A User-Customizable Energy-Adaptive Combined Static/Dynamic Scheduler for Mobile Applications

Trolan C.-L. Ma and Kang G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109-2122
{trolanma, kgshin}@eecs.umich.edu

Abstract

In mobile applications, the energy consumed by OS and application tasks primarily comes from limited DC battery source, which imposes an upper bound to the amount of time available for execution of tasks. To achieve the best Energy-aware Quality-of-Service (EQoS), it is important to prioritize the scheduling of critical tasks over non-critical tasks to improve overall performance while extending the battery life. Using the Combined Static/Dynamic scheduler (CSD) in the EMERALDS operating system [9, 10] as a basis, we developed the Energy-Adaptive CSD (EA-CSD) with an energy-aware scheduling algorithm[1] that executes tasks to achieve effective use of limited energy by favoring low-energy and critical tasks. Our simulation of the EA-CSD shows that battery life can be extended up to about 100% with varying degrees of performance degradation of up to about 40%, and the actual values of both are fully customizable by the user through parametric adjustment.

1 Introduction

The Combined Static/Dynamic (CSD) scheduler proposed by Zuberi *et al.* [9, 10] minimizes scheduling overheads without considering the energy requirement and criticality of tasks. In CSD, all tasks to be scheduled are queued in one of the two queues: *Static-Priority* (SP) queue, which adopts mainly Rate-Monotonic (RM) scheduling, and *Dynamic-Priority* (DP) queue, which utilizes mainly Earliest-Deadline-First (EDF) scheduling. The assignment of each task to one of these queues is decided by the relative period of that task to other tasks, which, in turn, determines the task priority. Typically, the DP queue contains tasks whose periods are shorter and therefore, priorities are higher than those in the SP queue. Once tasks are in the DP queue, they are scheduled based only on the EDF policy. EDF prioritizes tasks with earlier deadlines over those with later deadlines and does not take into account such factors as energy demand and criticality of individual tasks in calculation of priorities.

There are mainly two undesirable effects from this non-energy-aware version of CSD. First, some of the important tasks with later deadlines may not be able to meet deadlines in a multi-task environment consisting mainly of short-period yet relatively less important tasks. Second, due to the lack of energy-awareness of CSD, it is possible that most energy-demanding tasks dominate the CPU over the less energy-demanding tasks with their shorter periods, thereby reducing the battery life substantially.

In this paper, we propose a scheduling algorithm implemented on the *Energy-Adaptive CSD* (EA-CSD) to achieve the dual goals of maximizing the battery life of portable applications while prioritizing critical tasks over non-critical ones as energy becomes scarce. We will first present the semantics and motivation of the parameters in it. We will then simulate a typical example scenario having a mixed workload of tasks with varying levels of energy consumption and criticality. This will be followed by an analysis of the simulation results, which basically demonstrate two important characteristics of EA-CSD: (1) the EA-CSD outperforms the non-EA-CSD in its ability to extend battery life; (2) the EA-CSD is capable of scheduling and completing more critical tasks before deadlines than non-EA-CSD in a bounded power-on period. Moreover, we will also study how the variation of some of the customizable parameters in the EA-CSD alters the outcomes of simulation and how they compare to their expected behaviors under the design goal of EA-CSD.

2 The Energy-Adaptive CSD (EA-CSD)

2.1 Assumptions

Before discussing the design details of EA-CSD, we would like to point out many of the assumptions that apply to the non-EA-CSD presented in [9, 10] still apply here. **A1** - **A5** are the original assumptions of the non-EA-CSD either explicitly stated or implied in [9, 10]. **A6** - **A9** are new assumptions made specifically for simulating the EA-CSD.

A1. All tasks are periodic. This is not a limiting assumption since non-periodic tasks can be handled by

1. The work reported in this paper was supported in part by the Airforce Office of Scientific Research.

introducing periodic servers.
A2. All tasks have deadlines equal to the end of the period in which they are released.
A3. CPU time is the only resource considered for each periodic task as in most scheduling theories.
A4. No precedence constraints exist between tasks. One can relax this assumption as done for non-energy-aware algorithms, but we will instead focus on energy conservation.
A5. Tasks are fully preemptable by other tasks ready to be scheduled and executed.
A6. The scheduler deals with a fixed periodic task set in the entire duration when there is still energy available. No new task is introduced in the middle of task execution.
A7. Tasks consume energy at constant rates during execution in the entire scheduling period.
A8. Task execution is governed by a reward function such that only tasks completed before deadlines have utility to the users.
A9. No energy replenishment is assumed in the middle of scheduling. (See Section 2.2.6 for the case of energy replenishment midway through scheduling)

2.2 The energy-adaptive scheduling algorithm

2.2.1 Energy-Adaptive Dynamic Priority (EADP)

Each periodic task under EA-CSD is assigned an EADP to represent its scheduling priority relative to other tasks in the system. All periodic tasks currently running, waiting in the queues and even those which still have not been released into the system at any instant, are assigned an EADP each. The EADP associated with a periodic task basically serves as a “mega-priority” in that it determines the task period which, in turn, decides the priority of the task in the system as seen by the original non-EA-CSD.

In what follows, we describe the calculation of EADP for each task and define some non-standard parameters used in the calculation.

