

The Impact of Energy-Efficient Scheduler Overhead on the Performance of Embedded Real-Time Systems

Haisang Wu* Binoy Ravindran*

*ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{hswu02,binoy}@vt.edu

E. Douglas Jensen†

†The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

In this paper, we investigate the impact of scheduler overhead on energy-efficient, real-time scheduling algorithms for battery-powered, mobile embedded systems. We consider algorithms based on deadlines, and those based on time/utility functions (TUFs) and utility accrual (UA) scheduling criteria. The complete experiments include implementation of prototype algorithms in a Linux kernel, and measurement of the actual energy consumption and system performance. We plan to analyze the factors that can possibly affect the system-level performance with scheduler overhead, such as system load, energy model settings, and TUFs, and identify the conditions under which the TUF/UA algorithms have superior performance, despite their high overhead. The preliminary experiments in this paper consider important parameters, such as task execution times, for our future kernel implementation and measurements.

1. Introduction

Energy management is rapidly becoming an essential element of many embedded systems. Usually the CPU is the prime target for energy saving, since it consumes a substantial fraction of the total energy. Real-time embedded systems must produce timely results to satisfy the system's time constraints, while also being energy-efficient.

Dynamic Voltage Scaling (DVS), which adjusts the CPU's supply voltage and its corresponding clock frequency dynamically, is an effective low-power design technique for embedded real-time systems (see [1, 6, 8, 9] and the references therein). Since the energy consumption of CMOS circuits has a quadratic dependency on the supply voltage, lowering the voltage is one of the most effective techniques for reducing energy consumption.

In this paper, we consider battery-powered, embedded real-time systems that operate in environments with dynamically uncertain properties. These uncertainties include transient and sustained overloads on the CPU and other resources due to context dependent execution times.

Overloaded systems naturally lead to scenarios in which only the most critical activities can be executed. If some activities cannot be executed in a timely fashion, completing activities that are more important than those which are more urgent is often desirable. However, the urgency of an application activity is typically orthogonal to its relative importance. E.g., the most urgent activity can be the least important, and vice versa. Thus, clear distinction has to be made between the urgency and the importance of activities.

Deadlines by themselves cannot express both urgency and importance. Thus, we consider the abstraction of time/utility functions (or TUFs) [3] that express the utility of completing an application activity as a function of that activity's completion time. We specify deadline as a binary-valued, downward "step" shaped TUF; Figure 1 shows example step TUFs. Note that a TUF decouples importance and urgency—urgency is measured as a deadline on the x -axis, and importance is denoted by utility on the y -axis.

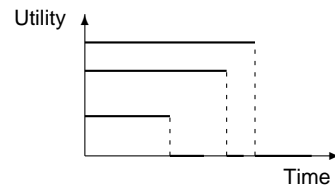


Figure 1. Example Step TUFs (that specify deadlines)

When activity time constraints are specified using TUFs, which subsume deadlines, the scheduling criteria is based on accrued utility, such as maximizing sum of the activities' attained utilities. We call such criteria, *utility accrual* (or UA) criteria, and scheduling algorithms that optimize them, as UA scheduling algorithms.

When energy constraints need to be treated at run-time, UA scheduling should aim at maximizing activities' attained utilities, while reducing system-level energy consumption for prolonging battery life. Typically, an energy-efficient UA algorithm will seek to meet all activity deadlines and minimize energy consumption, when sufficient CPU time is available for doing so.

Several energy-efficient UA algorithms are presented in the literature [10–13]. They show superior performance in terms of the sum of activities’ attained utilities, and system-level energy efficiency, compared to deadline-based energy-efficient algorithms, such as those in [6]. However, energy-efficient UA algorithms are more complex and thus incur more overhead than deadline-based ones. For example, given a ready task queue of length n , the complexity of PA-BTA [10] is $O(n^2)$, and ReUA [13] is $O(n^2 \log n)$, while LaEDF [6] only has complexity of $O(n)$.

There have been relatively few efforts on estimating the impact of scheduler overhead on energy-efficient real-time scheduling. In [8, 9], the overhead of frequency changes is assumed to be negligible compared to the common period. In [1, 6], such overhead is incorporated into the worst-case workloads of tasks, if it is not negligible.

In this paper, we focus on the impact of scheduler overhead on the performance of energy-efficient, real-time scheduling algorithms. We consider repeatedly occurring application activities that are subject to deadlines expressed using downward step TUFs, and consider a system-level energy consumption model [5], where each system component’s energy consumption is individually modeled and aggregated to obtain system-level energy consumption. We plan to fulfill the complete experiments including measuring queuing and DVS overhead of algorithms inside a Linux kernel, and analyze the factors that can possibly affect the system-level performance with scheduler overhead. Preliminary experiments in this paper study the overhead of different energy-efficient real-time algorithms without DVS—i.e., the algorithms work with a unique frequency. These results exploit important parameters such as execution times for our future work.

