof of Theorem 1. Now, we prove that, in the trace R, any feasible off-line scheduling must execute at least at speed $\frac{s^*}{2}$ at some moment. Clearly, all the events in $E(t^*)$ have deadlines no later than $t^* + D$. Suppose that event e_1 (e_2, respectively) is the earliest (latest, respectively) event in $E(t^*)$ arriving at the system at time t_1 (t_2, respectively). By definition, $t_2 - t_1$ is no more than D. Moreover, $d_2 - t_1 = t_2 + D - t_1 \le 2D$. Therefore, there must be some moment in interval $[t_1, t_2 + D)$ at which the scheduler is executed at some speed not lower than $\frac{C\cdot|E(t^*)|}{d_2-d_1} \ge \frac{C\cdot|E(t^*)|}{2D} = \frac{s^*}{2}$, which proves the theorem. ■

B. Feasibility Analysis of Algorithm OPT

We now analyze the maximal speed of the derived schedule from Algorithm OPT for any input trace constrained by an input arrival curve. We will first show that the derived bound of the maximal speed in Theorem 1 also works for Algorithm OPT, but it might not be tight. Later in this subsection, we will give a tighter bound by simulating an adversative trace.

The following lemma shows that the solution of Algorithm OPT lasts for at least D time units at the maximal speed of the schedule once it reaches the highest speed of the schedule.

Lemma 1: Suppose that t' is the earliest time at which $s_{OPT}(t', R) \equiv \sup_t s_{OPT}(t, R)$. The schedule executes at a constant speed from time t' to time $t' + D$.

Proof: Suppose for contradiction that the schedule changes the speed at time $\hat{t}$, where $t' < \hat{t} < t' + D$. By the definition of Algorithm OPT, we can only increase the speed at $\hat{t}$, which contradicts the definition of time t'. ■

With Lemma 1, we can prove a loose bound of the maximal speed by applying Algorithm OPT.

Theorem 3: For any trace R constrained by the upper arrival curve $\alpha^u()$, the maximal speed $s_{OPT\max}$ by applying Algorithm OPT for scheduling is at most $\frac{\alpha^u(D)}{D}$.

Proof: Let us define $E(t^*)$, s^*, and t^* by the same terminologies as in the proof of Theorem 1. With Lemma 1, we know that $s^* \le \frac{C\cdot|E(t^*)|}{D} \le \frac{\alpha^u(D)}{D}$. ■

As a result, if Algorithm AVR is applicable for the system, Algorithm OPT is also applicable. However, the above analysis might not be tight. For the rest of this subsection, we will analyze a tighter bound of the maximal speed of $s_{OPT}(t, R)$.

Suppose that $R^\dagger$ is an infinite *adversative trace* of the arrival curve $\alpha^u()$ that is defined *backwards* from some time $t^\dagger > 0$ until $-\infty$ as follows:

- Fix a time instant $t^\dagger$.

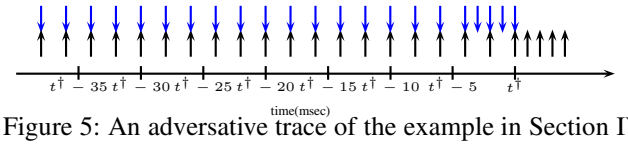


Figure 5: An adversative trace of the example in Section IV.

- All events before $t^\dagger$ are released such that $R^\dagger(t^\dagger) - R(t^\dagger - \delta) = \alpha^u(\delta)$ for any $\delta \geq 0$.

The adversative trace $R^\dagger$ can be treated as the trace that has the largest interference at time $t^\dagger$. For example, the adversative trace of the motivational example in Section IV is shown in Figure 5.

Note that, by definition, $R^\dagger$ is infinitely long for an upper-arrival curve that is infinitely long, in which case Figure 5 only presents a part of $R^\dagger$. The following lemma shows that the maximal speed in schedule $s_{OPT}(t, R)$ is bounded by $s^\dagger$ where $s^\dagger = s_{OPT}(t^\dagger, R^\dagger)$.

