Pruning-Based Energy-Optimal Device Scheduling for Hard Real-Time Systems*

Vishnu Swaminathan and Krishnendu Chakrabarty
Department of Electrical & Computer Engineering
Duke University
Durham, NC 27708, USA
{vishnus,krish}@ee.duke.edu

ABSTRACT

Dynamic Power Management (DPM) provides a simple, elegant and flexible method for reducing energy consumption in embedded real-time systems. However, I/O-centric DPM techniques have been studied largely for non-real-time environments. We present an offline device scheduling technique for real-time systems that generates an energy-optimal device schedule for a given task set while guaranteeing that all real-time deadlines are met. Our method takes as inputs a task set and a device-usage list for each task, and it schedules the tasks such that the energy consumed by the set of I/O devices is minimized. We compare our algorithm to an exhaustive enumeration method and show that the proposed algorithm is very efficient in terms of memory usage and computation time. We also present case studies to show that I/O-centric DPM methods can result in significant energy savings.

1. INTRODUCTION

Energy consumption is now recognized as an important design parameter for portable and embedded systems. A number of energy conservation techniques are currently being used in embedded systems. Several types of idle, standby and sleep modes are available in many processors. These modes are used to suspend or shut down the processor under low-load or no-load conditions. Compiler-level optimizations generate restructured machine code that allows for processor execution units to stay unused for long periods of time. These unused execution units can be shut down independent of each other when not in use, resulting in extended idle periods for the different processor modules. Power consumption of the CPU is also directly related to the clock speed. Many processors use a variable-speed clock that can be tuned to balance performance with power consumption [8].

Although hardware optimization significantly reduces energy consumption for embedded systems, even greater energy savings can be obtained if an intelligent software power management scheme is used that takes advantage of the power reduction capabilities of the hardware. The introduction of the ACPI standard in 1997 has enabled OS-based power management in several computer systems [1]. Power management by the OS is commonly referred to as *dynamic power management* (DPM).

I/O-centric DPM methods are either timeout-based, predictive or stochastic. Timeout-based schemes shut down devices when they have been idle for a pre-specified period of time. The next request for a device that has been shut down wakes the device up, which then processes the request. Stochastic methods use probability models to predict future idle periods [4].

Temporal behavior in real-time systems must be predictable to a high degree of accuracy. The consequence of a task missing a deadline can be disastrous. The critical nature of I/O device operation often results in devices staying powered up throughout system operation. Predictive and timeout-based schemes are therefore unsuitable for use in real-time systems. The inclusion of deadlines as an added design constraint makes DPM for real-time systems difficult.

Most research on real-time DPM techniques has been CPU-centric [7, 11, 15, 16, 18, 19, 20, 22]. An excellent survey of CPU-centric DPM is presented in [9]. Recent research also focuses on joint battery and variable-voltage scheduling in order to benefit from multiple power-saving strategies [14]. I/O-centric DPM has been studied extensively for non-real-time environments [4, 5, 13]. In [13], a device-utilization matrix keeps track of device usage and a processor-utilization matrix keeps track of processor usage of a given task. When the utilization of a device falls below a threshold, the device is put into the sleep state. In [5], devices with multiple sleep states are considered. Here too, the authors use a predictive scheme to shut down devices based on adaptive learning trees.

The above methods are not viable for real-time systems due to their inherently probabilistic nature. They assume there is no penalty associated with a delay in a task's start time. In real-time systems, a delay may result in a task

*This research was supported in part by DARPA under grant no. N66001-001-8946, in part by a graduate fellowship from the North Carolina Networking Initiative, and in part by DARPA and Army Research Office under Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the view of the DARPA and ARO agencies.

missing its deadline. Hence, it is critical to have I/O devices powered up and running at the correct times to guarantee that all application tasks meet their deadlines.