$EADP_i(t)$ of a task i in a set of n periodic tasks at time t is a dimensionless quantity that can be calculated by:

$$EADP_i(t) = (1-r)\varepsilon_i\frac{e(t)}{E} + rI_i\left(\frac{E-e(t)}{E}\right)$$

where
r = user-defined ratio between weights of energy-oriented and criticality-oriented objectives $(0 \le r \le 1)$
I_i = criticality-oriented priority (COP) $(0 \le I_i \le 1)$
E = total estimated energy budget available (Unit: Joule)
$e(t)$ = residual energy at time t (Unit: Joule)
c_i = execution time of task i in each period (Unit: ms)
P_i = period length of task i (Unit: ms)
$U_i = c_i/P_i$ = utilization of task i in a period
w_i = power rating of task i (Unit: mW)

$$\varepsilon_i = 1 - U_i w_i \Big/ \sum_{j=1}^{n} U_j w_j$$

= energy-oriented priority (EOP) $(0 \le \varepsilon_i \le 1)$

2.2.2 Energy-Oriented Priority (EOP)

EA-CSD can schedule tasks in an energy-efficient manner that extends the battery life. This is accomplished by favoring tasks that have a lower energy utilization as compared to other periodic tasks in the system. EOP evaluates a task’s expected proportion in consuming the system energy among all the tasks in the system and assigns a task priority based on this proportion. The more energy expected to be consumed by a task, the lower the EOP is, and vice versa. First, each task must have its own power rating (w), which can be measured by a power gauging tool like the PowerScope [3]. However, this power rating for the task alone is not enough to represent how power-hungry a task is. If a task only executes for a negligible amount of time in each period, its contribution to energy consumption of the system would be minimal. So, the utilization (U) of the task is included to reflect its actual share of energy in the system. The product Uw of this task divided by the sum of Uw’s of all tasks represents the energy proportion consumed by this particular task among all the tasks in the system. One minus this energy proportion results in higher EOPs for tasks consuming less energy and lower EOPs for tasks expected to have high energy demand.

2.2.3 Criticality-Oriented Priority (COP)

COP associates each task with a priority between 0 and 1 indicating the importance of meeting its deadline. Tasks that are highly critical and whose deadline misses may result in catastrophe are assigned COPs close to 1. Tasks that are optional and do not result in visible degradation of performance if they execute less frequently or even miss deadlines are given COPs close to 0. For an application, COP of a task at any given time can be determined dynamically by a preprogrammed formula which takes into account all the periodic tasks presently in the system and evaluates the relative importance of each of them. If a new task is introduced into the system in the middle of a scheduling process, an interrupt is issued to the OS kernel and the COPs of all tasks are re-computed. However, this is assumed not to occur frequently as stated earlier. Besides, another way to determine COP is to have the users specify the COP of each task currently running themselves.

However, this is more difficult to implement and any erroneous criticality specification by the users may result in a disastrous consequence.

2.2.4 User-defined ratio *r*

The user-defined ratio *r* provides the means of communication between the user and the EA-CSD. Through this ratio, the user can instruct the EA-CSD whether it should place more emphasis in energy consumption or criticality of a task in deciding on the EADP of the task at any given instant. When $r = 0$, criticality is not a concern for the user and the EA-CSD does not take that into account when computing EADPs for all the tasks. The resulting schedule will purely favor tasks with low energy consumption, but not those with high criticality, generally leading to long battery life with low performance. When $r = 1$, energy is not so much an important issue to the user, so it is disregarded by the EA-CSD. The resulting schedule will have more critical tasks running than non-critical ones, even if the latter have low energy consumption. This typically leads to shorter battery life, but performance is enhanced since more critical tasks are executed.

One can adjust *r* to an intermediate value between 0 and 1. For $r \in [0, 0.5)$, the energy consumption of a task plays a more crucial part in calculation of EADP and we say *the energy-oriented objective is favored* as tasks with low energy demand are preferred by EA-CSD. But, for $r \in (0.5, 1]$, the criticality of a task dominates in the resulting EADP, and *the criticality-oriented objective is favored* as EA-CSD tends to schedule more critical tasks than low-energy tasks.

2.2.5 Weights of EOP and COP in EADP as energy wanes

The formula for computing EADP is composed of the sum of two terms. The *energy consumption term* $(1-r)\varepsilon_i e(t)/E$ encapsulates the EOP ε_i and the *criticality term* $rI_i(E-e(t))/E$ includes the COP I_i. These two terms are proportional to $e(t)/E$ and $(E-e(t))/E$, respectively, and the EOP and COP are therefore weighted differently as *e* changes over time *t*. At the begining when the energy level is *E*, the criticality term vanishes and the energy consumption term dominates in the calculation of EADP. Since *e* decreases as time elapses, one can expect that the weight of the EOP is gradually decreasing while COP is weighted more and more heavily. This demonstrates that as the energy level wanes, EA-CSD places a heavier weight on the criticality of tasks in its schedule and tends to schedule them more often, taking less into account the tasks' energy consumption. So, EA-CSD biases more toward critical tasks as energy becomes scarce. It ensures more critical tasks' deadlines to be met, and reduces the dependency between meeting critical tasks' deadlines and the residual energy level in the application.

2.2.6 When the EADPs are computed and updated

When the mobile application begins executing a task set, the EADPs of all tasks are computed using information about energy consumption and criticality for individual tasks. Then, during the scheduling of tasks in EA-CSD, the EADPs of tasks are regularly updated and the interval between two successive updates is directly controlled by the user through the adjustment of update factor $f \geq 1$. Suppose an update occurs now and the residual energy in the system is *e*. Then, when the residual energy reaches the next update energy e_u, the EADPs will be updated for all tasks, where e_u is computed by:

$$e_u = e - e/f$$

The first update occurs at $e_u = E - E/f$ instead of $e_u = E$.

The updates will contine with the next update energy calculated using the above formula at each update until the remaining energy *e* is lower than a threshold energy e_t calculated by:

$$e_t = EF_t$$

where *E* is the total energy estimated in the system initially and F_t is the low threshold factor, which is usually decided by the application developer based on the usual characteristics of tasks in the system. This threshold prevents the system from infinitely updating the EADPs of tasks since the formula for calculating the update energy will be executed without any stopping condition.