The paper is organized as follows: Section 2 discusses models and defines the problem. Section 3 and 4 discuss the experimental methodology, preliminary results, and ongoing work. We conclude the paper in Section 5.

2. Models and Problem Definition

2.1. Activity and Timeliness Models

We consider the application to consist of a set of periodic tasks, denoted as $\{T_1, T_2, \dots, T_n\}$. Each task T_i has its own period P_i . All tasks are assumed to be independent—they do not share resources or have any precedence relationships. We assume that task T_i ’s worst case execution cycle (WCEC) is C_i . Under a constant speed i.e., frequency f (given in cycles per second), the worst case execution time of a task T_i is given by $e_i = \frac{C_i}{f}$.

An instance of a task is called a *job*. We refer to the j^{th} job (or invocation) of task T_i as $J_{i,j}$. The basic scheduling entity that we consider is the job abstraction. Jobs can be preempted at arbitrary times.

A job’s time constraint is specified using a step TUF. (It is possible to have non step TUF time constraints [2, 3]; here, we focus on step TUFs.) Jobs of a task have the same TUF. We use $U_i(\cdot)$ to denote task T_i ’s TUF, and use $U_{i,j}(\cdot)$ to denote the TUF of task T_i ’s j^{th} job. Completion of the job $J_{i,j}$ at a time t will yield a utility $U_{J_{i,j}}(t)$.

Each TUF $U_{i,j}$, $i \in \{1, \dots, n\}$ has an initial time $I_{i,j}$ and a termination time $X_{i,j}$. Initial and termination times are the earliest and the latest times for which the TUF is defined, respectively. We assume that $I_{i,j}$ is the arrival time of job $J_{i,j}$, and $X_{i,j} - I_{i,j}$ is the period P_i of the task T_i . If $J_{i,j}$ ’s $X_{i,j}$ is reached and execution of the corresponding job has not been completed, an exception is raised. Normally, this exception will cause $J_{i,j}$ ’s abortion and execution of exception handlers.

2.2. Energy Consumption Model

We consider Martin’s system-level energy consumption model [5, 10, 12] to derive the energy consumption per cycle. In this model, the CPU’s dynamic power consumption, denoted P_d , when operating at a frequency f can be given by $P_d = S_3 \times f^3$, where S_3 is constant.

Besides the CPU, there are also *other* system components that consume energy. Some system components such as main memory must operate at a fixed voltage, so their power can only scale with frequency. Their dynamic power consumption becomes $P_d = S_1 \times f$. Other components consume constant power with respect to the frequency, e.g., display devices. Thus, their power consumption can be represented as S_0 , where S_0 is constant. In practice, the quadratic term, $P_d = S_2 \times f^2$ is also included to account for the appearance of variations in DC-DC regulator efficiency across the range of output power, CMOS leakage currents, and other second order effects [5].

Summing the power consumption of all system components together, an equation for system-level energy consumption of a task is obtained as: $E_i = e_i \times (S_3 \times f^3 + S_2 \times f^2 + S_1 \times f + S_0)$. Thus, the expected energy consumption per cycle is given by:

$$E(f) = S_3 \times f^2 + S_2 \times f + S_1 + \frac{S_0}{f} \quad (1)$$

2.3. Problem Definition

We define a system-level performance metric, called *Utility and Energy Ratio* (UER) to integrate timeliness and energy consumption. The *system-level* UER is defined as the ratio of the total accrued utility to the total system-level energy consumption, i.e., $UER = \sum_{i=1}^n U_i / \sum_{i=1}^n E_i$.

Our scheduling criterion is to maximize system-level energy efficiency, which also implies maximizing the system-level UER. With non-negligible scheduler overhead, the factors that can influence system performance in-

clude scheduling algorithms, system load, the heights (values) of tasks' TUFs, and energy model settings. We consider energy-efficient UA algorithm EUA [12] and non-UA algorithms LaEDF [6], StaticEDF [6], and BaseEDF [12]. Their scheduling overhead and performance under different conditions will be compared and estimated.

3. Experimental Methodology

3.1. Description of Algorithms

The aggregate CPU demand of the concurrent tasks is $\sum_{i=1}^n \frac{C_i}{P_i}$ million cycles per second (MHz) [1]. We assume that the CPU can be operated at m frequencies $\{f_1, \dots, f_m | f_1 < \dots < f_m\}$, and define the system load as $Load = \frac{1}{f_m} \sum_{i=1}^n \frac{C_i}{P_i}$.