In [21], the authors have presented the first I/O-centric DPM algorithm for hard real-time systems. The online device scheduling algorithm in [21] takes as inputs a *predetermined* task schedule and the device-usage list for each task, and generates a sequence of sleep/working states for each device such that the energy consumed by the set of I/O devices is minimized. An online algorithm that runs in "real-time" must be fast and efficient. Due to its time-intensive nature, generating an energy-efficient *task schedule* (and thereby a device schedule) on the fly is often infeasible. With a predetermined task schedule, solving the device scheduling problem becomes much easier and can be performed in an online manner. However, using a predetermined task schedule implies that the start times of the tasks cannot be changed. By allowing task start times to be flexible, more energy-efficient device schedules can be generated, albeit offline. *Optimal* energy schedules can be generated by allowing both flexible start times and task reordering.

In [10], the authors model mode dependencies for a single resource and across multiple resources. This information is captured through the introduction of a mode dependency graph (MDG). An algorithm that is based on topological sorting is presented that takes as input a mode dependency graph and generates a set of valid mode combinations. A second algorithm then determines a sequence of modes for each resource such that all timing constraints are met and max-power requirements are satisfied for a given task set. In constrast, we are interested here in minimizing energy instead of satisfying max-power constraints. A schedule generated in [10] need not necessarily be an energy-optimal schedule for the task set. Furthermore, the work in [10] does not distinguish between I/O devices and processors. On the other hand, the model we assume is that of a set of periodic tasks executing on a single processor. These tasks use a given set of I/O devices. Given this scenario, the problem we attempt to solve is to minimize the energy consumed by the I/O devices.

In this paper, we present an offline DPM algorithm for hard real-time systems. We refer to the proposed algorithm as the energy-optimal device scheduler (EDS). For a given task set, EDS generates an energy-optimal device schedule by executing two jobs of the same task one after the other. This results in devices staying powered-down for extended periods of time.

The rest of the paper is organized as follows. In Section 2 we present a formal statement of our problem, including the terminology used in the paper and our assumptions. In Section 3, we develop the underlying theory. In Section 4, we describe our algorithm and in Section 5 we provide experimental results. Finally, in Section 6, we present our conclusions.

2. NOTATION AND PRELIMINARIES

In this section, we present the formal problem statement, our notation, and underlying assumptions.

We are given a task set $\mathcal{T}= \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n periodic tasks. Associated with each task $\tau_i \in \mathcal{T}$ are the following parameters:

- its *release* (or arrival) time a_i,
- its period p_i,
- its deadline d_i,
- its execution time c_i, and
- a *device usage list* L_i, consisting of all the I/O devices used by τ_i.

The hyperperiod H of the task set is defined as the least common multiple of the periods of all tasks. We assume that the deadline of each task is equal to its period, i.e., $p_i = d_i$. Associated with each *task set* $\mathcal{T}$ is a *job set* $\mathcal{J}= \{j_1, j_2, \ldots, j_l\}$ consisting of all the instances of each task $\tau_i \in \mathcal{T}$, arranged in ascending order of arrival time, where $l = \sum_{k=1}^{n} H/p_k$. Except for the period, a job inherits all properties of the task of which it is an instance.

The system also uses a set $K = \{k_1, k_2, \ldots, k_p\}$ of p I/O devices. Each device k_i has the following parameters:

- two power states—a low-power sleep state $ps_{l,i}$ and a high-power working state $ps_{h,i}$,
- a transition time from $ps_{l,i}$ to $ps_{h,i}$ represented by t^i_{wu},
- a transition time from $ps_{h,i}$ to $ps_{l,i}$ represented by t^i_{sd},
- power consumed during wake-up $P_{wu,i}$,
- power consumed during shutdown $P_{sd,i}$,
- power consumed in the working state $P_{w,i}$, and
- power consumed in the sleep state $P_{s,i}$.