Lemma 2: $s_{OPT}(t, R) \leq s^\dagger$ for any $t \geq 0$.

Proof: Suppose for contradiction that $\hat{t}$ is the earliest time in the schedule of trace R such that $s_{OPT}(\hat{t}, R) \geq s^\dagger$, where $s_{OPT}(\hat{t}, R) \geq s_{OPT}(t, R)$ for any t. By Lemma 1, we know that $s_{OPT}(t, R) = s_{OPT}(\hat{t}, R)$ for any $\hat{t} \leq t < \hat{t} + D$. Suppose that $\Lambda(t, \hat{t} + D, R)$ is the amount of computation done in time interval $[t, \hat{t} + D)$ by applying Algorithm OPT for trace R. Similarly, $\Lambda(t, t^\dagger + D, R^\dagger)$ is that for trace $R^\dagger$.

We assume that $s_{OPT}(\hat{t}, R) = (1 + \epsilon)s^\dagger$ for a positive ϵ since $s_{OPT}(\hat{t}, R) > s_{OPT}(t^\dagger, R^\dagger)$. By Algorithm OPT, we also know that

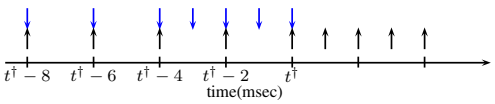
$$\begin{aligned}&\Lambda(\hat{t}, \hat{t} + D, R) = s_{OPT}(\hat{t}, R)D \\ =&(1 + \epsilon)s^\dagger \geq (1 + \epsilon)\Lambda(t^\dagger, t^\dagger + D, R^\dagger).\end{aligned}$$

We now discretize time interval in $[0, \hat{t}]$ by arrival times of events in trace R, i.e., $0 = t_0 < t_1 < t_2 < \cdots < t_n = \hat{t}$ such that events in R only arrive at the system at these discrete time instants. We denote the interval length of time interval $[t_k, \hat{t}]$ as δ_k. Suppose that in time interval $[t_k, \hat{t}]$, trace R releases r_k events. We know that at time instant t_k, trace R releases exactly $r_k - r_{k+1}$ events with computation demand $(r_k - r_{k+1})C$. Similarly, suppose that $r_k^\dagger$ is the number of events arriving the system in time interval $[t^\dagger - \delta_k, t^\dagger]$. By the definition of trace $R^\dagger$, we know that $r_k \leq r_k^\dagger$.

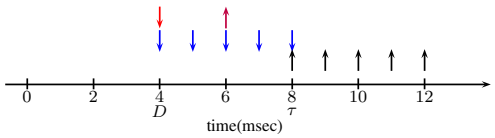
Based on the assumption $\Lambda(\hat{t}, \hat{t}+D, R) > (1+\epsilon)\Lambda(t^\dagger, t^\dagger + D, R^\dagger)$ and $r_k \leq r_k^\dagger$, we also know that $\Lambda(\hat{t}-\delta_k, \hat{t}+D, R) \geq (1+\epsilon)\Lambda(t^\dagger - \delta_k, t^\dagger + D, R^\dagger)$ for any $k = 0, 1, 2, \ldots, n$. This implies that $R(\hat{t}) - R(\hat{t} - \frac{D}{\epsilon}) > R^\dagger(t^\dagger) - R^\dagger(t^\dagger - \frac{D}{\epsilon})$ or $R(\hat{t}) > R^\dagger(t^\dagger) - R^\dagger(t^\dagger - \hat{t})$, which contradicts the definition of trace $R^\dagger$. ■

The trace $R^\dagger$ provides an upper bound of the speeds used by Algorithm OPT, but it is infinitely long. Therefore, deriving $s^\dagger$ requires to release infinite events, which is impractical. To conquer this issue, we will present how to derive an upper bound of $s^\dagger$ by using an *approximative* trace R' which has a fixed length.

