Adaptive Energy Management Scheme in Real-Time Energy Harvesting Embedded Systems

shruti khare
shrutikhare30@gmail.com
Adaptive Energy Management Scheme in Real-Time Energy Harvesting Embedded Systems

Shruti Khare

BS, University of Hartford, 2010

A Thesis
Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science At the University of Connecticut
June 2012
Master of Science Thesis

Adaptive Energy Management Scheme in Real-Time Energy Harvesting Embedded Systems

Presented by
Shruti Khare, B.S

Major Advisor __________________________________________________________
Lei Wang

Associate Advisor _______________________________________________________
John Chandy

Associate Advisor _______________________________________________________
Mohammad Tehranipoor

University of Connecticut
June 2012
Abstract

The purpose of this thesis was to develop an online adaptive energy management scheme using Adaptive Forward Prediction (AFP) algorithm for solar energy prediction. Two energy prediction schemes were used, namely Exponentially Weighted Moving Average (EWMA) and AFP, to schedule all the tasks with least deadline miss rate. The AFP scheme has a mean relative error of 6-10% which is much lower than exponentially weighted moving average (EWMA) algorithm with an error of 30%. The large difference in the error percentage between the two prediction algorithms is due to the adaptive nature of AFP as it tracks small changes in input signal and dynamically adjusts itself to the changes incurring smaller error percentage. On the other hand, EWMA algorithm requires prior knowledge of the signal from the previous day which doesn’t remain constant, thus introducing large prediction error. The proposed algorithm is executed in two parts- Firstly, an offline energy management algorithm using EWMA was developed which decides the speed at which the task should be executed depending on the energy availability. Secondly, using AFP algorithm the tasks speed and start time was dynamically adjusted according to the difference in the energy predicted by both the prediction algorithms during runtime. The results show that by using the proposed adaptive technique the deadline miss rate of the tasks was decreased by 15-30% in addition to the results accomplished by initial scheduling depending on the extra/less amount of energy predicted by AFP.
Acknowledgements

I would like to take this great opportunity to thank everyone who supported me throughout the journey of completing my Master’s Thesis; without their support this would have not been possible.

First and foremost, I would like to express my sincere gratitude towards my Major advisor Dr. Lei Wang for all his constant support and valuable advice. Every time after our weekly meetings I felt highly motivated and encouraged. I really appreciate him for being so flexible and letting me gain knowledge from few other projects.

I would like to acknowledge my lab mate Jianwei Dai who shared his ideas with me and was always ready to help me with any technical difficulties.

Finally, I would also like to thank my parents who stood beside me at all times being a source of strength including financial, spiritual and moral support. A special thanks to my sister who always reminded me of my dreams and encouraged me to achieve my goals.
# Table of Contents

Abstract .............................................................................................................................. iii  
Acknowledgments.............................................................................................................. iv  
List of Figures ................................................................................................................... vii  
List of Tables ................................................................................................................... viii  
Symbols and Acronyms ..................................................................................................... ix  

1. Introduction ................................................................................................................. 1  
   1.1 What is Real-Time Energy Harvesting Systems?........................................... 1  
      1.1.1 Real-Time Systems ................................................................. 1  
      1.1.2 Energy Harvesting Systems ................................................. 1  
   1.2 Real-Time Workload and Assumptions....................................................... 3  
      1.2.1 Task Model ......................................................................... 3  
      1.2.2 Real-Time Task Attributes.................................................... 4  
      1.2.3 Assumptions on task set in our algorithm .............................. 5  
   1.3 Real-Time Scheduling .................................................................................... 6  
      1.3.1 Static Scheduling ................................................................... 6  
      1.3.2 Dynamic Scheduling ............................................................. 6  
   1.4 Power Management in Real-Time System ..................................................... 7  
      1.4.1 Dynamic Power Management ................................................... 7  
      1.4.2 Dynamic Voltage/Frequency Scaling ...................................... 8  

2. This thesis .................................................................................................................... 9  
   2.1 System Model and Assumptions................................................................. 9  
      2.1.1 Energy Harvesting Unit (EHU) ............................................. 10  
      2.1.2 Energy Storage Unit (ESU) ................................................... 10  
      2.1.3 Energy Dissipation Unit (EDU) .............................................. 11  
   2.2 Related Research ........................................................................................... 13  
   2.3 Motivation ...................................................................................................... 14  
   2.4 Objective ...................................................................................................... 15  
   2.5 Main Contributions ...................................................................................... 16  

3. AFP Prediction Algorithm ......................................................................................... 18  
   3.1 Theory ...................................................................................................... 18  
   3.2 Simulations ................................................................................................... 21  

4. The AEMS Algorithm ............................................................................................. 24  
   4.1 Initial Offline Schedule using EWMA ...................................................... 24  
      3.1.1 Motivational Example ............................................................ 32  
   4.2 Adaptive Scheduling using AFP ............................................................... 36  
      4.2.1 Extra Energy Available (Ed is positive) ................................. 38  

4.2.1.1 Motivational Example.................................................... 43
4.2.2 Less Energy Available ($Ed$ is negative)............................ 46
  4.2.2.1 Motivational Example................................................... 48
  4.2.3 Assumptions on the size of the capacitor............................ 49

5. Results and Discussions........................................................................................................... 51

6. Conclusion and Future Work.................................................................................................. 55

REFERENCES ............................................................................................................................. 56

APPENDICES

APPENDIX A: Code for initial scheduling using MATLAB ......................... 61
APPENDIX B: Code for adaptive scheduling for $Ed>0$ using MATLAB ........ 63
APPENDIX C: Code for adaptive scheduling for $Ed<0$ using MATLAB .......... 65
List of Figures

Figure 1: Real-Time task with its attributes ................................................................. 4
Figure 2: A Real-Time Energy Harvesting Embedded System Model .......................... 10
Figure 3: Illustration of prediction interval (slots) in EWMA ..................................... 13
Figure 4: Forward Prediction Filter ............................................................................. 19
Figure 5: Linear Forward Predictor with a Transversal Adaptive Filter ..................... 20
Figure 6: Energy prediction using AFP and EWMA for a real solar energy profile .... 22
Figure 7: Solar energy generated profile, predicted data using AFP algorithm and absolute error ............................................................................................................. 23
Figure 8: Solar energy harvested profiles with four variations .................................... 33
Figure 9: Normalized remaining energy at the end of execution of each task at its highest speed and their slowdown factors (harvested energy comparable to energy dissipated) ............................................................................................................. 34
Figure 10: Normalized remaining energy for all tasks assumed to be executed at its highest speed and their slowdown factors (Scarce energy harvested) ............... 35
Figure 11: The 3 task sets with their starting and finishing time, scheduled according to the available energy .......................................................................................... 36
Figure 12: Illustration of prediction interval (slots) in AFP vs EWMA for a day .......... 37
Figure 13: Example of wastage of extra “slack” ........................................................... 40
Figure 14: Tasks scheduled using initial and adaptive scheduling when $Ed>0$ for two hyper period ............................................................................................................ 45
Figure 15: Tasks scheduled using initial and adaptive scheduling when $Ed<0$ for two hyper period ............................................................................................................. 48
Figure 16: Variation of deadline miss rate (%) with the maximum capacity of the capacitor and the amount of energy harvested ......................................................... 52
Figure 17: Graph showing the change in % deadline miss rate as $Ed$ and utilization increases .................................................................................................................... 53
Figure 18: Graph showing the increase in % deadline miss rate as $Ed$ decreases and utilization increases ............................................................................................... 54
List of Tables

Table 1: Power dissipation of different tasks at different speed ........................................ 32
Table 2: Attributes for Job 4 and Job 5 with different slowdown factors for a specific
example .......................................................................................................................... 44