For example, if $E = 2500$, $f = 2$, $F_t = 0.1$, the updates will occur at $e = 1250, 625, 312.5$. The next update point after 312.5 (156.25) is smaller than $2500 \times 0.1 = 250$ and therefore EADPs will not get updated, starting from $e = 156.25$ and on.

If the energy is replenished in the middle of scheduling, an interrupt is issued to the OS and the total amount of energy after replenishment is assigned to *E*. The EADPs for all tasks are re-computed with the system's remaining energy *e* now reset back to *E*. Using the update factor *f* and low threshold factor F_t, e_u and e_t are re-computed and the scheduling session progresses in the usual way using the updated values of the above parameters after returning from interrupt routine, starting with energy level *E* and original task periods (or modified task periods based on amount of energy replenishment). This kind of replenishment is assumed to occur infrequently during scheduling since frequent interrupts involve costly context switches undesirable for real-time applications.

2.2.7 Stretching of task period

In CSD, task period is the pivotal factor in determining how often a task is scheduled. It decides whether a task should be in the DP or SP queue and, if it is in the latter, what priority the task should assume. The EADP of a task is utilized to determine whether a task's period should be stretched for less frequent execution and if so, how much it needs to be stretched. A task which has an EADP lower than the maximum EADP among all tasks at any given time is considered to be less eligible to run than the maximum-EADP task. Its period needs to be stretched based on the difference between its own EADP and the maximum EADP so that its actual priority is lowered than before (assuming it is in the SP queue). If the stretch is sufficiently large enough for a task originally in the DP queue, a task can be demoted from the DP queue to the SP queue and be scheduled based on their periods only after all the tasks in DP queue complete execution. By stretching the correct tasks, EA-CSD attempts to dynamically adjust the relative priorities of tasks in the system reflective of the user-specified goals, be it the energy-oriented objective or the criticality-oriented objective.

The amount of period stretch for a task at any update point is directly proportional to the difference between the task's own EADP and the maximum EADP among all tasks at that time. For example, suppose at the updating time of EADP, task *i* has EADP equal to $EADP_i$ and the maximum EADP among all the tasks is $EADP_{max}$. Then, the period stretched S_i for task *i* is calculated by: :

$$S_i = \frac{EADP_{max} - EADP_i}{EADP_i} \times k$$

where *k* is called the *stretch constant* and is application-dependent, and the new period for task *i* ($P_{i,new}$) is:

$$P_{i,\,new} = S_i + P_{i,\,old}$$

However, *tasks cannot be infinitely stretched in their periods.* An upper bound on task period as compared to the initial unstretched period is necessary because if tasks are infinitely stretched in their periods, they are invoked less frequently to an extent that causes the task to fail achieving their minimum required real-time performance. Audio and video applications are examples that need an upper bound for period in order to ensure a minimum, acceptable throughput. Thus, every task has an upper bound in task period calculated based on a maximum stretch factor F_s for the application. This factor can be individually assigned on a per-task basis or one factor can be applied to all tasks in a certain application. The resulting upper-bound period of a task P_{max} having initial period P_o scaled by maximum stretch factor F_s is calculated as:

$$P_{max} = P_o F_s$$

2.2.8 Addition of "Best-Effort Queue"

A third queue, called "Best-Effort queue", is added to EA-CSD. It accommodates tasks whose periods have been significantly stretched beyond the maximum-period task in the SP queue. Tasks in Best-Effort queue have the lowest priority in the task set and are not executed unless all the tasks in both the DP queue and SP queue have completely finished. Besides, there is no guarantee that these tasks in Best-Effort queue will ever be executed at all before the battery runs out of energy (hence the name "Best-Effort" due to its lack of guarantee).

3 A case study - The PalmPilot

3.1 Selection of periodic task set for simulation

We developed an EA-CSD simulator (written in C++) that fully models the EMERALDS OS environment and performs all the scheduling activities starting from the time when energy source is renewed to the point when energy is completely exhausted. It also displays statistics about the number of invocations and timely completions for each task at the end of the run, as well as the final battery life and the quality index as a measurement of overall performance of the system in the entire power-on period. Interesting conclusions are then drawn from these results, including how EA-CSD outperforms non-EA-CSD and how variation of parameters affects the performance of EA-CSD. Such a simulation offers much more flexibility to study the tradeoffs between different design parameters than actual implementation on EMERALDS OS.

The simulation of the EA-CSD is run on a typical task set consisting of a mixed workload with varying degrees of energy consumption rate and criticality. To this end, we have chosen several tasks in a PalmPilot Professional running PalmOS version 2.0.4 with energy consumption rates reported by Ellis [2]. The task set is selected from the application Hiker's Buddy, which takes NMEA protocol strings sent by a Global Positioning System (GPS) receiver and plots the hiker's current location on a map downloaded from the server.

The task set running under EA-CSD consists of a total of 7 periodic tasks. The first 4 tasks include primarily critical tasks integral to the running of the Hiker's Buddy. The other 3 tasks are less critical, with the first two tasks being some background batch jobs consisting of register-based computation loops and the last one being a periodic task for keeping the LCD display of the PalmPilot on.

Table 1. Profiles of the 7 selected periodic tasks in simulation of EA-CSD running on PalmPilot.