We assume that requests can be processed by the devices only in the working state. All I/O devices used by a job must be powered-up before a task starts execution due to the real-time nature of the tasks. However, there are no restrictions on the time instants at which device states can be switched. Each job must complete its execution by its associated deadline. At time $t = 0$, all devices are powered up. We assume without loss of generality that $t^i_{wu} = t^i_{sd} = t_0$ and $P_{wu,i} = P_{sd,i} = P_0$ for all i. Note that this assumption does not limit the proposed algorithm in any way. The assumption only simplifies the presentation; the algorithm can be used for any set of parameter values. The power consumed by device k_i in the sleep state is less than the power consumed during transition, which in turn is less than the power consumed in the working state, i.e., $P_{s,i} < P_0 < P_{w,i}$. Most real-life devices satisfy this assumption. However, for devices that do not meet this requirement (hard disks, for example), one can use the concept of the *breakeven time* as explained in [7] to prevent erroneous device state transitions. Using breakeven times as a metric to determine when to switch device power states also helps in handling devices that possess multiple power states. This is made clearer in the next section. However, in this paper, to simplify the presentation, we only consider devices with two power states. The execution times $c_1, c_2, \ldots, c_n$ of the jobs are all greater than the maximum transition time t_0 of the devices. The energy consumed by device k_i is $E_i = P_{w,i}t_{w,i} + P_{s,i}t_{s,i} + mP_0t_0$, where m is the number of state transitions, $t_{w,i}$ is the total time spent by device k_i in the working state, and $t_{s,i}$ is the total time spent in the sleep state.

The problem we address is that of determining a set of start times for the jobs such that the the energy consumed by the set of devices $\sum_{i=1}^{p} E_i$ is minimized while ensuring that all tasks meet their deadlines. The task schedule provides a minimum-energy device schedule.

In the following section, we present our approach and the underlying theory.

Task	Arrival time	Completion time	Period (Deadline)	Device-usage list
τ_1	0	1	3	k_1
τ_2	0	2	4	k_2

Table 1: A simple task set.

	j_1	j_2	j_3	j_4	j_5	j_6	j_7
a_i	0	0	3	4	6	8	9
c_i	1	2	1	2	1	2	1
d_i	3	4	6	8	9	12	12

Table 2: List of jobs for Table 1.

3. THEORY

A straightforward approach to determine an energy-optimal schedule is to exhaustively enumerate all possible schedules for the job set and then select the one with the minimum energy. However, such an approach, being extremely time- and memory-intensive even for small job sets, is infeasible in practice.

Instead of exhaustive enumeration, we generate a schedule tree and iteratively prune branches when it can be guaranteed that the optimal solution does not lie along those branches. The schedule tree can be pruned based on two factors—time and energy. Temporal pruning is performed when a partial schedule results in a missed deadline deeper in the tree. This type of pruning is common to both the exhaustive enumeration technique and EDS. The second type of pruning—which we call *energy pruning*—is the central idea on which our offline scheduling algorithm is based. The remainder of this section explains the energy pruning algorithm. We illustrate this pruning method through the use of an example.

Table 1 describes a simple task set consisting of two tasks. While the device-usage lists of the tasks in this example are non-overlapping, the pruning algorithm also works for overlapping device-usage lists. Table 2 lists the instances of the tasks, arranged in increasing order of arrival. In this example, we assume a working power of 5 units, a sleep power of 1 unit, a transition power of 3 units and a transition time of 1 unit.

We now explain the generation of the tree and the pruning algorithm. A vertex v of the tree is represented as a 3-tuple (i, t, e) where where i is a job j_i and t is a valid start time for j_i. The third element e of the 3-tuple represents the total energy consumption up to time t. An edge z connects two vertices (i, t, e) and (k, l, m) such that job j_k can be successfully scheduled at time l given that job j_i has been scheduled at time t. A path from the root vertex to any intermediate vertex v has an associated order of jobs that is termed a *partial schedule*. A path from the root vertex to a leaf vertex constitutes a *complete schedule*. A *feasible schedule* is a complete schedule in which no job misses its associated deadline. We note that temporal pruning eliminates all *infeasible* partial schedules. Thus, every complete schedule is a feasible schedule. We now explain the generation of the tree in more detail.