List of Algorithms

Algorithm 1: Initial scheduling .......................................................................................... 30
Algorithm 2: Adaptive scheduling when \( Ed >0 \) .......................................................... 43
Algorithm 3: Adaptive scheduling when \( Ed <0 \) ......................................................... 47
## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Time instant</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>Individual tasks indexed as $i$</td>
</tr>
<tr>
<td>$J_i$</td>
<td>Instances of the tasks in the queue after sorting</td>
</tr>
<tr>
<td>$d_i$</td>
<td>Absolute deadline of job $J_i$</td>
</tr>
<tr>
<td>$D_i$</td>
<td>Relative deadline of the job $J_i$</td>
</tr>
<tr>
<td>$w_i$</td>
<td>Worst case execution time of the job $J_i$</td>
</tr>
<tr>
<td>$a_i$</td>
<td>Arrival time of job $J_i$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>Period of job $J_i$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Release time of job $J_i$</td>
</tr>
<tr>
<td>$St_i$</td>
<td>Start time of job $J_i$</td>
</tr>
<tr>
<td>$Ft_i$</td>
<td>Finish time of job $J_i$</td>
</tr>
<tr>
<td>$S_n$</td>
<td>Slowdown factor ranging from 1 to $n$</td>
</tr>
<tr>
<td>$P_D(\tau_i, S_n)$</td>
<td>Power dissipation of task $i$ with a speed $S_n$</td>
</tr>
<tr>
<td>$E_D(St_i, Ft_i)$</td>
<td>Energy dissipation of job $J_i$ from its start time to finish time</td>
</tr>
<tr>
<td>$E_C(Ft_i)$</td>
<td>Energy stored in the capacitor at the finish time of job $J_i$</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
<td>Maximum capacity of the capacitor to store energy</td>
</tr>
<tr>
<td>$E_{\text{thres}}$</td>
<td>Intermediate level of the capacitor</td>
</tr>
<tr>
<td>$Ed(k)$</td>
<td>Difference in energy predicted by AFP and EWMA algorithm for $k^{th}$ slot</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 What is Real-Time Energy Harvesting Systems?

1.1.1 Real-Time Systems

The correctness of real-time systems depends not only on the logical results of computation, but also on the physical instant at which the results are produced [1]. This means that it is not only important for the system to finish the computation of the task but also meet the response time constraints. Some of the examples of real-time systems include helicopter control system, aircraft navigation system and multimedia [2].

A real-time system has a number of tasks running in it. There is certain deadline associated with each real-time system. If the duty is completed after this deadline, the undertaking is considered to be failed. Real-time system can be classified according to its deadline. One of the main classifications is the hard and soft real-time system [3] [4] [5]. A deadline miss in hard real-time system can be catastrophic which means that meeting all the deadlines for such a system is necessary. Some examples of hard real-time system include car engine control system and heart pacemakers. On the other hand, in soft real-time system missing a deadline can result in a significant loss but it is not disastrous. The delay in tasks can be tolerated in this type of systems. This kind of system behavior is seen in multimedia (e.g. live video streaming). Missing few instances of the tasks during video streaming will only degrade the quality of the video but will not destroy the system completely.

1.1.2 Energy Harvesting Systems

Wireless technology enables us to locate sensors in remote areas for monitoring/sensing purposes. This is important as we can’t reach these remote places all
the time to replace the battery after it is discharged. For example, the use of autonomous wireless distributed sensor networks (WDSN) [6] in the event of natural catastrophe and artificial disruptions is very critical. It consists of spatially distributed sensors which help in monitoring physical and environmental conditions like temperature, pressure, etc. These sensor networks have size, power and memory constraints which have to be handled properly in order to obtain an efficient network. The routing of battery powered WDSN is a complicated and demanding task because of limited battery life and less processing efficiency of the sensor [7]. Therefore, we can use renewable energy such as wind, solar, etc as an alternative as it increases the life time of these WDSN in environmental applications. The survey records have proved solar powered WDSN to be functionally better than the battery powered WDSN. Several prototypes like Heliomote [8] and Prometheus [9] have also been designed to prove the dominance of energy harvesting systems.

One of the successful examples for the use of energy harvesting devices in inaccessible areas is NASA’s Mars Exploration Rovers [10] [11]. Mars Exploration Rovers are solar powered devices and as the name hints it was designed to explore the planet mars. It was designed with the objective of characterizing a large variety of rocks and soil in order to get some clue about the past water usage on this planet. The main structure includes an outward protruding ring of ribs which is enveloped by solar panels which harvest the energy and charge the two lithium battery for powering the system.
1.2 Real-Time Workload and Assumptions

1.2.1 Task Model

A real-time system generates some computations which must be completed before the deadline. In many real-time systems these computations are referred to as tasks. In a hard real-time system it is necessary for the tasks to meet the deadlines to save the system from undergoing permanent failure which is little flexible in case of soft real-time systems. The tasks can be classified in three major categories namely periodic, aperiodic and sporadic tasks [12].

Periodic Task Model- Periodic tasks are activated repeatedly at a constant time interval. The interval between two successive activations is called as Period and the activations are often referred to as instances/Jobs. Each instance of the tasks can be defined using its arrival time, period and worst case execution time. The arrival time of instances in periodic tasks is usually known beforehand (sometimes assumed to be at the beginning of the period) and the deadline is assumed to be at the end of the period.

Sporadic Task Model- Sporadic tasks are similar to periodic tasks except it is invoked at irregular time but there is a minimum time interval between each successive job. Such tasks are associated with non-periodic external device interrupts which invokes new jobs at random but each jobs separated by minimum $T$ time intervals between them. Each task can be defined using a worst case execution time and relative deadline which is mainly considered to be its period.

Aperiodic Task Model- Aperiodic tasks are invoked at irregular time interval. There is no specific time interval between the two successive jobs like in case of sporadic tasks. The instances are activated by other applications or external events. We cannot get prior
information of the release time of these jobs. In cases where the interrupts arrive at regular interval, still the tasks will be considered as aperiodic tasks as there is no guarantee of the tasks arriving at regular intervals. These jobs in aperiodic tasks can be defined using its arrival time, deadline and worst case execution time.

### 1.2.2 Real-Time Task Attributes

Each real-time task $\tau_i$ can be characterized using five attributes: Release time, arrival time, worst case execution time, deadline and period [13].

![Real-Time task with its attributes: arrival time, deadline, execution time, release time and period.](image)

**Figure 1: Real-Time task with its attributes: arrival time, deadline, execution time, release time and period.**

**Release Time ($R$)** - This is the time when the task is available for execution. Release time is different from arrival time in a way that a task which has been released is not necessarily ready for execution. Sometimes for periodic tasks, we consider all the jobs are released at the beginning before the execution i.e. time=0, in that case the tasks are said to have no release time.
**Arrival Time** \((a)\) - Arrival time of a task is the time at which the task is ready for execution. The jobs of a specific task can be computed anytime at or after its arrival time depending upon the resource available for execution.

**Absolute Deadline** \((d)\) – The instant of time when the task needs to be completed is called as the absolute deadline. For simplicity, in future we will refer absolute deadline as just the deadline of the task.

**Relative Deadline** \((D)\) - Relative deadline is also the deadline of the task relative to its arrival time. This is the maximum allowable time by which the job should be completed after its arrival time. In other words, relative deadline \((D)\) is the difference between the absolute deadline \((d)\) and the arrival time \((a)\).

**Worst Case Execution Time** \((w)\) - This is the maximum amount of computation time a task needs to get executed in a specific hardware platform. We consider the worst case scenario for the execution time so that there is no additional deadline miss incorporated due to disparity in the execution time.

**Period** \((P)\) - Period of a task is the time frame after which the job repeats itself. Each task is released at a constant rate called its period. This defines the delay between the two consecutive jobs of a task. The period of a task is an essential parameter to differentiate between periodic, aperiodic and sporadic tasks. A hyper period for a set of tasks is the least common multiple of the period of all the tasks.

1.2.3 **Assumptions on task set in our algorithm**

In our algorithm, we have assumed periodic tasks set with different periods. Each task is defined using its period and worst case execution time due to the following assumptions.
• All jobs of a task are assumed to arrive at the beginning of their period and the first instance of a task arrives at time=0. In other words, all instances of a task have same relative arrival time \( (a=0) \).

• All jobs of a task have same release time which is at its arrival time \( (R=a) \).

• All jobs of a task have its relative deadline equal to its period \( (D=P) \).

• All jobs of a task have the same worst case execution time \( w \).

• All tasks have different power dissipation at a specific slowdown factor.