For a given $\tau > D$, suppose that R' is an approximative finite trace defined for the interval $[0, \tau]$ as follows:



(a) An infinite trace $R^\dagger$



(b) A finite trace R' with $\tau = 8$

Figure 6: Traces $R^\dagger$ and R' for the example in Section IV, where the event arriving at time 2 is changed to arrive at time 4 with absolute deadline 6.

- Release events such that $R'(\tau) - R(\tau - \delta) = \alpha^u(\delta)$ for any $\tau \geq \delta \geq 0$.
- For an event τ_i in R' with arrival time less than D, i.e., $a_i < D$, change the arrival time of the event to D without changing its absolute deadline.

Note that, trace $R^\dagger$ is a valid trace for the input event stream, while R' is artificially created only for the purpose of the feasibility analysis. Figure 6 presents an example for R' when $\tau = 8$ msec for the example in Section IV. The following lemma shows that for any given R' constructed as described above, the maximum speed at time τ by applying Algorithm OPT for trace R' is an upper bound of $s^\dagger$.

Lemma 3: $s^\dagger \leq s_{OPT}(\tau, R')$.

Proof: Since $t^\dagger$ could be any value, we will assume that the trace $R^\dagger$ is constructed such that $t^\dagger$ is τ. Let E' ($E^\dagger$, respectively) be the set of events in trace R' ($R^\dagger$, respectively). By the construction of trace R', if an event is released at time t with $t \geq D$ in $R^\dagger$, the event is also released at time t in R'. Moreover, if an event is released at time t with $t < D$ in $R^\dagger$, the event is released at time D in R' but still with the same absolute deadline. For notational brevity, we index the events in E' and $E^\dagger$ such that e_i in E' and e_i in $E^\dagger$ have the same absolute deadline by a one-to-one mapping.

Now, let's discretize the time interval $(D, \tau + D]$ as $t_0 < t_1 < t_2 < \cdots < t_n$ such that the speed changes either in the speed schedule of $R^\dagger$ or R', where t_0 is defined as D and t_n is $\tau + D$. Therefore, we know that at any time t with $t_k \leq t \leq t_{k+1}$ for some $0 \leq k \leq n-1$, $s_{OPT}(t, R') = s_{OPT}(t_k, R')$ and $s_{OPT}(t, R^\dagger) = s_{OPT}(t_k, R^\dagger)$. To prove the statement, we are going to prove that $s_{OPT}(t_k, R') \geq s_{OPT}(t_k, R^\dagger)$ for any $k = 0, 1, 2, \ldots, n$ by applying mathematical induction.

Let $C_i(t, R')$ ($C_i(t, R^\dagger)$, respectively) be the remaining computation at time t for event e_i in trace R ($R^\dagger$, respectively) by applying Algorithm OPT. As $s_{OPT}(t_k, R) = \max_{e_j} \left\{ \sum_{e_i : a_i \leq t_k, d_i \leq d_j} \frac{C_i(t_k)}{d_j - t_k} \right\}$ for speed decision in Algorithm OPT, we are going to prove that $C_i(t_k, R') \geq C_i(t_k, R^\dagger)$ for any event e_i and any $k = 0, 1, 2, \ldots, n$, which implies $s_{OPT}(t_k, R') \geq s_{OPT}(t_k, R^\dagger)$.

When $k = 0$, since trace R' does not release those events arriving in time interval $(0, D]$ in $R^\dagger$ before time D, we know $C_i(t_0, R') \geq C_i(t_0, R^\dagger)$. Suppose that when $k = k'$, $C_i(t_{k'}, R') \geq C_i(t_{k'}, R^\dagger)$ for any event e_i in $E^\dagger$ and E'. Let's now consider time instant $t_{k'+1}$.