The root vertex of the tree is a dummy vertex. It is represented by the 3-tuple $(0, 0, 0)$ that represents dummy job j_0 scheduled at time $t = 0$. At $t = 0$, a total of 0 units of energy have been consumed by the I/O devices. We next find all jobs that are released at time $t = 0$. The jobs that are released at $t = 0$ for our example are j_1 and j_2. Job j_1 can be scheduled to start at times $t = 0$, $t = 1$, and $t = 2$

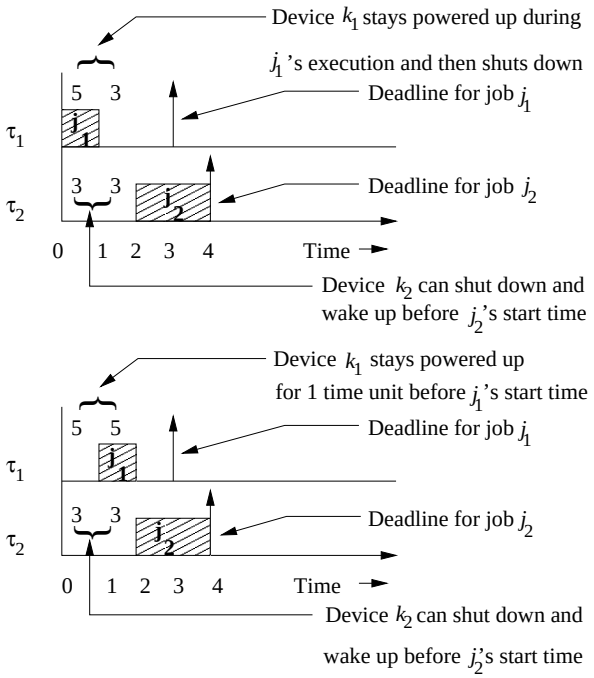

Figure 1: Calculation of energy consumption.

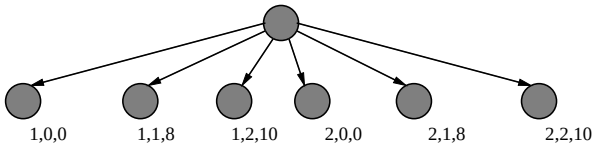

Figure 2: Partial schedules after 1 scheduled job.

without missing its deadline. We then compute the energy consumed by all the devices up to times $t = 0$, $t = 1$, and $t = 2$. The energy values are 0, 8 and 10 units, respectively (Figure 1)[1]. We therefore draw edges from the dummy root vertex to vertices $(1, 0, 0)$, $(1, 1, 8)$, and $(1, 2, 10)$. Similarly, job j_2 can be scheduled at times $t = 0, t = 1$ and $t = 2$ and the energy values are 0, 8, and 10 units respectively. Thus, we draw three more edges from the dummy vertex to vertices $(2, 0, 0), (2, 1, 8)$ and $(2, 2, 10)$. Note that job j_2 would miss its deadline if it were scheduled at time $t = 3$ (since it has an execution time of 2 units). Therefore, no edge exists from the dummy node to node $(2, 3, e)$, where e is the energy consumption up to time $t = 3$. Figure 2 illustrates the tree after one job has been scheduled. Each level of depth in the tree represents one job being successfully scheduled.

We then proceed to the next level. We examine every vertex at the previous level and determine which jobs can be scheduled next. By examining node $(1, 0, 0)$ at the previous level, we see that job j_1 would complete its execution at time $t = 1$. The only other job that has been released at $t = 1$ is job j_2. Thus, j_2 can be scheduled at times $t = 1$ and $t = 2$ *after* job j_1 has been scheduled at $t = 0$. The energies for these nodes are computed and edges are drawn from $(1, 0, 0)$ to $(2, 1, 10)$ and $(2, 2, 14)$. Similarly, examining vertex $(1, 1, 8)$ results in vertex $(2, 2, 16)$ at level 2. The next

[1]By assuming that all $c_i > max_k\{t_0\}$ and using breakeven times as the device transition metric, EDS can handle devices with multiple power states and still generate energy-optimal schedules.