1.3 Real-Time Scheduling

1.3.1 Static Scheduling

Static scheduling [14] is also sometimes referred to as offline scheduling. In this type of scheduling the tasks are assigned to the processor before execution and the priority of tasks does not change during execution. The greatest challenge of offline scheduling is that it is important to predict the behavior of the system beforehand as the scheduling decisions are made during compilation time and cannot be changed during implementation. The scheduling decisions are made based on certain parameters of the task set such as arrival time, deadline and worst case execution time. Offline scheduling incurs less computational overhead as compared to dynamic scheduling.

1.3.2 Dynamic Scheduling

Dynamic scheduling is done during runtime and thus it is also called as online scheduling. The tasks are rearranged during execution based on certain parameters like earliest deadline. Any new instance of the task coming in with the earliest deadline will preempt all the other tasks in the queue and will get the highest priority. Dynamic
scheduling is flexible and adaptive. The overhead due to online scheduling is significant and have to be taken in account. One of the best examples of dynamic priority scheduling is the *Earliest Deadline First (EDF)* [15] [16] [17] scheduling. The priority of all the instances is inversely proportional to its deadline. This means the job of any task with the least deadline has the highest priority and is executed first. The priority is dynamic as it changes with the new arriving jobs with smaller deadline. If at any particular time there are two jobs with the same absolute deadline, the priority is assigned randomly which means the tie is broken arbitrarily.

### 1.4 Power Management in Real-Time System

The amount of energy harvested each day at a particular time varies in a non-deterministic manner. Therefore, energy management schemes are important in energy harvesting embedded systems as it helps in reducing the deadlines miss rate [18] [19]. Some of the control techniques to reduce energy consumption of the integrated circuits especially microprocessor system are Dynamic Voltage Frequency Scaling (DVFS) [20] [21] and Dynamic Power Management (DPM) [22] [23] [24]. Few of the modern microprocessors using DVFS technique are Intel’s Xscale [25] and Transmeta’s Crusoe [26].

#### 1.4.1 Dynamic Power management

This is a technique used to decrease the power consumption of a system. In dynamic power management the power manager (PM) puts the devices which are not being used in sleep mode and some power is saved [27] [28]. These devices become active again when some requests arrive and work in high power state. It is useful to put
the devices in sleep mode only when we know that it can remain in that state for a long
time as this process introduces high energy overhead. This kind of power management
 technique can be time consuming.

1.4.2 Dynamic Voltage/Frequency Scaling

Dynamic Voltage Scaling [29] [30] is a popular and broadly used power
management method in real-time systems. This method can be used to decrease the
power consumption of a system at the time when the sources are limited. The dynamic
power dissipation in a CMOS circuit is strongly dependent upon the clock frequency \( f \)
and the supply voltage \( V \) [31]. The frequency and the voltage are kind of coupled
together. Changing the time of execution of a particular job changes the frequency which
in turn changes the voltage at which the job is executed. The Power dissipation is directly
proportional to the square of the voltage supplied to the circuit as shown in the equation
(1) where \( C_L \) is the gate load capacitance. Another option to decrease power consumption
is lowering the frequency as it is directly proportional to dynamic power. The reason of
not using frequency scaling alone is that it decreases the average power consumption but
the energy still remains the same, thus decreasing the throughput.

\[
P_{\text{dyn}} = C_L V^2 f \quad \text{(1)}
\]

The delay of the circuit \( t_d \) [32] is inversely proportional to \( V \) assuming \( V_T \) to be zero.

\[
t_d = K C_L \frac{V}{(V - V_T)^2} \quad \text{(2)}
\]

The relationship of circuit delay to the supply voltage \( V \) is shown above in equation (2).
Here \( K \) is a constant which is determined by the output gate size and \( V_T \) is the threshold
voltage.
2. This Thesis

2.1 System Model and Assumptions

The real-time energy harvesting systems considered in this paper as shown in Figure 2 can be modeled using three major units: the energy harvesting unit (EHU), the energy storage unit (ESU) and the energy dissipation unit (EDU). The EHU harvests the energy from external sources like wind, sun, etc. This energy helps to power the other hardware/software used in the system. The ESU stores this harvested energy for future use at times when less energy is harvested than required. This helps continuous execution of tasks even at times of deficiency.

Apart from the applications running in the EDU, there are additional softwares running in the CPU: the prediction unit and the scheduler. The energy harvested at a particular time varies in a non-deterministic manner. The prediction unit predicts the future availability of the energy whereas the scheduler schedules the tasks. The first dynamic priority scheduler used in our algorithm is Earliest Deadline First (EDF) which gives highest priority to the instance of the task with imminent deadline [33]. The other scheduler used is voltage/frequency scheduler which alters the microprocessor’s operating voltage at runtime depending on the energy predicted by prediction unit and energy in the storage module. The required energy by the EDU is drawn mainly from the EHU. In cases, where the energy dissipated by the EDU is greater than the energy harvested by EHU we extract the energy stored in ESU and execute the jobs. On the other hand, if the energy harvested is surplus we store the residual energy in ESU.
2.1.1 Energy Harvesting Unit (EHU)

In this paper we talk about a real-time embedded system which is powered by an Energy Harvesting Unit (EHU) [34]. In our case we deal with solar energy which is harvested by solar panels and we denote it by $E_H$. Energy harvested from time $t_1$ to $t_2$ can be calculated using the following formula [35]:

$$E_H(t_1,t_2) = \int_{t_1}^{t_2} P_H(t) dt$$

..... (3)

Here $P_H(t)$ is the power output of the energy source as a function of time. Therefore, it is not possible to determine the exact amount of energy harvested beforehand but we can certainly predict the energy harvested by shadowing the previous energy source profile.

2.1.2 Energy Storage Unit (ESU)
The energy storage unit can be an ultra capacitor [36] or super capacitor [37] which can be used to store the extra amount of energy harvested for the future use at times of crisis. There is always some amount of energy wasted in the process of charging and discharging. So we ignore this loss and assume an ideal battery for our case. There is an upper limit to the storage device denoted by $E_{max}$ which is the maximum capacity of the capacitor. The lower limit of the capacitor is assumed as $E_{thres}$ and not zero which is the energy reserved in the capacitor for worst case scenarios.

2.1.3 Energy Dissipation Unit (EDU)

The variable speed processor used in our algorithm is assumed to be working with $N$ discrete frequencies ranging from $f_i$ (max) $<=$ $f_n$ $<=$ $f_N$ (min) where $f_i$ is the maximum and $f_N$ is the minimum frequency at which the processor can work. The power consumption of the jobs running in the processor is dependent on the processors frequency. Thus the power consumption and voltage level ranges from $P_{D1}$ $<=$ $P_{Dn}$ $<=$ $P_{DN}$ and $V_1$ $<=$ $V_n$ $<=$ $V_N$ respectively for the corresponding frequencies. The power consumption of an instance of a task $n$ depends on its voltage and frequency level which is given by the relationship as shown in equation (1).

We consider a slowdown factor [38] which is ratio of the current frequency to the maximum frequency of the processor and is denoted by $S_n$. This factor can range from $S_{min}$ to 1.

$$S_n = \frac{f_n}{f_{max}}$$ ….. (4)

We assume that each task from a task set have different power dissipation which also changes with its frequencies. Thus a task will have maximum power dissipation at its
maximum frequency and it decreases as the frequency decreases. For convenience, we will define power dissipation of a task as a function of task index and its slowdown factor $P_D(\tau_i, S_n)$. The energy dissipation of the task can be found as shown below:

$$E_D(S_t, F_{t_i}) = P_D(\tau_i, S_n) \times (w_i / S_n) \quad \cdots \quad (5)$$

The instances of the tasks that are ready to get processed enter the ready queue as shown in Fig. 2. Each instance of the task in the queue is denoted by two tuples $(P_1, w_1)$ where $P_1$ is the period of the task and $w_1$ is the worst case execution time of the ready task $\tau_1$. All the instances of the tasks in the queue are sorted in ascending order of their deadline using EDF policy. In order to avoid confusion later in the algorithm, we will denote the periodic instances in the queue sorted by EDF scheduling as jobs. The task instance with highest priority is referred to Job$_1$ ($J_1$) and so on.