For the case that $s_{OPT}(t_{k'+1}, R^\dagger) < s_{OPT}(t_{k'}, R^\dagger)$, by the hypothesis $C_i(t_{k'}, R') \geq C_i(t_{k'}, R^\dagger)$, we also know that $s_{OPT}(t_{k'+1}, R') < s_{OPT}(t_{k'}, R')$. If $s_{OPT}(t_{k'+1}, R') < s_{OPT}(t_{k'}, R')$, Algorithm OPT does not execute any event with absolute deadline later than t_{k+1}. Therefore, we know that $C_i(t_{k'+1}, R') \geq C_i(t_{k'+1}, R^\dagger)$ for any event e_i in $E^\dagger$ and E'.

For the case that $s_{OPT}(t_{k'+1}, R') > s_{OPT}(t_{k'}, R')$, we know that at time $t_{k'}$ Algorithm OPT decides to run at a constant speed from $t_{k'}$ to a time instant $t^\sharp$. Since $s_{OPT}(t_{k'+1}, R') > s_{OPT}(t_{k'}, R')$, we know that $t^\sharp \geq t_{k'+1}$ and $t^\sharp$ is the absolute deadline of an event. By the hypothesis $C_i(t_{k'}, R') \geq C_i(t_{k'}, R^\dagger)$, it is not difficult to see that $C_i(t, R') \geq C_i(t, R^\dagger)$ at any time instant t before $t^\sharp$. Therefore, since $t_{k'+1} \leq t^\sharp$, we know that $C_i(t_{k'+1}, R') \geq C_i(t_{k'+1}, R^\dagger)$ for any event e_i in $E^\dagger$ and E'. For the case that $s_{OPT}(t_{k'+1}, R^\dagger) > s_{OPT}(t_{k'}, R^\dagger)$, the analysis is similar.

Therefore, by induction hypothesis, we prove the statement that $s_{OPT}(t_k, R') \geq s_{OPT}(t_k, R^\dagger)$ for any $k = 0, 1, 2, \ldots, n$, which also proves the lemma. ■

Based on Lemma 2 and Lemma 3, we know that $s_{OPT}(\tau, R')$ is a safe upper bound of the maximal speed in $s_{OPT}(t, R)$, where R is any trace constrained by an upper arrival curve, and R' is an approximative trace for the same upper arrival curve.

Theorem 4: For any trace R characterized by an upper arrival curve $\alpha^u()$ and relative deadline $D > 0$, the maximal speed $s_{OPT\,\max}$ by applying Algorithm OPT for scheduling is at most $s_{OPT}(\tau, R')$, where R' is an approximative trace for $\alpha^u()$ with length $\tau > D$.

Proof: This can be shown directly with Lemma 2 and Lemma 3. ■

Example 5.2: For the example in Section IV, based on Theorem 4, we can compute an upper bound on the maximum system speed that Algorithm OPT uses when processing any trace R constrained by the arrival curves described in the example. Figure 7 illustrates the influence of parameter τ on the tightness of the result from Theorem 4. As we can see for this example, the analysis is tight for $\tau \geq 8$ msec, i.e., $s^\dagger = s_{OPT}(8, R') = 0.8418$ GHz.

VI. Competitive Analysis for Energy Minimization

We say that an algorithm has a ρ-competitive ratio if it derives solutions that are at most ρ times of the energy consumption of the optimal off-line solutions. It has been shown in [4], [22] that the competitive ratio of Algorithm AVR (OPT, respectively) is $2^{\gamma-1}\gamma^\gamma$ (γ^γ, respectively) when the system has more than one event stream. As a result, even though Algorithm AVR and Algorithm OPT cannot derive optimal solutions due to the lack of precise arrival information a priori, they can still provide solutions that are not far away from the optimal ones that are computed with clairvoyant information on arrival times.