Task ID#	Description	Execution Time (ms)	Period (ms)	Criticality (COP)	Power Rating (mW)
1	Parsing NMEA sentences + receiving from GPS	5	100	0.80	90
2	Keeping serial line open to GPS	7	40	0.95	60
3	Downloading map from serial line	10	100	0.90	150
4	Searching map with memory-intensive computation	10	30	0.80	140
5	Background looping register-based computation job #1	6	50	0.30	125
6	Background looping register-based computation job #2	3	20	0.20	125
7	Keeping LCD display on	10	150	0.10	40

Table 1 shows the profiles of the 7 selected periodic tasks in different aspects including task execution time and period, as well as power rating and criticality. All the power ratings are directly extracted from the corresponding measurements in [2] for higher accuracy. All the remaining parameters are values closely approximating the actual situation while the Hiker's Buddy is concurrently running with some background jobs.

3.2 Time trace of task set

Figure 1 shows the time trace of the 7 periodic tasks running under EA-CSD in PalmPilot under the conditions: r = 0.9, f = 10, F_t = 0.1 and k = 10. The task queues are polled every 5000s and the length of the numbered bar is proportional to the period of the task with that ID number. Initially at time t = 0s, EADP is first computed. At t = 5000s, we observe a conspicuous change in the task period and priority position (assuming all tasks are in SP queue, which in reality is not necessarily true) for Task 6. Task 6 (COP = 0.2, w = 125) can be classified as a non-critical task with high energy consumption and is the kind of task biased against most by EA-CSD. Therefore, it is the first task which has its period stretched substantially from the outset. As a result of the stretch, its priority position goes down from 1st to 3rd in a period of 5000s. At t = 15000s, while Task 6 continues to have its period stretched as it becomes less eligible for scheduling, the period of another less critical and power-hungry Task 5 (COP = 0.3, w = 125) also gets stretched enough that it is demoted from 4th to 6th. At t = 20000s, Task 6 declines to 4th in priority. So, up to this point, the effect of EA-CSD in favoring tasks with high criticality and low energy consumption speaks for its ability to schedule tasks to satisfy both energy- and criticality-oriented objectives.

With r = 0.9, the criticality of tasks is of utmost importance in making scheduling decisions. Task 2 (COP = 0.95) and Task 3 (COP = 0.9) are the 2 most important tasks in the system as judged by the user. As a result, Task 2 and Task 3 do not suffer from any period stretch at all and remain two of the tasks cared most by EA-CSD. As Task 5 and Task 6 have their periods stretched past those of Task 2 and Task 3, Task 2 and Task 3 climb to 1st and 3rd from 3rd and 5th, respectively.

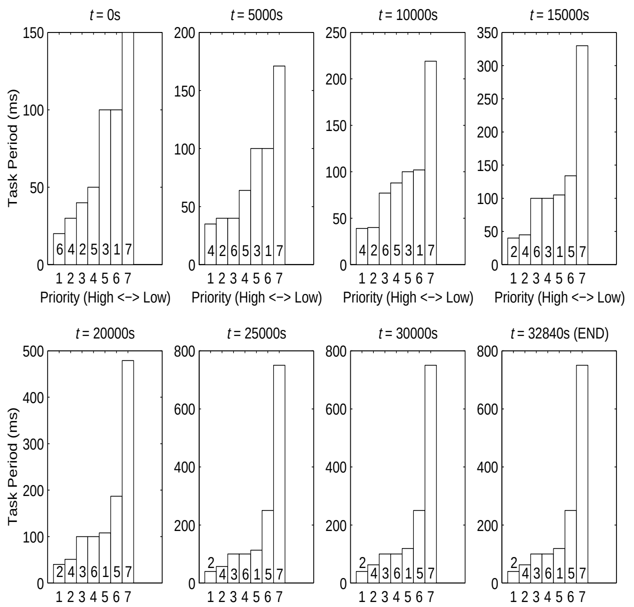


Figure 1. Time trace of task periods for the 7 periodic tasks running under EA-CSD in PalmPilot. Numbers inside (or on top of) bars indicate task ID# and tasks are arranged in ascending order of task period. The numbers under each diagram reflect task priorities if all tasks are in SP queue with "1" being the highest priority.

As energy becomes seriously scarce at the end of the power-on period, Task 5 and Task 6 basically have their periods constant and their priority positions fixed since they have reached the upper bounds of their periods. However, at this point, the periods of all tasks are already very indicative of the actual priority that each task deserves under energy-adaptive and criticality-aware scheduling. Least critical Task 7 (COP = 0.1) remains the task having the longest period all the way, which is just a manifestation that low criticality asks for more period stretching.

3.3 Task invocation and execution profiles

The bar chart in Figure 2 compares the numbers of tasks invoked and completed within deadlines over the entire duration when energy is still available on the PalmPilot, using the same parameters as that in Section 3.2. As we can see, task criticality dominates in the EA-CSD's decision to stretch task periods. So, two of the least critical tasks, Task 5 and Task 6, have a lower number of tasks invoked and completed in a timely manner in the EA version than in the non-EA version. The difference between the non-EA and EA versions is particularly pronounced for Task 6, which has a marginally lower COP than Task 5 and has dropped more than 50% in the number of invocations and timely completions. Task 7 has more invocations in the non-EA version, but its invocation is stifled in the EA version since it is the least critical task among all 7 tasks and its period gets stretched enough that it remains at the bottom of priority ranking throughout. On the other hand, the two most important tasks in the system, Task 2 and Task 3, both have a substantially larger number of invocations and completions before deadlines in EA version than in non-EA version. Particularly, Task 2 has the highest COP and lowest energy consumption among all tasks and is therefore heavily favored by EA-CSD. Its numbers of invocations and timely completions both have jumped up by more than 45%. Task 1 (COP = 0.8) and Task 4 (COP = 0.8) have slightly lower COPs and are not as heavily favored by EA-CSD as Task 2 and Task 3.