StaticEDF uses the constant speed given by $\sum_{i=1}^n \frac{C_i}{P_i}$, and a "ceiling" up to the lowest suitable frequency in $\{f_1, f_2, \dots, f_m\}$. It switches to the lowest frequency whenever there is no ready task. Combining the static schemes in [1] and [6], StaticEDF is the static optimal solution to the DVS problem of minimizing CPU energy consumption for the periodic task model with step TUFs and under the available frequency set. BaseEDF is EDF without DVS support and uses the highest frequency. LaEDF is the Look-ahead RT-DVS for EDF [6]. These three schemes only consider minimizing CPU's energy consumption, and abort infeasible tasks during overloads.

EUA considers the system-level energy consumption. At each scheduling event, EUA sorts jobs in the ready queue based on their UERs in a non-increasing order, and then inserts the jobs into a tentative schedule in the order of earliest termination time first. Upon inserting a job, EUA performs feasibility check, and ensures the tentative schedule is feasible by dropping some jobs; that is, the predicted completion time of each job left in the tentative schedule never exceeds its termination time.

Equation 1 implies that there is an optimal clock frequency (not necessarily the lowest) to minimize E_i of a task T_i . EUA refers to this optimal frequency, and performs stochastic DVS to calculate a frequency for the currently selected job i.e., the one at the head of the tentative schedule.

3.2. Modeling Scheduler Overhead

At each scheduling event, the total scheduler overhead O_t comprises the overhead of scheduling the ready job queue, i.e., queue-scheduling overhead O_q , and the overhead for possible frequency/voltage changes, i.e., switching overhead O_s . In number of cycles, O_q depends on the number of jobs n in the ready queue as well as the scheduling process, and O_s is decided by the specific processor.

O_s is zero when there is no frequency switching, and it varies with switching between different levels of speeds.

Switching from a lower frequency to a higher one usually requires more overhead than switching in the other direction, since this requires both frequency and voltage changes thus more cycles, while switching in the other direction may require only frequency changes but no voltage changes.

The time and energy overhead for O_t is decided by the frequency selected by the operating system to run the scheduler. Currently we are implementing our DVS scheme in a Linux kernel. Thus, in the preliminary work of this paper, we assume that O_s can be obtained through processor data sheets, and focus on queue-scheduling overhead O_q .

3.3. Parameter Settings

To study queue-scheduling overhead O_q , we implement the algorithms in our previously developed scheduling framework called *meta-scheduler* [4]. The meta-scheduler is an application-level framework for implementing UA schedulers on POSIX RTOSes, without RTOS modifications. We use QNX Neutrino 6.2.1 RTOS [7] in our study. All the algorithms are implemented in QNX without DVS—they are executed under a unique frequency. Thus, LaEDF, BaseEDF, and StaticEDF have the same overhead as EDF.

We select task sets with 10 to 20 periodic tasks. For each task, we assign a mean value to its WCEC. Its actual WCEC is uniformly distributed in the range of 0 and twice mean. The task period is also randomly generated using a similar uniform distribution. Finally, the task periods are scaled by a constant chosen such that the system load (i.e., utilization of the tasks) reaches a desired value. The system load is defined as $Load = \frac{1}{f} \sum_{i=1}^n \frac{C_i}{P_i}$, where f is the CPU frequency. The utility values of TUFs are randomly generated.

4. Experimental Results and Planned Work

4.1. Effect of Scheduling Overhead without DVS

To investigate the impact of different scheduler overhead for applications with a broad magnitude of time constraints, we introduce a metric called "Termination Time Miss Load" (XML). The XML of a scheduler is the load after which the scheduler begins to miss task termination times.

In the case of independent periodic tasks, EDF is assured to meet all deadlines during under-loads. Thus, an ideal EDF scheduler (without any overhead) should have XML as 1.0. Similarly, since EUA are equivalent to EDF during under-loads [12], it should have XML also equal to 1.0. However, an actual implementation in the RTOS incurs certain overhead. Furthermore, the scheduler overhead tends to manifest itself for shorter execution time tasks. That is, the shorter task execution times, the lower XML is. Thus, the relationship between XML and the average task execu-

tion time provides a way of evaluating the actual overhead of a particular scheduler.