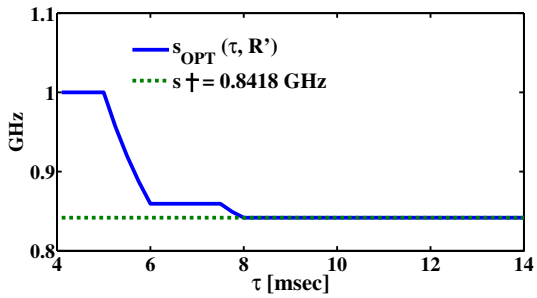


Figure 7: Tightness of the result of Theorem 4 for different lengths τ of trace R' for the example in Section IV.

Suppose that an on-line algorithm is applicable for the event stream. We are going to show that for systems with only one event stream, the competitive ratios of Algorithm OPT and Algorithm AVR are both 2^γ for any input trace constrained by the given event stream. Suppose that R is given to the off-line algorithm and $s_{OFF}(t, R)$ is the speed at time t of the optimal off-line schedule for energy minimization derived with the off-line algorithm proposed by Yao et al. [22]. The following lemma states the relation between $s_{OFF}(t, R)$ and the on-line schedules.

Lemma 4: For any input trace R and any time instant t, $s_{OFF}(t, R) \geq \frac{1}{2}s_{OPT}(t, R)$ and $s_{OFF}(t, R) \geq \frac{1}{2}s_{AVR}(t, R)$.

Proof: Suppose for contradiction that at time instant t^* with $s_{OFF}(t^*, R) < \frac{1}{2}s_{AVR}(t^*, R)$. We can show that this contradicts the optimality of the off-line solution. The optimal off-line algorithm in [22] repeatedly finds the maximum *intensity* of intervals. To calculate the intensity of an interval $[t_\ell, t_r]$, the algorithm counts the number events that arrive in this interval and also associate with deadlines in this interval. The assumption of $s_{OFF}(t^*, R) < \frac{1}{2}s_{AVR}(t^*, R)$ leads to the predicate that $\frac{|E_{t_\ell, t_r}|}{t_r - t_\ell} < \frac{1}{2}s_{AVR}(t^*, R)$ for any interval $[t_\ell, t_r]$ with $t_\ell \leq t^* < t_r$, where E_{t_ℓ, t_r} is the set of events e_i with $t_\ell \leq a_i \leq d_i \leq t_r$. However, with similar arguments in the proofs of Theorem 2, we know that $\frac{|E_{t^*-D, t^*+D}|}{t^*-D-(t^*+D)} \geq \frac{1}{2}s_{AVR}(t^*, R)$, which reaches the contradiction.

The proof for Algorithm OPT is very similar, and is omitted due to space limitation. ■

We conclude this section by showing the competitive ratio of Algorithm AVR and Algorithm OPT.

Theorem 5: Algorithm AVR and Algorithm OPT are both 2^γ-competitive on-line algorithms for energy minimization.

Proof: The energy consumption is the integration of power consumption in time. Since the speed of the optimal off-line schedule is at least 0.5 of the on-line schedule by applying either Algorithm AVR or Algorithm OPT, we know

	1	2	3	4	5	6	7	8	9	10
p	198	102	283	354	239	194	148	114	313	119
j	387	70	269	387	222	260	91	13	302	187
d	48	45	58	17	65	32	78	–	86	89
C	36	40	70	110	80	50	60	50	50	60
D	110	140	310	445	280	240	200	120	340	200

Table I: Parameters for the 10 different streams in [msec].

that the energy consumption of their resulting solution is at most 2^γ times of the optimal off-line energy consumption. ■

VII. Performance Evaluation

In this section, we evaluate the performance of Algorithm SD, Algorithm AVR, and Algorithm OPT in terms of energy usage and required maximum speed, and compare them to the off-line algorithm proposed in [22], denoted as Algorithm Offline. Algorithm Offline is completely clairvoyant, it knows the arrival times of all events in advance and makes scheduling decisions with the optimal energy consumption and the optimal required maximum speed. Algorithm Offline is used only as a reference. At the end, we also compare the results observed in the experiments with results computed with Theorems 1 and 4.