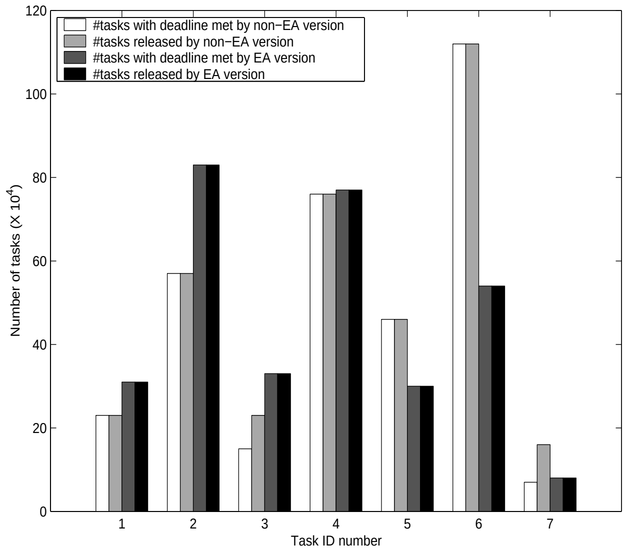


Figure 2. Number of tasks released and meeting deadlines during the entire power-on duration of the PalmPilot for non-EA- and EA-CSD.

3.4 Non-EA-CSD vs. EA-CSD

In order to compare the performances of non-EA-CSD and EA-CSD and see the value of incorporating energy-adaptivity in the latter, it is necessary to define certain performance metrics for comparison. We will use several performance metrics (**M1 - M3**) to compare the difference between the two versions of CSD as follows.

M1. *Battery life* provides a concrete measurement of how capable the two versions of CSD are in conserving energy by executing tasks that have lower energy consumption first.

M2. *The total sum of COPs of all instances of tasks that have completed before their deadlines in the whole power-on period* provides an idea of the total level of criticality of tasks executed and how the two versions fare against each other in favoring more critical tasks as energy wanes.

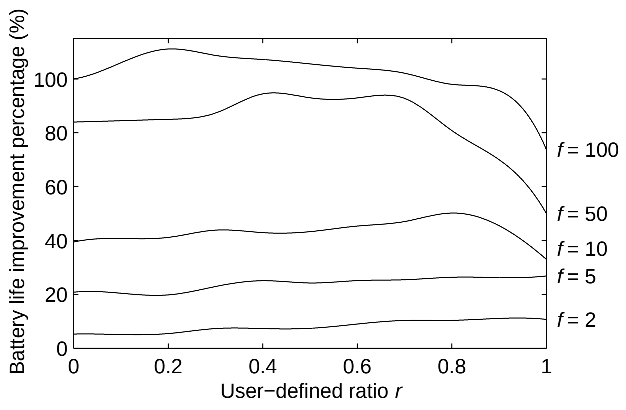
M3. *The total sum of COPs (**M2**) divided by the battery life (**M1**)* provides a quality index which is decided by the amount of COP obtained per time unit in the entire power-on period. This is necessary because the total sum of COPs in **M2** does not take into account the battery life over which the COPs are accumulated. As a result, the quality index is a metric indicative of the average performance for the application over time.

In all of the following simulation experiments, the effect of varying one parameter at a time in the EADP equation is analyzed to see the sensitivity of the above performance metrics.

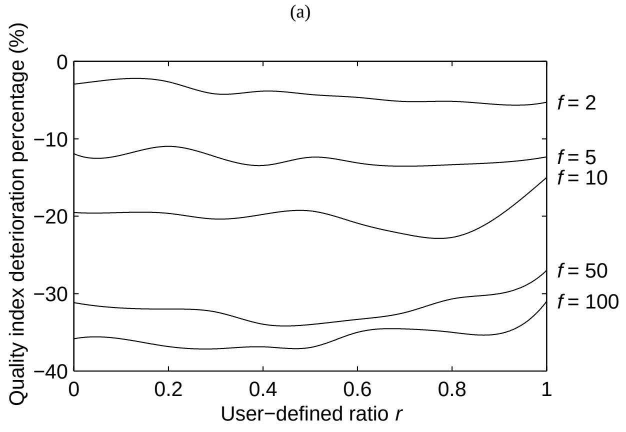
3.4.1 Effect of variation of user-defined ratio *r*

This part of simulation involves varying the user-defined ratio *r* to see if it can achieve the effect that we expect. The simulation uses $F_t = 0.1$, $k = 10$.

Figure 3 gives a comprehensive picture that is very close to our expectation. For the curves labelled with $f = 10$, $f = 50$, $f = 100$ in Figure 3(a), we see a general trend that the curves remain pretty much constant (or go up somewhat) and then undergo an abrupt slump as *r* increases from 0 to 1. When *r* is near 0, the energy consumption term dominates in deciding period stretch, and more energy is conserved than when *r* is large, where the energy consumption term is pretty much ignored. Therefore, more critical tasks are scheduled to meet

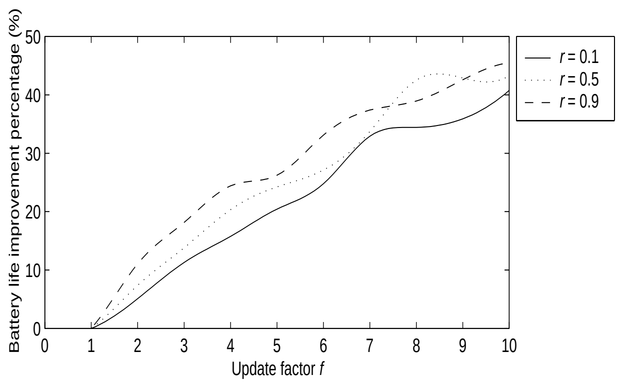


(a)