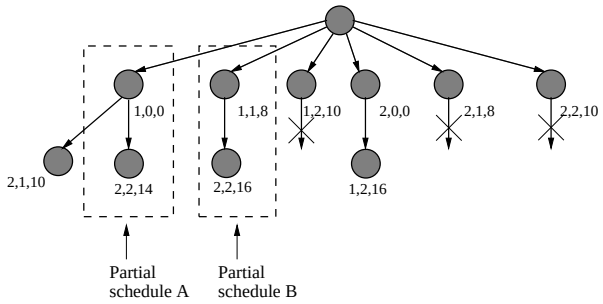

Figure 3: Partial schedules after 2 scheduled jobs.

vertex at level 1—$(1, 2, 10)$—results in a missed deadline at level 2. If job J_1 were scheduled at $t = 2$, it would complete execution at time $t = 3$. The earliest time at which j_2 could be scheduled is $t = 3$; however, even if it were scheduled at $t = 3$, it would miss its deadline. Thus, scheduling j_1 at $t = 2$ does not result in a feasible schedule. This branch can hence be pruned. Similarly, the other nodes at level 1 are examined and longer partial schedules are created. Figure 3 illustrates the schedule tree after two jobs have been scheduled. The edges that have been crossed out represent branches that are not considered due to temporal pruning.

At this point, we note that vertices $(2, 2, 14)$ and $(2, 2, 16)$ represent the same job (j_2) scheduled at the same time $(t = 2)$. However, the energy consumptions for these two vertices are different. This observation leads to the following theorem:

THEOREM 1. *When two vertices at the same tree depth representing the same job being scheduled at the same time can be reached from the root vertex through two different paths, and the orders of the previously scheduled jobs along the two partial schedules are identical, then the partial schedule with higher energy consumption can be eliminated without losing optimality.*

Proof: Let us call the two partial schedules at a given depth Schedule A and Schedule B, with Schedule A having lower energy consumption than Schedule B. We first note that Schedule B has higher energy consumption than Schedule A because one or more devices have been in the powered-up state for a longer period of time than necessary in Schedule B. Assume that i jobs have been scheduled, with job j_i being the last scheduled job. Since we assume that the execution times of all jobs are greater than the maximum transition time of the devices, the state of the devices at the end of job j_i will be identical in both partial schedules. By performing a time translation (mapping the end of job j_i's execution to time $t = 0$), we observe that the resulting schedule trees are identical in both partial schedules. However, all schedules in Schedule B after time translation will have an energy consumption that is greater than their counterparts in Schedule A by an energy value E_δ, where E_δ is the energy difference between Schedules A and B. Finally, it is easy to see that the energy consumed *during* job j_i's execution in Schedule A is less than or equal to its execution in Schedule B. This completes the proof of the theorem. □

The application of this theorem to the above example results in partial schedule B in Figure 3 being discarded. As one proceeds deeper down the schedule tree, there are more vertices such that the partial schedules corresponding to the paths to them from the root vertex are identical. It is this

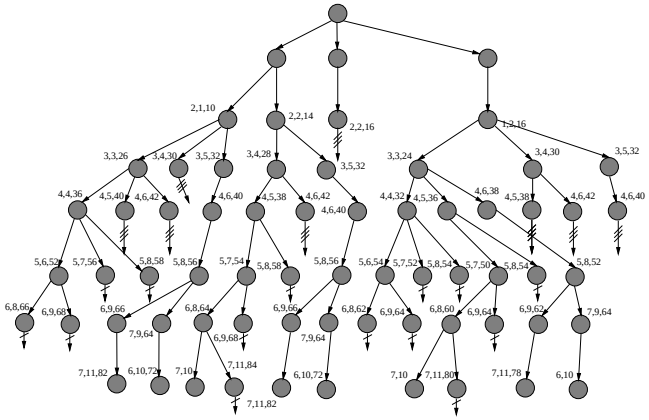

Figure 4: Complete schedule tree.

"redundancy" that allows for the application of Theorem 1, which consequently results in tremendous savings in memory while still ensuring that an energy-optimal schedule is generated. By iteratively performing this sequence of steps (vertex generation, energy calculation, vertex comparison and pruning), we generate the complete schedule tree for the job set. The complete tree is shown in Figure 4. We have not shown paths that have been temporally pruned. The edges that have been crossed out with horizontal slashes represent energy-pruned branches.

In the following section, we present and explain the EDS algorithm.

4. THE EDS ALGORITHM

In this section, we present the EDS algorithm in pseudocode form and briefly explain its operation.