Experimental Setup: For comparing the performance of the four algorithms, we use a set of 10 different event streams adapted from [21]. The streams are described with period, jitter, minimum distance, worst-case execution time, and relative deadline as given in Table I. For each stream, 10 random traces have been generated to be as close to the upper arrival curve as possible, each with length of 20000 msec. We measure the maximum speeds reached by each algorithm for scheduling the different traces for a particular stream, and calculate the average energy needed by each algorithm for scheduling the different traces for a particular stream.

When speed-dependent power consumption P_{ind} is negligible, on-line DVS scheduling must be done carefully since the critical speed does not play a role. Therefore, to demonstrate the impact of on-line DVS algorithms, we will assume P_{ind} is 0 and $s^*_{\min}$ is 0 in our performance evaluation. The power consumption function of Intel XScale is adopted here for performance evaluation, in which the power consumption function $P(s)$ is approximately $0.04 + \hbar(1.56(\frac{s}{1GHz})^3$Watt. The maximum speed is 1GHz.

Evaluation Results: Results for the maximum speed required by all algorithms for the different streams are shown in Figure 8. For all streams, Algorithm AVR requires the highest speed. Followed by Algorithm OPT which more accurately computes at each time instance the speed necessary to process all unprocessed events in the system within their deadlines. Algorithms SD and Offline need the smallest maximum speeds. Algorithm Offline always chooses the optimal speed and this speed cannot be higher than the one computed with Algorithm SD but it may be lower. This is the case when a trace significantly deviates from the worst-case

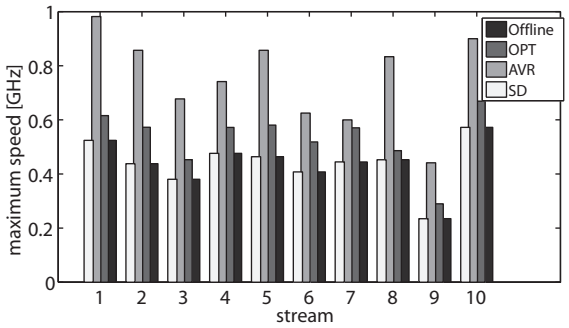


Figure 8: Maximum system speeds required by evaluated algorithms from each stream.

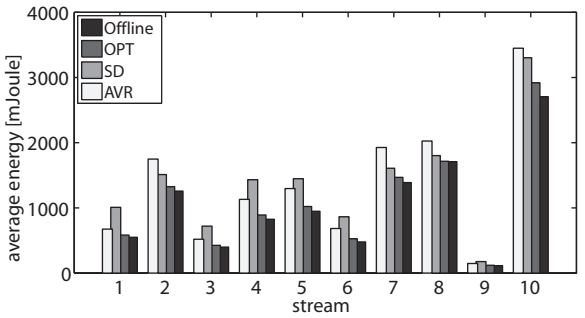


Figure 9: Average energy required by the evaluated algorithms from the different traces for each stream.

arrival because Algorithm SD relies only on the worst-case arrival (α^u) to compute its constant execution speed.

Results for the average energy required by the algorithms to process all events from the different traces for each stream are shown in Figure 9. As can be expected, Algorithm Offline always shows the smallest average amount of energy. Followed closely by Algorithm OPT. Algorithms SD and AVR perform the worst. For the more bursty streams (1,3,4,5,6,9), Algorithm SD performs worse than AVR because it runs at a constant speed calculated with the assumption that a burst can happen at any time instant. While for the less bursty streams, Algorithm SD runs at a more optimal speed than Algorithm AVR and consumes less energy on the average.

In a summary, the experimental results clearly show the trade-offs in using the different scheduling algorithms. A static DVS schedule such as Algorithm SD, requires a low constant system speed that guarantees meeting real-time constraints. However, it may be energy inefficient when events do not arrive following the worst-case scenario. Such an algorithm can be suitable for systems that need to meet real-time constraints but available energy is not their primary constraint. On the other hand, online DVS algorithms can offer energy savings but they may require a higher system speed. There is a clear advantage for prefering Algorithm OPT than AVR since it requires a lower maximum system speed and it is more energy efficient on the average.