For example, if $t_1$ has a deadline at 10 and $t_2$ at 6, then $t_2$ will be executed first. In this case the system is considered to be preemptive. We assume that if all the instances of the tasks are running at its full speed, there will be no deadline misses of the tasks. The jobs can be slowed down by a factor $S_n$ in cases where we have insufficient energy to run the job at its full speed which in turn increases the worst case execution time to $w_n/S_n$.

Apart from the applications running in the processor, prediction algorithm and scheduler is also running inside the CPU. In different energy management algorithms and voltage-scaling techniques it is important to evaluate the exact amount of energy harvested in the near future, which is difficult. Some of the energy prediction schemes can be found in [39] [40]. In our thesis, we have used two different prediction units. The first one being EWMA in which a day is divided into number of slots. The amount of energy predicted for all the slots in a day is calculated using the real value at the end of same slot in the
previous day and the predicted value of the previous slot in the same day. These values are then stored in form of a vector. Suppose, we divide the day into 1440 slots, therefore a vector of 1440 predicted values will be formed at the beginning of the day as shown in Figure 3. Although this algorithm shows an error of 30-50%, it can be used to get an estimation of the energy profile throughout the day. The next prediction unit AFP gives us a more accurate prediction of the energy harvested in larger slots (1 hour or 30 minutes) with error less than 5-10% using an adaptive technique by changing the weights of the filter as described later in Section 3.

![Figure 3: Illustration of prediction interval (slots) in EWMA](image)

2.2 Related research

Many researchers have extensively studied low power systems in accordance with renewable energy using DVFS. Some of the previous energy management schemes mainly focus on reducing the energy consumption by the tasks, hence reducing the CPU power and also meeting the tasks deadline. Dondi et al. [41] proposed a scheduling algorithm which depends on the energy predictor to keep the workload consistent over time. The paper aims in increasing the number of tasks executed each day mainly dealing with periodic tasks and on-demand tasks. Here, the priority of the tasks is not according to its deadline but the type of task being executed. Vishnu et al. proposed a mixed integer
linear programming model in [42] which aims at decreasing the total energy consumption and deadline miss rates. Moreover, here it was assumed that the amount of solar energy harvested at any particular time is unlimited. It is important to consider the energy availability at all times especially at night because there is a possibility of energy shortage at that time. Liu et al. in [43] proposes a load-matching task scheduling algorithm using adaptive technique. A realistic charging and discharging model is used in this paper. The authors of [33] aim at exploiting the time slack as much as possible and also reducing the deadline miss rate. There is a good tradeoff between energy saving and speed depending on the available energy which includes the energy stored in the capacitor and the energy harvested in the future.

2.3 Motivation

In most of the above mentioned papers, the energy predictor is either assumed to be perfect or the harvested energy is assumed to be constant [44]; which is not the case in reality. Another problem that can arise in [33] is that the tasks are executed at full speed when there is sufficient energy available to execute the task within its deadline. This causes a risk of having negligible energy stored in the capacitor for the execution of next job in the queue. This problem is taken care in our algorithm by keeping some reserve energy in the capacitor at all times for the worst case scenarios. Energy management schemes require accurate energy predictors to estimate future energy availability. Therefore, an energy predictor is required so that we can predict the future energy which is essential for energy management schemes. Many researchers have come up with different algorithms to predict the solar energy, few of them being Exponential Weighted
Moving Average (EWMA), Weather-Conditioned Moving Average (WCMA) and Weather-Conditioned Selective Moving Average (WCSMA) [45] [46] [47] [48]. These algorithms gave an error of 30%, 10% and 10% respectively. All of the above mentioned scheduling methods/algorithms are unique in their own way. They all aim at different issues of energy management in embedded systems but there are few drawbacks. These works either assume the energy harvested at all times are constant and unlimited or a perfect energy predictor model for their algorithm, which in reality is not viable. It is very important to deal with the realistic energy predictors which give an error ranging from 10-40% depending on different algorithms proposed and the variation in the real data.

2.4 Objective

In this thesis, we propose an adaptive energy management scheme (AEMS) for energy harvesting embedded systems. We have considered energy predictors for our scheme which are not perfect and try to minimize the deadline miss rate as much as possible under these imperfect conditions. This can be done using adaptive techniques by compensating for the difference in energy predicted by two prediction algorithms. AEMS addresses the problem of using ideal energy predictor in energy management techniques which further increase the overall deadline miss rates for these energy management schemes in real conditions. To resolve this issue, we first use an offline initial scheduling algorithm employing EWMA and then implement an online (adaptive) scheduling which uses Adaptive Forward Prediction (AFP) algorithm as described in Section 3. We use EWMA prediction scheme for initial scheduling as it is a frequently used and low cost algorithm. It calculates the amount of energy likely to be harvested in the entire day.
which helps the initial schedule to get an estimate of energy harvesting profile for each day. Further, by using AFP prediction algorithm which gives us a better idea of the energy predicted the next hour (or 0.5 hour), we try to show that the adaptive algorithm proposed in this paper can further refine the initial scheduling to get desired results. The online adaptive algorithm modifies the initial algorithm in conditions where we might have more/less energy harvested than predicted by EWMA at a particular time. This way we compensate for the difference in energy efficiently thus minimizing errors in real conditions. We were successful in decreasing the deadline miss rate by 15-30% under different workload conditions. The online adaptive algorithm when combined with any other energy management algorithm (as initial scheduling) gives a better result than those management schemes alone.

2.5 Main Contributions

The energy harvested from the environment varies in a non-deterministic manner. It is not possible to know how much energy will be harvested at a particular time of the day. Therefore power management is required in energy harvesting system as the available energy for execution can only be predicted but not confirmed unlike the battery powered systems. Therefore this thesis focuses on:

1. Developing an energy prediction scheme which uses adaptive technique to predict the energy available in the near future (one slot) using energy harvested at previous slots with high accuracy and reliability.
2. Create an offline scheduling algorithm which decides the speed of all the jobs of a task according to the rough estimate of the energy available throughout the day
using EWMA. Offline scheduling gives us a rough idea about the instances of the tasks which might not be able to meet the deadline due to timing or energy constraints.

3. After getting a rough idea about the speeds of all the jobs through offline scheduling, we change the speed of the jobs online in such a way that the deadline miss rate is further decreased. This is done by utilizing the extra energy available and also managing the tasks at the time of shortage. This is done in accordance with the solar energy predicted by AFP algorithm.
3. AFP prediction algorithm

3.1 Theory

It is very important to have a reliable and accurate energy predictor in power management techniques as it increases the efficiency of the optimization techniques of the energy harvesting systems [49] [50]. An adaptive technique was employed to predict the future availability of solar energy as it tracks small changes in the input signal and dynamically adjust to the changes. Adaptive techniques are advantageous because they do not require a priori knowledge of the signal as in case of EWMA which introduces large error percentage. This large error percentage is due to the high impact of the previous day values on the predicted value as the previous day data might vary significantly with the present day data. It estimates the amount of future energy harvested and then calculates the error in the prediction which is the difference between the predicted energy and the real energy. This error is then feedback to the adaptive algorithm block to modify the filter’s parameters/weights. The parameters are changed in such a way so as to minimize the least mean square error of the output signal.

We prefer to predict energy each hour (or 30 minutes) rather than minute as the solar energy does not vary so frequently. The real energy harvested each hour is represented by \( x(n-t) \). Here \( n \) represents the time of the day in hours and \( t \) represents the position of previous inputs from time \( n \) where \( t = \{ 0, 1, 2 \ldots \} \).
Figure 4: Forward Prediction Filter

Figure 4 shows a N-tap linear prediction filter [51] [52] with inputs namely \(x(n - 1), x(n - 2), x(n - 3) \ldots \) and \(x(n - N)\). We estimate the energy harvested at time \(n\) with the help of linear combinations of the past \(N\) input samples. We use \(N\) delay blocks to get a delayed version of the present input. The inputs are then multiplied by a unique weight and it changes with iterations in such a way so that the error between desired signal and predicted signal is minimized. The weights here mainly signify the importance given to each input in their contribution towards the energy prediction. This filter can be well defined with the help of the following equations:

\[
x_{in} = [x(n-1), x(n-2), x(n-3), \ldots x(n-N)] \\
y(n) = \sum_{k=1}^{N} w_k x(n-k)
\]

Here \(x_{in}\) are the previous inputs at time \([n - 1, n - 2, \ldots n - N]\), \(y(n)\) is the predicted value for time \(n\) and \(N\) is order of the filter.
Adaptive filter block use LMS algorithm as shown in Figure 5, to update the weights of the inputs $x_{in}$. It uses the steepest descent to find filter weights $w$ which minimizes the LMS error. The LMS error is defined by:

$$ e(n) = E|e(n)^2| $$

where $e(n)$ is the prediction error for the current iteration ‘n’ and $E|.|$ denotes the expected value.