Figure 5 describes the algorithm in pseudocode form. EDS takes as input the job set $\mathcal{J}$. The algorithm generates all possible minimum energy schedules for the given job set. The algorithm operates as follows. The time counter t is set to 0 and the openList is initialized to contain only the root vertex $(0, 0, 0)$ (Lines 1 and 2). In lines 3 to 10, every vertex in the openList is examined and nodes are generated at the succeeding level. Next, the energy consumptions are computed for each of these newly generated vertices (Line 11). Lines 15 to 20 correspond to the pruning technique. For every pair of replicated vertices, the partial schedules are checked and the one with the higher energy consumption is discarded. Finally, the remaining vertices in the currentList are appended to the openList. The currentList is then reset. This process is repeated until all the jobs have been scheduled, i.e., the depth of the tree equals the total number of jobs (Lines 25 to 28). Note that several minimum-energy schedules can exist for a given job set. EDS generates all possible unique schedules with minimum energy for that schedule. One final comparison of all these unique schedules results in the set of schedules with the absolute minimum energy.

EDS generates optimal energy solutions only for task sets for which at least one non-preemptive schedule exists. However, there exist task sets that do not satisfy this requirement. Such task sets, though, do not consist entirely of periodic tasks; instead, they are a mix of periodic and sporadic tasks. For mixed task sets, we are not aware of any optimal scheduling algorithm that minimizes I/O device energy for hard real-time systems. EDS is a first step towards a more

```
Procedure OFFLINE(J, l)
J: Job set.
l: Number of jobs.
openList: List of unexpanded vertices.
currentList: List of vertices at the current depth.
t: time counter.
begin
1.  Set t = 0; Set d = 0;
2.  Add vertex (0,0) to openList;
3.  for each vertex v = (j_i, time) in openList {
4.      Set t = time + c_i;
5.      Find set of all jobs J' released up to time t;
6.      for each job j ∈ J' {
7.          if j has been previously scheduled
8.              continue;
9.          else {
10.             Find all possible scheduling instants for j;
11.             Compute energy for each generated vertex;
12.             Add generated vertices to currentList;
13.         }
14.     }
15.     for each pair of vertices v1, v2 in currentList {
16.         if v1 = v2 and
                partial schedule(v1) = partial schedule(v2) {
17.             if E_v1 > E_v2
18.                 Prune v1;
19.             else
20.                 Prune v2;
21.         }
22.     }
23.     Add unpruned vertices in currentList to openList;
24.     Clear currentList;
25.     Increment d;
26.     If d = l
27.         Terminate.
28. end
```

Figure 5: Pseudocode description of EDS.

Task	Execution time	Period (Deadline)	Device list
τ_1	1	4	k_1
τ_2	3	5	k_2

Table 3: Task set 1.

general device scheduling algorithm for mixed workloads.

We next present experimental results and illustrate the efficiency of our scheduling algorithm.

5. EXPERIMENTAL RESULTS

We evaluated EDS for several periodic task sets with varying hyperperiods and number of jobs. We compare the memory requirement of the tree with the pruning algorithm to the memory requirement of the tree without pruning. Memory requirement is measured in terms of the number of nodes at every level. We also illustrate the efficiency of the pruning algorithm by presenting the total number of possible valid schedules with and without pruning.

Our first task set consists of two tasks with a hyperperiod of 20. This simple task set is described in Table 3. For the sake of simplicity, we have assumed that these tasks use independent devices. We have assumed, also for the sake of simplicity, that the device characteristics are the same for both devices. We assume a working power of 5 units, a transition power of 3 units and a sleep power of 1 unit for both devices.

The task set in Table 3 results in a job set consisting of 9 jobs, as shown in Table 4.

If all devices are powered up throughout the hyperperiod,

	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
a_i	0	0	4	5	8	10	12	15	16
c_i	1	3	1	3	1	3	1	3	1
d_i	4	5	8	10	12	15	16	20	20

Table 4: Job set corresponding to Table 3.