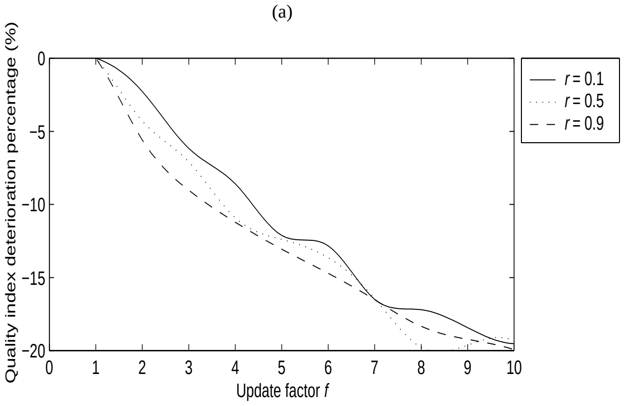


(b)

Figure 3. Battery life improvement and quality index deterioration with varying user-defined ratio *r* for EA-CSD. Curves only begin to look concave in both areas when *f* is greater than or equal to 10. Battery life improvement tops out 100% with 40% deterioration of performance when *f* = 100.



(a)



(b)

Figure 4. Battery life improvement and quality index deterioration with varying update factor *f* for EA-CSD. Battery life extends more and quality index declines as update factor increases starting from 1.

deadlines (as demonstrated in Figure 2 with *r* = 0.9) at the expense of energy-ignorant scheduling by EA-CSD as *r* approaches 1. Energy-ignorant scheduling simply leads to shorter battery life since some power-hungry tasks can use up more CPU cycles.

The near-constant slope in the intermediate range of these curves can be explained by the lingering dominance of the energy consumption term over the criticality term in EADP computation in the early stage of scheduling. In this range of *r*, since the weight of energy consumption term is more important than the criticality term in calculating EADP when energy is still near full, some high-energy tasks get their periods stretched early, saving much of the energy. The period stretching of primarily less critical tasks when energy is near exhausted occurs late or not frequently enough (due to relatively low update factor) so that the energy sacrificed in energy-ignorant scheduling is not sufficient to offset the energy savings earlier. This explains why the battery life improvement remains at a near-constant level in the intermediate range of *r*.

For the curves labelled with *f* = 10, *f* = 50, *f* = 100 in Figure 3(b), we see the reverse trend of the curves as compared to Figure 3(a) which reveals the complementary relationship between battery life and quality index. It also demonstrates the tradeoff between them and one cannot easily achieve both with a fixed amount of energy resources. As *r* nears 1, more critical tasks get completed before deadlines, adding to the total COP which translates to a higher quality index (less deterioration) within a shorter battery life.

In both Figures 3(a) and 3(b), for the curves with *f* lower than 10, the relatively low update factors do not provide sufficient number of updates of EADPs of tasks and therefore task periods are stretched less often. As a result, the energy consumption term still dominates in scheduling decisions. This explains why the curves are not as concave as those with higher update factors.

3.4.2 EA-CSD extends battery life at expense of performance degradation

From Figure 3(a), when *f* is greater than 1, the battery life is extended over the non-EA version no matter what the user-defined ratio is. However, an accompanying undesirable effect is the corresponding deterioration (negative percentage indicates deterioration in Figure 3(b)) of quality index, which represents a performance degradation over the non-EA version. The key point for the design of EA-CSD is to be able to let the users to determine the particular level of battery life and performance desired themselves through a set of user-adjustable parameters.

3.4.3 Effect of variation of update factor *f*

A. Battery life extension and performance deterioration

Next we will see the effect of changing *f* on battery life and quality index. When $f = 1$, from the formula of calculating the next update energy e_u, the EA-CSD does not update the EADPs of individual tasks in the system at all and hence tasks are scheduled in the same way as non-EA-CSD without energy-adaptive updates. Therefore, as *f* gets closer to 1, EA-CSD does not provide any significant extension of battery life and it basically degenerates to non-EA-CSD. This is demonstrated in Figure 4 where there is 0% improvement on battery life and 0% decline in quality index as compared to non-EA-CSD when $f = 1$.

As *f* increases from 1 to 10, battery life gets longer with a corresponding deterioration in performance. This can basically be attributed to the fact that more updates mean periods of tasks of high energy consumption are more often stretched, allowing those less power-hungry tasks to take over the CPU more frequently. As a result, battery life can be extended by as much as 40% with a moderate update factor of 10. The corresponding decline in quality index can be explained by the fact that the total amount of COP earned during the power-on period does not grow as fast as the rate of extension in battery life. A smaller amount of COP is achieved per time unit. Quality level declines by around 20% as *f* increases to 10.

B. Update overhead analysis

One may wonder from Figure 4 whether *f* can be *infinitely* increased to extend the battery life without bounds at the expense of a relatively less significant decline in performance. The answer to this question is "No". As exhibited in Figure 5(a) with $r = 0.9$, $F_t = 0.1$ and $k = 10$, as the update factor gets larger and larger to around 100, the battery life extension percentage starts to tail off until it more or less becomes flat or even rebounds back. This is due to the substantially larger overheads involved in calculating the EADP and updating task periods as the update factor rises. Similar situations also happen for performance deterioration in Figure 5(a) and increase percentage in total amount of COP earned in