In our case, we have considered the desired signal to be one delayed sample of the input $x(n)$ which is the energy harvested at $(n-1)^{th}$ minute. This is because the desired signal $x(n)$ at time $n$ is not known. To find the real error at the present time $n$ we have to wait for one delay. A delay block was introduced at the end of the adder to get a delayed version of the predicted value. To update the weights, we need to find out the forward prediction error which is the error between the desired input $d(n-1)$ and the predicted value $y(n-1)$. The error was then calculated using the following equation:

$$ e = d(n-1) - y(n-1) $$
The weights at an instance \((n+1)\) are updated according to the equation:

\[
w(n + 1) = w(n) + \mu (e \times x_{in})
\]

….. (10)

Here \(w(n)\) is the weight of the adaptive filter at an instance \(n\) and \(\mu\) is the step size or the convergence rate which means how fast or slow the weights of the filter will converge so that the predicted energy merges with the real energy harvested at that particular time.

3.2 Simulations

The proposed algorithm was tested using real and generated solar energy profile. The real solar energy profile was obtained from [53] which give us solar energy generated per hour in Hartford Bradley Intl AP, CT. A very commonly used solar energy profiles was also generated using the following equation [54]:

\[
P_{H}(t) = |10N(t)\cos(t/150\pi)\cos(t/120\pi)|
\]

….. (11)

where \(N(t)\) is the random number generator from a normal distribution with mean 0 and variance 1.
Figure 6: Energy prediction using AFP (top) and EWMA (bottom) for a real solar energy profile

The above simulation was done for 400 hours including sunny and cloudy days followed by night. Comparing energy predicted by both the algorithms, it is evident that AFP algorithm traces the input data better than EWMA. AFP scheme gives us an enhanced idea of the energy available at the next hour than that predicted by EWMA for a full day. The prediction error can be decreased further by 25-50% after using AFP scheme depending on the variation in the input signal.

Next, the algorithm was tested using generated solar energy profile. The day was divided into 48 slots which mean each slot was 30 minutes. ‘N’ previous inputs were considered in order to predict the energy available in the next slot. This profile was generated using equation (11) and was tested using different random numbers to check the algorithm for different degree of randomness. Every time the algorithm gave a different error percentage depending on the randomness introduced. The error varied
between 6-15% depending on the input signal. It was successfully able to track the input signal as shown in the graph below:

Figure 7: Solar energy generated profile, predicted data using AFP algorithm and absolute error
4. The AEMS Algorithm

AEMS algorithm dynamically adjusts itself to difference in energy predicted by the two prediction algorithm namely EWMA and AFP. First, we estimate the amount of energy harvested for a full day using EWMA and then apply our static initial scheduling. This initial scheduling will have significant errors as EWMA algorithm shows an error of 30-50%. This high error is because EWMA needs prior knowledge of the input signal and the input is not consistent over time. This algorithm proves good for consistent weather conditions but shows significant errors when sunny and cloudy days are mixed. The initial schedule step is required to have a rough estimate of the start time and execution time of the tasks that will be executed throughout the day. The instances of the tasks in the queue referred to as “Jobs \( J_i \)” are sorted in accordance to its deadline and the initial scheduling is applied on them. After this an adaptive technique is applied for energy management which uses the AFP prediction algorithm to get a better prediction of energy being harvested at the next hour. AFP algorithm is also effective during mixed weather conditions including sunny and cloudy conditions. Using adaptive technique, we compensate for the difference in energy predicted (more or less) in each slot and change the scheduling accordingly just before executing the jobs in that slot. This helps us to utilize the energy in a more effective way and also decrease the deadline miss rate.

4.1 Initial Offline schedule using EWMA

Offline scheduling was used for scheduling all the tasks according to the rough estimate of the energy available throughout the day. Offline scheduling incurs less computation overhead than online scheduling. The main aim of the initial scheduling is to
decide the slowdown factor for all the jobs depending upon the energy availability. This means that for the execution of a job we first consider the amount of energy harvested during its execution time and then decide the amount by which the job can be stretched in order to avoid energy deficiency at any point of time. The tasks considered in our algorithm are periodic tasks which become ready for execution at a given constant rate called its period denoted by ‘\( P \)’. The jobs enter the ready queue as soon as they are released and are sorted according to their deadlines with the earliest deadline first policy. Using the earliest deadline policy, we can guarantee that all the jobs of the tasks will meet its deadline if the schedulability condition is guaranteed. The equation below shows the condition which guarantees the tasks schedulability for a set of \( M \) periodic tasks with worst case execution time \( w_i \) and period \( P_i \):

\[
\sum_{i=1}^{M} \frac{w_i}{P_i} \leq 1 \quad \text{….. (12)}
\]

The above condition can only guarantee the timing constraint of the periodic task set assuming that there is no energy constraint of the system. In reality, energy harvesting systems do have energy constraints and thus energy should be saved using voltage/frequency scaling whenever possible to avoid energy deficiency at any time during execution of the tasks. In offline algorithm, we assume that all instances of the task are getting executed at the highest speed of the processor. As we know each task has different power dissipation at different speed and the maximum dissipation being at the highest speed. Thus assuming that the job is running at highest speed gives us the maximum power requirement for a particular job. This way we can calculate the status of the capacitor at the end of the job execution if it is executed at maximum speed. The
following condition shows the energy in the capacitor at the end of job execution whose start time and finish time is given by $St$ and $Ft$ respectively:

\[ E_c(Ft_i) = E_c(St_i) + E_H(St_i, Ft_i) - E_D(St_i, Ft_i) \]  \quad \ldots \quad (13)

$E_c(St_i)$ is the energy stored in the capacitor at the start time, $E_H(St_i, Ft_i)$ is the energy harvested from the environment between the start time and finish time of the Job $J_i$ and $E_D(St_i, Ft_i)$ is the energy dissipated by the Job during its execution.

The speed of each job is based on the energy left in the capacitor at the end of execution of the job at its highest speed. The capacitor was marked with three main regions. The uppermost level is known as $E_{\text{max}}$. An overflow occurs if the energy exceeds this level and the extra energy is wasted. The lower most level is considered to be zero and an underflow occurs if the level is at zero which means there is no energy left in the capacitor for usage. The third level is the $E_{\text{thres}}$ which is the minimum amount of energy which we want to store in the capacitor at all times while making the decision about the speed of the job so that the worst case scenarios can be avoided. This means that there can be a situation where we use up all the energy to execute the current job at its maximum speed which might hamper the other tasks in the future to meet their deadlines due to energy constraints. So it is good to keep some energy reserved for later use when the energy harvested is scarce. There can be another situation where we slow down the job to the maximum possible extent just before its deadline in order to achieve the maximum energy savings. This kind of slowing down of the jobs can give rise to conditions where we have sufficient energy to execute the present job at the maximum speed, but still there is a timing constraint. Even if the job runs with the slowdown factor equals one which is its worst case execution time, it cannot meet its deadline as the jobs
before it were slowed down to their maximum extent. Therefore, it is very important to maintain some balance between energy saving and energy usage. It is good to save the energy especially in energy harvesting systems but it is also important to utilize the energy fully whenever required like in case of overflow conditions.

We can possibly have four cases using the three limits mentioned above. The first case is when the energy left in the capacitor is between $E_{\text{max}}$ and $E_{\text{thres}}$. We call it as ‘safe’ position as we want the capacitor to be filled all the time. The second case is when the energy lies between $E_{\text{thres}}$ and level zero. This situation is ‘critical’ as serious attention is required to save some energy for future use. The third situation is the ‘underflow’ condition where level of capacitor is less than or equal to zero and last being the ‘overflow’ condition where it is above $E_{\text{max}}$. In each of these different situations we chose the slowdown factor for the jobs accordingly.