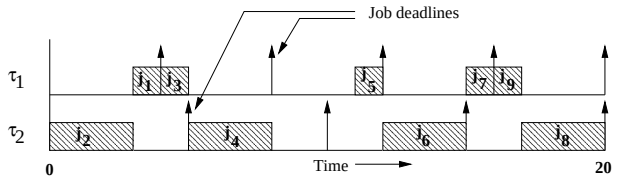

Figure 6: Optimal task schedule for Table 3.

the energy consumed by the I/O devices for this task schedule is 200 units. Figure 6 shows an optimal task schedule generated using EDS. The energy consumption of the optimal task (device) schedule is 134 units, resulting in a 33% reduction in energy consumption.

We see that minimum energy will be consumed if (i) the time for which the devices are powered up is minimized, (ii) the time for which the devices are shutdown is maximized, and (iii) the number of device transitions is minimized.

The rapid growth in the state space of the exhaustive enumeration method is made evident in Table 5. We see that the number of vertices generated by the exhaustive enumeration technique is enormous, even for a relatively small task set as in Table 3. In contrast, EDS requires far less memory. We see that the total number of vertices for EDS is 87% less than that of exhaustive enumeration.

By changing the periods of the tasks in Table 3, we generated several job sets whose hyperperiods ranged from $H = 20$ to $H = 40$ with the number of jobs J ranging from 9 to 13. For job sets larger than this, the exhaustive enumeration method failed due to lack of computer memory. These experiments were performed on a Sun workstation with 512 MB of RAM and 2 GB of swap space. The results are shown in Table 6. A comparison of the execution times of the two methods in Table 7.

Finally, we compare EDS to the LEDES algorithm from [21]. We use an example job set whose details can be found in [21] (Table 2). The job set consists of eight jobs which use five I/O devices. The properties of the devices are described in Table 8. This job set consumes 583 units using LEDES. When compared to the device schedule with all devices powered-up, this is an energy reduction of 50%. Using EDS, the energy consumption drops to 481 units, resulting in an added energy savings of 17%. The increased savings in energy through the use of EDS arises from the facts that LEDES (i) takes as input a predetermined task schedule, and (ii) cannot modify task start times in any way. With EDS, this restriction is relaxed, thereby making it more flexible.

Our final data set is shown in Table 9. The device parameters are described in Table 8. These parameters have been taken from real-life devices. This task set consists of six tasks that are used in a distributed sensor network, which serves as a typical example of an energy-constrained real-time system. It consists of a data cache application, a signal processing task, an RF-modem tasks and a few housekeeping processes. The energy consumption using LEDES is 60 units. With EDS, the energy consumption drops to 45 units, resulting in savings of 25%.

Tree depth i	No. of Vertices at depth i		Memory savings
	No pruning	Pruning	
1	7	7	0%
2	4	3	25%
3	20	13	35%
4	18	7	61%
5	76	23	69%
6	156	40	74%
7	270	33	87%
8	648	50	92%
9	312	17	94%
Total	1512	193	87.2%

Table 5: Percentage memory savings.

Job set	No. of vertices		No. of feasible schedules	
	Exhaustive	Pruned	Exhaustive	Pruned
H=20,J=9	1512	238	312	17
H=30,J=11	252931	4110	121016	836
H=35,J=12	2,964,093	13818	1,668,673	3024
H=40,J=13	23,033,089	43783	14,464,680	8123
H=45,J=14	DNT	84107	DNT	17187
H=55,J=16	DNT	592,091	DNT	112,363
H=60,J=17	DNT	959,872	DNT	208,741

DNT: Did not terminate after 24 hours

Table 6: Comparison of memory requirements.

6. CONCLUSIONS

In this paper, we have presented a novel offline device scheduling algorithm that generates energy-optimal schedules for a given periodic task set while also guaranteeing that no task deadline is missed. We have shown that our pruning technique results in tremendous memory savings, thereby allowing us to schedule a large number of jobs in an efficient manner. The EDS algorithm schedules multiple instances (jobs) of the same task to execute successively and thus minimizes the number of device state transitions. Our results show that DPM for I/O devices can provide significant energy savings in real-time systems.