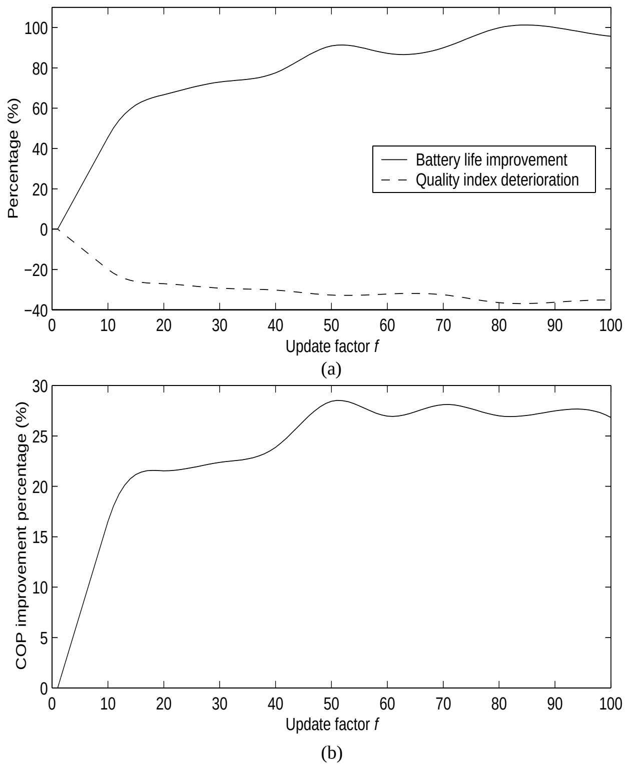


Figure 5. Battery life improvement, quality index deterioration and total COP improvement with varying update factor *f* for EA-CSD. Notice that all the curves tail off and even rebound back as update becomes more frequent due to large overheads involved in updating.

Figure 5(b).

The action of calculating the EADPs requires the parsing of all the tasks in all queues in the system and updating EADPs, which typically incur O(*n*) overheads for a total of *n* tasks in system. The check for maximum EADP among all *n* tasks can be done simultaneously by keeping track of the maximum EADP up to a point as EA-CSD calculates each EADP and only updates the maximum EADP in case there is a higher-EADP task. Then the EA-CSD parses through each task again in all queues and stretches the periods of tasks based on the maximum EADP just found, which incurs O(*n*) overheads again. The dominating factor in processing overheads comes from the final step in each update - sorting tasks in SP and Best-Effort queues based on the updated periods and demoting tasks from one queue to another if necessary. Assuming there are *m* tasks in the unsorted DP queue, there are ($n - m$) tasks in the sorted SP queue and Best-Effort queue. Using a simple sorting algorithm like an insertion sort, the average case sorting overhead is in the order of $O((n-m)^2)$. The demotion of tasks takes only negligible overheads compared to the sorting since it happens infrequently and not for each task in the queues. Any costs

Table 2. Steps in updating EADPs and periods of tasks and the ensuing sorting along with their associated orders of overhead costs.

Update Step Number	Description	Order of Overhead Costs
1	Calculation of EADPs and keeping track of maximum EADP thus far as EA-CSD goes through each task in queues	O(n) *
2	Update (stretching) of periods for all tasks in queues based on maximum EADP found in Step 1	O(n) *
3	Sorting of tasks based on updated periods in SP queue and Best-Effort queue and demotion of tasks between queues	O($(n\text{-}m)^2$) **
* Assuming there are totally n tasks in the system. ** Assuming there are m tasks in the DP queue and totally n tasks in the system. **Note:** Total overhead is in order of O($(n\text{-}m)^2$) overall since Step 3 dominates over Step 1, 2.		

associated with demotion are also amortized over a long period between successive updates. Table 2 summarizes all the essential steps in updating of tasks and their associated overhead costs.

For an update factor of lower than 100, total update overhead is small compared to the sum of the execution times of tasks and so, the overheads generated typically can be amortized over a long period of time between successive updates. However, as the update factor begins to exceed 100, the update overheads start to become comparable to the execution times of the tasks themselves, virtually overloading the system with extra energy-consuming tasks with useless values (of zero criticality). Therefore, the battery life extension percentage, performance deterioration percentage and COP increase percentage all gradually tail off or even rebound back as update factor increases beyond a certain threshold.

C. *f vs. r for customization*

As introduced in Sections 2.2.4 and 2.2.6, both the update factor f and user-defined ratio r are user-adjustable parameters of the EA-CSD. Comparing Figures 3 and 4, we can easily detect the difference of magnitudes of impact on battery life and performance of f and r. Generally speaking, f is a more sensitive user-adjustable parameter which produces relatively drastic changes in battery life and performance while r impacts slightly on those two, usually within only a limited range of values given a fixed f. For example, with r = 0.9, battery life improvement jumps surprisingly fast from 0% to about 45% as f increases from 1 to 10. On the other hand, with f = 10, the battery life improvement is 40% at r = 0 and drops to 30% at r = 1. It is therefore reasonable to use f as a coarse-grained adjustment parameter and r as a fine-grained parameter for fine-tuning the metrics given a coarse level achieved by f. After all, r still has the additional capability over f to decide the bias toward either energy-oriented or criticality-oriented objective.

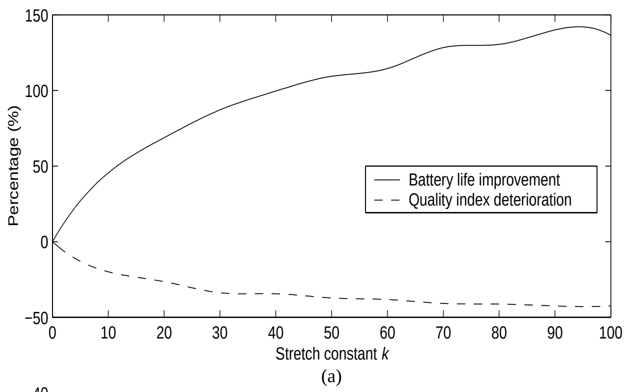
3.4.4 Effect of variation of stretch constant *k*

Another customizable parameter for application developers, but not users, is the stretch constant k. k can be used to determine the extent to which the task periods are stretched during each update for those tasks that have lower EADPs than the maximum EADP. The larger the value of k, the more the periods of tasks get stretched, and vice versa. In general, when k gets larger, it will take smaller number of updates for the EA-CSD to figure out the final convergent priority order (that does not change from a certain point on) of tasks based on the energy consumption and criticality of tasks in the system. Usually this implies a more substantial battery life improvement as k is increased. Since the increase in total amount of COP cannot keep pace with the rapidly improving battery life, there is a corresponding degradation of performance as k gets larger. These two phenomena are demonstrated in Figure 6(a), where the simulation is performed with r = 0.9, f = 10, F_t = 0.1.