In this initial schedule, we try to schedule all the jobs as soon as possible so that we have more time to finish other jobs within their deadline. Let’s assume we have $M$ number of jobs in the ready queue. The start time and finish time for each job in the queue can be represented by $St$ and $Ft$ respectively. The start time of the first job $J_1$ in the queue is equal to its arrival time and is expressed as:

$$St_1 = a_1$$  \hspace{1cm} (14)

The start time of the remaining $(M-1)$ jobs are calculated using:

$$St_i = \max(a_i, Ft_{i-1})$$  \hspace{1cm} (15)

where $i$ ranges from 2 to $(M-1)$.

**Case 1:** $E_c(Ft_i) \geq E_{\text{thres}}$ and $E_c(Ft_1) < E_{\text{max}}$

In this condition, which we call it as the ‘safe’ condition, there is room for some
energy saving by slowing down the job. On the other hand, we do not want to slow down the job completely as it might result in deadline miss for the future jobs due to timing constraint. As our processor works with $N$ frequencies, the slowdown factor will range from $(S_1(S_{\text{max}}), \ldots S_n, \ldots S_N(S_{\text{min}}))$ where $S_1$ and $S_N$ are the slowdown factors of the tasks running at the highest (max) and lowest (min) speed respectively. Thus we find the intermediate speed at which the processor works using the following equation:

$$S_{\text{mean}} = S(\text{ceil}(\frac{\text{min}+\text{max}}{2})) \quad \text{...... (16)}$$

Here ceil rounds off the index to the nearest integer either greater than or equal to the index. We then check if it is possible to meet the deadline of the job starting at a time as calculated using equation (14) or (15) by using the above mentioned $S_{\text{mean}}$. If the deadline is not met using $S_{\text{mean}}$ we increase the slowdown factor to the next higher index. This can be explained using the condition below:

$$St_i + \frac{w_i}{S_n} \leq d_i \quad \text{...... (17)}$$

The $S_n$ in equation (17) ranges from $S_{\text{mean}}$ to $S_1 (S_{\text{max}})$. This way we get the maximum energy saving as possible and also meet the deadline. In worst case scenario, the job even when executed with the highest speed $S_1$ wouldn’t be able to meet the deadline due to timing constraint imposed by the previous jobs which was slowed down to a great extent.

**Case 2:** $E_c(F_{t_j}) \geq 0$ and $E_c(F_{t_j}) < E_{\text{thres}}$

The above equation signifies a state where the capacitor is in ‘critical’ condition and energy saving becomes essential. Here we change the slowdown factor to the minimum keeping the deadline in consideration. We consider $S_{\text{min}}$ and check if the job
meets its deadline using the equation defined in (17). If the equation doesn’t stand true, we increase the slowdown factor to the next higher index. We do this till equation (17) is satisfied. There can be a situation where even at $S_{\text{max}}$ the job doesn’t meet its deadline. This will be due to the timing constrained explained in Case 1.

**Case 3:** $E_c(F_t_i) < 0$

This is the underflow condition which shows that the job cannot be executed at its highest speed. Here, we want to find out the speed at which $E_c(F_t_i)$ is greater than 0 for Job $J_i$. As we know, the jobs with highest slowdown factor consume maximum power and the power consumption decreases with speed. Thus, we check the energy dissipation of the job with different speed ranging from $S_{\text{max}}-1$ to $S_{\text{min}}$. Suppose the speed at which $E_c(F_t_i)>0$ is $S_k$. We then check if the deadline of the job is met using $S_k$. If it is true, we consider a loop from $S_k$ to $S_{\text{min}}$ and chose the minimum slowdown factor at which both energy requirement and timing requirements are met. If at $S_k$ the timing requirement is not met, we still execute the job at a speed $S_k$ so that we can change the speed of this job during adaptive scheduling when we have more energy as predicted by AFP algorithm. One expected outcome can be that the $E_c(F_t_i)$ does not become greater than zero even at the minimum slowdown factor. In that case, we will assign the job the lowest slowdown factor $S_{\text{min}}$ so that we can adaptively change the speed during online scheduling in case if energy difference is positive that is more energy predicted by AFP than EWMA.

**Case 4:** $E_c(F_t_i) >= E_{\text{max}}$

This is the overflow situation which means that the energy left in the capacitor at the end of the execution of the job at its highest speed is greater than or equal to $E_{\text{max}}$. Any energy saved due to voltage scaling will be wasted as there is no space in the capacitor to store it.
Thus we assign the maximum speed to the current job to avoid wasting any energy.

**Algorithm 1: Initial scheduling**

**Input:** M periodic task set with its period and worst case execution time  
**Require:** A processor working with N discrete frequencies ranging from $f_1$ being the maximum (max) and $f_N$ minimum (min).

I. Sort the tasks in the queue according to its deadline with the earliest deadline first.

II. Determine the start time of the tasks

1. for $i=1:M$
2. if $i==1$ then
3. $St_1=a_1$
4. else
5. $St_1=\max(a_i, Ft_i)$
6. end if
7. end for

III. Determine the speed at which the task will be executed considering energy availability

1. Assume $\tau_i$ is executed with $w_i$ ($S_{max}=1$)
2. Calculate remaining energy in the capacitor at the end of execution using equation $E_c(Ft_i) = E_c(St_i) + E_H(St_i, Ft_i) - E_D(St_i, Ft_i)$

3. If $E_c(Ft_i) \geq E_{thres}$ & $E_c(Ft_i) < E_{max}$
4. Calculate $S_{mean} = S(ceil(\frac{\min + \max}{2}))$
5. for $S_n=S_{mean} : S_{max}$
6. if $St_i + \frac{w_i}{S_n} <= d_i$
7. $S=S_n$
8. break;
9. end if
10. $S=S_{max}$
11. end for

12. elseif $E_c(Ft_i) \geq 0$ & $E_c(Ft_i) < E_{thres}$
13. for $S_n = S_{\text{min}}$: $S_{\text{max}}$
14. \hspace{1em} if $S_t + \frac{w_i}{S_n} \leq d_i$
15. \hspace{2em} $S = S_n$
16. \hspace{2em} break;
17. \hspace{1em} end if
18. \hspace{1em} $S = S_{\text{max}}$
19. \hspace{1em} end for

20. elseif $E_c(F_{t_i}) < 0$
21. \hspace{1em} for $S_n = S_{\text{max}}$: $S_{\text{min}}$
22. \hspace{2em} $E_{\text{new}}(St_i, F_{t_i}) = P_d(\tau_i, S_n) \frac{w_i}{S_n}$
23. \hspace{2em} If $E_c(St_i) + E_h(St_i, F_{t_i}) - E_{\text{new}}(St_i, F_{t_i}) > 0$
24. \hspace{3em} If $S_t + \frac{w_i}{S_n} > d_i$
25. \hspace{4em} $S = S_n$
26. \hspace{4em} break
27. \hspace{4em} else
28. \hspace{5em} for $S_t = S_{\text{min}}$: $S_n$
29. \hspace{6em} if $S_t + \frac{w_i}{S_n} \leq d_i$
30. \hspace{7em} $S = S_t$
31. \hspace{7em} break
32. \hspace{7em} end if
33. \hspace{7em} end for
34. \hspace{4em} end if
35. \hspace{4em} end if
36. \hspace{4em} $S = S_{\text{min}}$
37. \hspace{4em} end for

38. elseif $E_c(F_{t_i}) \geq E_{\text{max}}$
39. \hspace{1em} $S = S_{\text{max}}$
40. end if

IV. Determine the finish time of the tasks

41. $F_{t_i} = St_i + (w_i/S)$
4.1.1 Motivational Example

A simple example will be used to demonstrate the algorithm explained above. We consider 3 periodic tasks with its execution time and period as: (1, 5), (2, 10) and (3, 20) and a utilization factor of 0.55. We know that its arrival time is the beginning of its period and deadline the end of the period. All the instances of the tasks enter the ready queue according to its deadline. The processor is assumed to be working with 6 discrete slowdown factors which are 1, 0.7, 0.5, 0.3, 0.2 and 0.1. The power consumption of different tasks at different speed is shown below:

<table>
<thead>
<tr>
<th>Power Dissipation</th>
<th>S=1</th>
<th>S=0.7</th>
<th>S=0.5</th>
<th>S=0.3</th>
<th>S=0.2</th>
<th>S=0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1(τ₁)</td>
<td>300</td>
<td>200</td>
<td>90</td>
<td>50</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Task 2(τ₂)</td>
<td>400</td>
<td>250</td>
<td>100</td>
<td>50</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>Task 3(τ₃)</td>
<td>650</td>
<td>400</td>
<td>200</td>
<td>90</td>
<td>50</td>
<td>40</td>
</tr>
</tbody>
</table>

**Table 1: Power dissipation of different tasks at different speed**

The corresponding voltages for the different slowdown factors are 10, 7, 5, 3, 2 and 1. The maximum capacity of the capacitor $E_{\text{max}}$ is assumed to be 3000 and $E_{\text{thres}}$ as 200. The capacitor is assumed to be full at the beginning of the execution of the tasks.