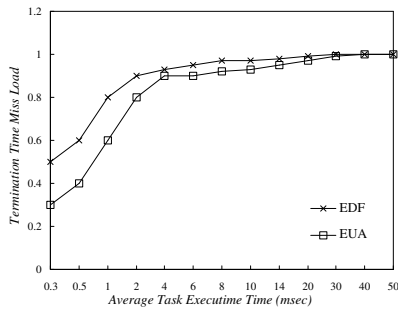


Figure 2. Termination Time Miss Load

We have measured the XMLs of LaEDF, BaseEDF, StaticEDF, and EUA implemented in QNX on a 450 MHz Pentium II PC. With a unique frequency, LaEDF, BaseEDF, and StaticEDF have identical overhead as EDF. Thus, plots for EDF and EUA are presented in Figure 2. The results show that our implementation of simple scheduling algorithms in QNX, such as EDF, is negligible for tasks with average execution time of as small as a few hundred microseconds. For complex scheduling algorithms, such as EUA, the average task execution time shall be at least in the magnitude of a few milliseconds so that the impact of scheduler overhead is negligible.

4.2. Planned Experiments

Results in Section 4.1 only consider queue-scheduling overhead O_q . To account for the impact of DVS overhead O_s on the system's energy saving and performance, we plan to evaluate the algorithms with full DVS supports.

We are implementing a prototype of EUA on the PC architecture. The hardware platform considered is a Hewlett-Packard Pavilion ze4700 with a single mobile AMD Athlon™ XP2500+ processor with a maximum operating frequency of 1.8 GHz. This processor features the PowerNow! technology and supports multiple frequencies. Further, its frequency and voltage can be adjusted dynamically under software control.

The prototype EUA is being implemented as a set of modules and patches that hook into the Linux kernel 2.4.18. We are using RedHat 7.3 (Valhalla) as the software platform. Although it is not a real-time operating system, Linux is easily extended through modules, and provides a robust development environment familiar to us. Our current implementation work is dealing with four major issues: (1) adding new system calls; (2) modifying the process control block; (3) adding the EUA module; and (4) modifying the standard Linux schedulers.

In our hardware platform, the laptop battery is removed and the system is running using the external DC power

adapter. We plan to use current probes and a digital oscilloscope to measure the actual power consumption of the laptop as the product of current and voltage supplied. We will analyze EUA's performance by comparing the measured energy savings with simulation results in [12].

5. Conclusions

In this paper, we study the impact of scheduler overhead of energy-efficient real-time scheduling algorithms. Preliminary experiments in this paper study the overhead of different algorithms without DVS support. These experiments consider important parameters such as task execution times for our future kernel implementation and measurements.

We are implementing a prototype of our algorithm on the PC architecture, and plan to measure the actual energy consumption and system performance. With our complete experiments, we anticipate to analyze factors affecting the impact of scheduler overhead, among which the most important ones are the energy model, system load, TUF parameters, and possibly the ratio of switching overhead to queue-scheduling overhead. In particular, we expect to identify the conditions under which TUF/UA algorithms have superior performance, despite their traditional high overhead.

References

- [1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *IEEE RTSS*, pages 95–105, December 2001.
- [2] R. Clark, E. D. Jensen, and et al. An adaptive, distributed airborne tracking system. In *IEEE WPDRTS*, volume 1586 of *LNCSS*, pages 353–362. Springer-Verlag, April 1999.
- [3] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time systems. In *IEEE RTSS*, pages 112–122, December 1985.
- [4] P. Li, B. Ravindran, S. Suhaib, and S. Feizabadi. A formally verified application-level framework for real-time scheduling on posix real-time operating systems. *IEEE Transactions on Software Engineering*, 30(9):613–629, September 2004.
- [5] T. Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, August 1999.
- [6] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SOSP*, pages 89–102, 2001.
- [7] QNX. QNX Neutrino RTOS. <http://www.qnx.com/>.
- [8] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. *IBM Journal of Research and Development*, 47(5/6):689–702, Sept./Nov. 2003.
- [9] C. Rusu, R. Melhem, and D. Mosse. Multi-version scheduling in rechargeable energy-aware real-time systems. In *ECRTS*, July 2003.
- [10] J. Wang, B. Ravindran, and T. Martin. A power aware best-effort real-time task scheduling algorithm. In *IEEE WSTFES/ISORC Workshop*, pages 21–28, May 2003.
- [11] H. Wu, B. Ravindran, and E. D. Jensen. Energy-Efficient, Utility Accrual Real-Time Scheduling Under the Unimodal Arbitrary Arrival Model. In *IEEE/ACM DATE*, pages 474–479, March 2005.
- [12] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Cpu scheduling for statistically-assured real-time performance and improved energy efficiency. In *IEEE/ACM CODES+ISSS*, pages 110–115, Sept. 2004.
- [13] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems. In *Proc. of ACM EMSOFT*, pages 64–73, Sept. 2004.