In Figure 6(a), battery life lengthens and performance declines as k continues to increase from 0 to 90. From 90 to 100, the curves tail off and rebound back since each task is assocated with an upper-bound period length P_{max} as described in Section 2.2.7. Further increasing k does not effectively change the task periods as most long-period tasks have already reached their maximum periods. Figure 6(b) shows the increase percentage in the total amount of COP earned during the power-on period as k varies. It follows a similar trend as battery life improvement for the same reason.

4 Related work

Unlike EA-CSD, the majority of research efforts on energy conservation for mobile applications have rarely considered individual characteristics of application tasks, such as energy consumption and criticality, to distinguish between the different eligibilities of tasks for energy resources.



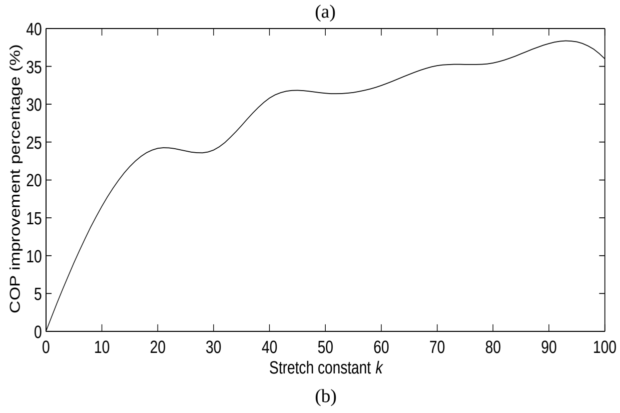


Figure 6. Battery life improvement, quality index deterioration and total COP improvement with varying stretch constant *k* for EA-CSD. Note that the curves tail off and even rebound back as *k* gets larger due to the upper bound imposed on period length .

Some researchers analyze the reduction in energy consumption by powering down unused components [1, 6, 8] or switching the system into sleeping mode in predicted idle periods [5, 7]. However, the former saves energy only on a few specific power-hungry hardware components while the latter relies too much on predictions in order to decide the mode of operation, which decreases the system's predictability and reliability in handling hard real-time tasks.

One of the more closely related work is Odyssey [4], which provides an API that supports energy conservation over multiple concurrent applications. However, even though it monitors energy usage for each individual application based on energy availability, it does not distinguish the difference between the criticalities of applications and assign them priorities for energy usage.

5 Conclusion

The EA-CSD in this paper is not aimed at providing an optimal algorithm for assigning task priority based on energy consumption and criticality of tasks. Rather, it represents a good approach that takes into account both the energy consumption and criticality of tasks in making scheduling decisions to accomplish the user-specified goals in conserving energy and favoring the execution of critical tasks. To our best knowledge, there are no known scheduler-based approaches that deal with the problem of energy conservation in mobile applications, let alone the problem of criticality maximization of tasks executed. EA-CSD does not advocate system-wide energy conservation by globally restricting energy usage of all tasks. Instead, it provides a built-in, automatic approach that assigns priorities based on intrinsic power and criticality characteristics of individual tasks to increase energy savings and ensure critical tasks are timely executed. User variability is realized when users have complete freedom and ease of choosing the level of performance desired in the above areas without constant manual monitoring and caring about internal details of implementation. EA-CSD is therefore a good starting point for further research and development in energy-adaptive scheduler.

References

[1] F. Douglis, P. Krishnan, B. Marsh, "Thwarting the power-hungry disk", In *Proceedings of 1994 Winter USENIX Conference*, Jan. 1994.

[2] C. Ellis, "The case for higher-level power management", In *the 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pp. 162-167, Rio Rico, AZ, Mar. 1999.

[3] J. Flinn, M. Satyanarayanan,"PowerScope: a tool for profiling the energy usage of mobile applications", In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2-10, New Orleans, LA, Feb. 1999.

[4] J. Flinn, M. Satyanarayanan, "Energy-aware adaptation for mobile applications", In *Proceedings of 17th ACM Symposium on Operating Systems Principles,* Dec. 1999.

[5] C.-H. Hwang, A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation", *IEEE/ACM International Conference on Computer-Aided Design*, pp. 28-32, 1997.

[6] R. Kravets and P. Krishnan, "Power management techniques for mobile communication", In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*, Oct. 1998.

[7] M. Srivastava, A. P. Chandrakasan, R. W. Broderson, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation", *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1, pp. 42-55, 1996.

[8] M. Stem, R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices", *IEICE Transactions on Communications, Special Issue on Mobile Computing*, 80(8), Aug. 1997.

[9] K. M. Zuberi, P. Pillai, K. G. Shin, "EMERALDS: a small memory real-time microkernel", In *Proceedings of 17th ACM Symposium on Operating Systems Principles, Published as Operating Systems Review,* 34(5): 277-291, Dec. 1999.

[10]K. M. Zuberi, "Real-time operating system services for networked embedded systems", PhD Thesis, University of Michigan, 1998.