Four different energy harvesting profiles used were numbered from 0 to 3. The profile numbered 0 is where least amount of energy was predicted throughout the day and 3 being the highest. All the simulations were done using these four different energy profiles predicted by EWMA which is shown in Figure 8.
Figure 8: Solar energy harvested profiles with four variations

Figure 9 and 10 shows the amount of energy left in the capacitor at the end of execution of all the jobs at its highest speed and the slowdown factor chosen for each jobs. The energy in the capacitor at the end of execution was normalized with respect to the maximum capacity of the capacitor \( \frac{E_c (St_i) + E_H (St_i, Ft_i) - E_D (St_i, Ft_i)}{E_{\text{max}}} \). When the red line goes above 1, it means that the capacitor is overflowing.

Two different cases are shown: First, where energy harvested is sufficient enough to execute all the jobs before its deadline using energy harvested profile \( (E_H=3) \). In this case, we observe that even at the highest speed of execution the amount of energy left in the capacitor is either very close to one or greater than one. For the jobs where \( E_c (Ft_i) \) exceeded one we executed it in full speed and if it was less than one we prefer to increase the execution time of the task to a point where it just meets its deadline.
Figure 9: Normalized remaining energy $E_c(Ft_i)$ at the end of execution of each task at its highest speed and their slowdown factors (harvested energy comparable to energy dissipated)

The second case demonstrates the job getting executed at an environment where we have scarce energy as shown in energy harvested profile ($E_H=1$). The harvested energy was decreased evenly by 50% and the same set of tasks was executed and the behavior of the algorithm was tested. In this case, 30% of the jobs missed its deadline due to energy deficiency. The jobs which missed the deadline were also considered in offline scheduling with their finish time greater than the deadline. This is because initial scheduling is just to get an idea about the energy profile and how much each job can be slowed down so that any deadline miss can be avoided. The scheduling will change during execution of the task adaptively in order to decrease the deadline miss.
Figure 10: Normalized remaining energy $E_c(F_t)$ for all tasks assumed to be executed at its highest speed and its slowdown factors (Scarce energy harvested)

Figure 11 shown below demonstrates a set of 3 tasks with periods of 5, 10 and 20. The hyper period for these tasks is 20 which is the least common multiple of the three periods. All the jobs are displayed using its start time and finish time, arranged according to the earliest deadline first and slowed down to the extent as shown in Figure 10. Three different colors of the jobs signify that there are three different tasks. The jobs which missed its deadlines are shown with a dotted red arrow. The jobs of all tasks were shown in a voltage verses time diagram where each job was slowed down and shown with their respective voltage level.
Adaptive scheduling using AFP

The task set scheduled in offline scheduling algorithm was based on the energy predicted by EWMA algorithm with a high error percentage. This prediction algorithm was useful to get a rough estimate of the energy harvested throughout the day so that a rough scheduling can be done offline. This kind of initial scheduling is important as it allows us to calculate the job’s start time, finish time and slowdown factor offline. This helps us to overcome the problem of energy shortage for a particular job due to the excessive energy usage on the previous job. In paper [33], the task is scheduled to run at highest speed if it has sufficient energy irrespective of the energy availability in the future which can give rise to the problem mentioned above. Therefore, an adaptive scheduling is required after the initial scheduling where we can recalculate the start time, finish time and slowdown factor of all the tasks in order to avoid any deadline miss. The scheduling
is changed before the execution of the jobs in each slot. Energy predicted by EWMA is done for smaller slots whereas AFP predicts energy harvested in larger slots. The energy harvested throughout the day can be estimated by EWMA as the calculation is based on the energy predicted in the previous slot and real energy harvested in the same slot of the previous day. Therefore, we don’t need to wait for real energy harvested in the previous slot as in case of AFP prediction. Thus the adaptive scheduling is done online as the energy available for only one slot can be known at a time. The basis for the change in scheduling is the error in the energy prediction by two algorithms: EWMA and AFP. Figure 12 demonstrates different size of prediction interval (slots) in both the prediction algorithms.

**Figure 12: Illustration of prediction interval (slots) in AFP vs. EWMA for a day**

Initial scheduling is done for the entire day as we know the periodic tasks arrival time and deadline along with the energy prediction. On the other hand, adaptive scheduling is done only for one slot at a time as at the end of each slot we know the energy being harvested in the following slot. Thus we calculate the difference in the energy prediction by AFP and EWMA $Ed(k)$ for each slot.

\[
Ed(k) = E_{H,AFP}(k) - \sum_{i=30k-29}^{30k} E_{H,EWMA}(i) \quad \ldots (18)
\]
Here $E_{H,AFP}(k)$ denotes the energy predicted by AFP algorithm for the $k^{th}$ slot. $E_{H,EWMA}(i)$ represents the energy predicted by EWMA algorithm for the same slot where $i$ ranges from $(30k-29)$ to $30k$.

There are two possible situations where the initial algorithm needs to be modified.

1. When $Ed(k)$ is positive, this signifies that the energy harvested during the $k^{th}$ slot is more than predicted by EWMA. In this kind of scenario, we need to make use of this extra energy in order to meet the deadline of the job which missed its deadline in the initial scheduling.

2. On the contrary, if $Ed(k)$ is negative it indicates that the energy predicted by AFP (with lesser prediction error than EWMA) is less than the amount of energy forecasted by EWMA for the $k^{th}$ slot. In this state, some jobs have to be removed from the queue without execution in order to avoid energy and time deficiency for future tasks. The decision of which job should be removed without execution will be made based on the least amount of jobs in that particular slot missing its deadline.

4.2.1 Extra Energy Available ($Ed$ is positive)

It was observed in Figure 11 that in conditions of scarce energy harvested, 4 jobs missed its deadline considering two hyper periods ($=40$) of all the tasks. The deadline miss of these jobs was either because of the energy constraint or timing constraint imposed on them due to previous jobs. Therefore, we need to adjust the initial algorithm
in such a way that we minimize the deadline miss rate of all the jobs by utilizing the energy efficiently for a case where energy predicted by AFP algorithm is more.

It is known from Section 1.4.2 that the power consumption of a particular task increases as the frequency/voltage increases. We can use the extra energy to run some job in the slot at a higher speed (higher frequency) in order to achieve extra “slack” which can be transferred to the future jobs in order to decrease the deadline miss rate of the jobs running in the system. Ideally, the job with minimum power dissipation and lowest slowdown factor should be chosen for execution at a better speed in order to achieve system wide efficiency.

There can be a possibility where the above solution fails if we chose to execute the job which was slowed down the most in the slot. The problem in using the above solution to decrease the deadline miss rate is that each job has a different arrival time and the job cannot be executed before its arrival time. Therefore, there can be a situation where this additional slack which is earned by the extra energy harvested is of no use to the jobs which missed its deadline in the initial algorithm. This situation is shown with the help of two periodic tasks with periods of 5 and 10 represented by yellow and blue color respectively in the figure shown below. The arrival time is at the beginning of the period which is 0, 5 and 10 for task 1 and 0, 10 and 20 for task 2.
It is clearly visible that given enough energy to execute any job in the queue at its highest speed, the maximum slack will be achievable if we execute the Job 3 (second instance of task1) at highest speed. This slack is of no use to the job missing its deadline (Job 5- third instance of task1) as the slack cannot be transferred further. The reason being the Job 4 which arrives at time=10. Therefore, the task’s starting time can be no sooner than time=10. As a result, the slack produced is wasted and thus wasting the extra energy predicted by AFP algorithm.