	1	2	3	4	5	6	7	8	9	10
AVR	0.982	0.857	0.677	0.742	0.857	0.625	0.6	0.833	0.441	0.9
Thm. 1	0.982	0.857	0.677	0.742	0.857	0.625	0.6	0.833	0.441	0.9
OPT	0.616	0.573	0.453	0.572	0.581	0.518	0.571	0.486	0.29	0.669
Thm. 4	0.616	0.577	0.455	0.58	0.587	0.523	0.573	0.486	0.293	0.67

Table II: Validation of results from Thm. 1 and Thm. 4.

Experimental results show the tightness of the theoretical results from the previous part of the paper. In Table II, we compare the observed maximum speed from all traces scheduled with Algorithm AVR for each stream and compare it with the calculated result for the stream with Theorem 1. We do the same for Algorithm OPT and Theorem 4 with approximative traces of length $\tau = 3D$ msec where D is the relative deadline of a stream. Results show that for a given stream it is possible to compute a tight upper bound for the maximum speed of Algorithm OPT which is valid for all traces from this stream even when a relatively short approximative trace is used.

Please note that generating the approximative trace is very efficient, taking less than 1 second on a Pentium 4 laptop.

VIII. Conclusion and future work

The paper presents a framework for feasibility analysis of two on-line DVS scheduling algorithms, Algorithm AVR and Algorithm OPT. The analysis can be used for systems with general input event streams that are constrained by arrival curves. We present how to compute tight upper bounds on the maximum system speeds that these algorithms may require. Such analysis shows the feasibility of on-line DVS algorithms to real-time systems where requesting a speed that is higher than the maximum available can lead to missed deadlines. The competitive ratio for energy minimization of the two algorithms is also discussed and new results are presented. The theoretical results have been illustrated with a motivational example and a performance evaluation.

For simplicity of presentation, we only present the results for one event stream. But it is not difficult to see the analysis can be extended to multiple event streams. Suppose that $\alpha_i^u()$ is the upper arrival curve for computation time at speed $s_{\max}$ of event stream S_i associated with relative deadline D_i. Algorithm AVR and Algorithm OPT will not use any speed higher than $\sum_{i=1}^{N} \frac{\alpha_i(D_i)}{D_i}$ for N event streams. This is also tight for Algorithm AVR, but not tight for Algorithm OPT. How to design an adversative trace for Algorithm OPT is an open issue. For future research, we would like to analyze the maximum speed Algorithm OPT requires for multiple event streams, and develop our framework to a wider range of algorithms.

References

[1] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539, 2004.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS*, pages 95–105, 2001.

[3] N. Bansal, H.-L. Chan, T.-W. Lam, and L.-K. Lee. Scheduling for speed bounded processors. In *ICALP '08: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 409–420, 2008.

[4] 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.

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

[6] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367, 2006.

[7] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 153–162, 2006.

[8] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, pages 289–294, 2007.

[9] R. Cruz. A calculus for network delay. i. network elements in isolation. *Information Theory, IEEE Transactions on*, 37(1):114–131, Jan 1991.

[10] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *RTSS*, 2009.

[11] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Periodic power management schemes for real-time event streams. In *CDC*, 2009.

[12] R. Jejurikar and R. K. Gupta. Procrastination scheduling in fixed priority real-time systems. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 57–66, 2004.

[13] R. Jejurikar and R. K. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *DAC*, pages 111–116, 2005.

[14] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *DAC*, pages 275–280, 2004.

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

[16] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *ECRTS*, pages 105–112, 2003.

[17] M. Li, B. J. Liu, and F. F. Yao. Min-energy voltage allocation for tree-structured tasks. *J. Comb. Optim.*, 11(3):305–319, 2006.

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

[19] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 21–24, 2001.

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

[21] 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.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[23] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, pages 183–188, 2002.

[24] 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.