7. REFERENCES

[1] Advanced Configuration and Power Interface (ACPI), *http://www.teleport.com/∼acpi.*

[2] Analog Devices Multiport Internet Gateway Processor. *http://www.analog.com.*

[3] AMD Ethernet Controllers PCNet Family, *http://www.amd.com/products/npd/overview/21135.pdf*

[4] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management", *IEEE Trans. CAD,* vol. 16, no. 6, pp. 813–833, June 1999.

[5] E.-Y. Chung, L. Benini and G. De Micheli, "Dynamic power management using adaptive learning tree", *Proc. Intl. Conf. CAD,* pp. 274–279, 1999.

[6] Fujitsu MHL2300AT Hard Disk Drive. *http://www.fujitsu.jp/hypertext/hdd/drive/overseas/mhl2xxx/mhl2xxx.html.*

[7] C. Hwang and A. C-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation", *Proc. Intl. Conf. CAD,* pp. 28–32, 1997.

[8] Intel Speedstep technology. *http://www.intel.com*

[9] N. K. Jha, "Low power system scheduling and synthesis", *Proc. Intl. Conf. CAD,* pp. 259–263, 2001.

[10] D. Li, P. Chou, and N. Bagherzadeh, "Mode selection and mode-dependency modeling for power-aware embedded systems," *Proc. Asia South Pacific Design Automation Conference,* January 2002. pp. 697–704.

[11] J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems", *Proc. Intl. Conf. CAD,* pp. 357–364, 2000.

[12] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM,* vol. 20, pp. 46–61, 1973.

[13] Y-H. Lu, L. Benini and G. De Micheli, "Operating system directed power reduction", *Proc. Intl Conf. Low-Power Electronics and Design,* pp. 37–42, 2000.

[14] J. Luo and N. K. Jha, "Battery-aware static scheduling for distributed real-time embedded systems", *Proc. DAC,* pp. 444–449, 2001.

[15] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems", *Proc. Symp. Operating Systems Principles*, pp. 89–102, 2001.

[16] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors", *Proc. DAC,* pp. 828–833, 2001.

[17] RamBus RIMM. *http://www.rambus.com*

[18] Y. Shin, K. Choi and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors", *Proc. Intl. Conf. CAD,* pp. 365–368, 2000.

[19] A. Sinha and A. Chandrakasan, "Energy efficient real-time scheduling", *Proc. Intl. Conf. CAD,* pp. 458–463, 2001.

[20] V. Swaminathan and K. Chakrabarty, "Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems", *Proc. Asia South Pacific Design Automation Conf.,* pp. 251–254, 2001.

[21] V. Swaminathan, K. Chakrabarty and S. S. Iyengar, "Dynamic I/O power management for hard real-time systems", *Proc. Intl. Symp. Hardware/Software Co-Design (CODES),* pp. 237–243, 2001.

[22] M. Weiser, B. Welch, A. Demers and S. Shenker, "Scheduling for reduced CPU energy", *Proc. Symp. Operating System Design and Implementation,* pp. 13–23, 1994.

[23] Zoom Embedded Modem Family, *http://www.zoom.com/datasheets/embedded.shtml.*

Job set	Execution time	
	Exhaustive	EDS
H=20,J=9	< 1s	< 1s
H=30,J=11	2.3s	< 1s
H=35,J=12	28.2s	4.6s
H=40,J=13	7m 15s	35.2s
H=45,J=14	DNT	2m 29.5s
H=55,J=16	DNT	2h 24m 15s
H=60,J=17	DNT	5h 10m 23.2s

DNT: Did not terminate after 24 hours

Table 7: Comparison of execution times.

Device no.	Device	$P_{w,i}$	$P_{s,i}$
k_1	HDD [6]	2.3W	.80W
k_2	Modem [23]	1W	.625W
k_3	NIC [3]	1W	.3W
k_4	DSP [2]	1W	.46W
k_5	RDRAM [17]	1W	.45W

Table 8: Power consumption of various devices.

Task	Arrival time	Completion time	Deadline (Period)	Device list
Data cache	0	3	35	k_1, k_2
File write	0	5	35	k_1
DSP	0	7	35	–
RF Transmit	0	2	35	k_2
Housekeeping	0	7	35	k_1, k_3
Memory copy	0	3	35	–

Table 9: A distributed sensor network task set.