We can overcome the problem of wastage of slack by considering all the jobs’ arrival time which lies in between the job getting executed at a better speed and the job which missed its deadline. Including the arrival time of these jobs in the algorithm ensures that the energy difference is efficiently utilized in order to get the maximum useful slack.
Our next aim is to present an algorithm which achieves the best results in utilizing the extra energy available to us. The above mentioned problem should be considered during the decision of allocation of extra energy to any job. The following steps help us to locate the jobs in a particular slot whose speed has to be increased with respect to the energy available in order to maximize the tasks meeting its deadline.

1. Check for the first job which missed its deadline in the present slot after the initial scheduling. Let us assume it to be $J_k$.

2. Check any job with a higher priority than $J_k$ which lies in the current slot whose starting time is greater than or equal to arrival time of $J_k$ or finish time is greater than arrival time of $J_k$. This condition does not guarantee that the slack gained due to extra energy predicted can be transferred to the job $J_k$, but this is surely a necessary condition in order to transfer the slack to the needful task.

3. Each higher priority jobs which satisfies condition 2 is chosen and we increase the speed of the job to one index (higher speed) each time. Let us assume job $J_l$ running with a speed of $S_n$ is the first higher priority job whose start or finish time lies after the arrival time of $J_k$. The new energy dissipation is calculated using equation (19) and the speed is increased by one level until inequality (20) holds true. As soon as $Ed$ decreases below the difference of energy dissipation of the job at two different speeds, the inequality breaks. Thus job $J_l$ is executed with the highest speed possible until inequality (20) holds.

$$E_{D_{new}}(St_l, Ft_l) = P_D(\tau_{m}, S_{n-1}) \times (\frac{w_l}{S_{n-1}})$$ \hspace{1cm} \text{..... (19)}
\[ Ed > E_{D, new} - E_D (St_i, Ft_i) \]  \hspace{1cm} \text{..... (20)}

\( St \) and \( Ft \) are the start and finish time of \( J_i \) respectively. \( P_D (\tau_m, S_{n-1}) \) is the power dissipation of job \( J_i \) which belongs to task \( \tau_m \) at a speed \( S_{n-1} \). \( (w_i / S_{n-1}) \) is the amount of time \( J_i \) takes to execute at a speed \( S_{n-1} \).

4. The maximum possible speed, for each higher priority job, with the additional available energy is calculated. Each of these jobs is considered to have the same starting time as in the initial scheduling and the finish time is calculated using the new speed.

\[ Ft_i = St_i + w_i / S_{new} \]  \hspace{1cm} \text{..... (21)}

The other jobs which come after \( J_i \) until \( J_k \) will be executed with the same speed as decided by initial scheduling. The start time allocated to these intermediate jobs will be the maximum of either the finish time of previous job or its arrival time. Thus a new finish time is computed for \( J_k \) corresponding to each of the higher priority job which is removed.

5. The difference in finish time of \( J_k \) is calculated, which is the difference between the finish time of \( J_k \) in initial scheduling and new finish time calculated above. The maximum difference in finish time is evaluated and the job which contributes to the maximum slack is selected to be executed with highest speed.
Algorithm 2: Adaptive scheduling when $Ed > 0$

Input: M periodic task set with their corresponding start and finish time as scheduled by initial scheduling algorithm.

Require: The difference in energy predicted by AFP and EWMA for a particular slot.

I. Select the appropriate task which gives maximum usable slack and calculate its slowdown factor
   1. Consider a loop for $i=1$: $t$ where $t$=jobs of the tasks which lies in the present slot.
   2. If $\tau_i$ missed its deadline, take another loop for $k=1:i$
   3. Check if $St_k$≥$a_i$ or $Ft_k$>$a_i$
   4. Consider a loop for speed of each $k^{th}$ task ranging from its current speed to highest speed
   5. Calculate energy dissipation due to this new speed
   6. Increase the speed until the following inequality holds true
      \[ Ed > E_D(S_{new}) - E_D(S_{current}) \]
   7. Calculate the new start time and finish time due to the new speed of the task
   8. Schedule the lower priority tasks ranging from $s=(j+1)$ until $i$ using the same seed as selected in initial scheduling:
      \[ St_s = \max(a,Ft_{s-1}) \]
      \[ Ft_s = St_s + w_s \]
   9. The new finish time of $i^{th}$ task is considered and slack is calculated due to all $j^{th}$ task
   10. The task which contributes maximum slack is chosen to be the best choice in order to utilize the extra energy efficiently
   11. End if
   12. End for
   13. End for

II. Reschedule the lower priority tasks from the chosen task till $i^{th}$ task, the same way as described in step 8 above.

4.2.1.1 Motivational example

The similar scenario as in Section 4.1.1 is considered here. In scarce energy conditions, we found out that 4 tasks missed its deadline. This scheduling was adjusted
during adaptive scheduling according to the extra energy available. Let’s assume that the extra energy harvested for the current slot is 557.38. The 6th job in the queue missed its deadline in initial scheduling as shown with red dotted arrows. Job 4 and 5 lies after the arrival time of job 6. The attributes for job 4 and 5 at different speeds are shown in the table below.

<table>
<thead>
<tr>
<th>Job 4(3rd instance of task 1)</th>
<th>Slowdown factor(S)</th>
<th>Power dissipation(PD)</th>
<th>Execution time(w)</th>
<th>Energy dissipation(ED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4</td>
<td>0.3</td>
<td>50</td>
<td>3.33</td>
<td>166.5</td>
</tr>
<tr>
<td>S3</td>
<td>0.5</td>
<td>90</td>
<td>2</td>
<td>180</td>
</tr>
<tr>
<td>S2</td>
<td>0.7</td>
<td>200</td>
<td>1.428</td>
<td>285.6</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>300</td>
<td>1</td>
<td>300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Job 5(1st instance of task 3)</th>
<th>Slowdown factor(S)</th>
<th>Power dissipation(PD)</th>
<th>Execution time(w)</th>
<th>Energy dissipation(ED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>0.5</td>
<td>200</td>
<td>6</td>
<td>1200</td>
</tr>
<tr>
<td>S2</td>
<td>0.7</td>
<td>400</td>
<td>4.28</td>
<td>1712</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>650</td>
<td>3</td>
<td>1950</td>
</tr>
</tbody>
</table>

Table 2: Attributes for Job 4 and Job 5 with different slowdown factors for a specific example

Given the extra energy harvested, Job 4 can be executed with S1 as the difference in energy dissipation of the job at earlier speed S4 and new speed S1 was found to be 133.5 which is less than Ed. The slack earned due to this new speed was 2.33. On the other hand, Job 5 can only be executed at S2 (earlier speed being S3) as the difference in energy dissipation was found to be 242.2 which is less than Ed.
dissipation exceeds $Ed$ if executed with $S_1$. The slack earned in this case was found to be 1.72 which is less than the one due to Job 4. In both the cases, it was found that all the slack produced can be transferred to the task which missed its deadline. Therefore, Job 4 was chosen to get executed at a better speed. The diagram below shows the results of the adaptive scheduling for the current slot (assuming each slot to be of 40min). The upper part of the diagram shows initial scheduling and lower part displaying the result of adaptive scheduling applied on initial scheduling.

Figure 14: Tasks scheduled using initial (top) and adaptive (bottom) scheduling when $Ed >0$ for two hyper period
4.2.2 Less Energy Available \((Ed\) is negative\)

There can be a situation where AFP algorithm predicts less energy available than what we initially forecasted. In this scenario, we need to adjust the initial scheduling in such a way that maximum task can be executed with the given amount of energy harvested. In this case, voltage/frequency scaling is not of great use as we already slowed down the jobs according to the energy available in initial scheduling keeping the timing constraint in mind. Slowing down jobs in order to save some energy for future use is not a good idea as this might miss the deadlines of more jobs.

As there are some jobs which might have missed deadlines in the initial scheduling due to energy constraint it is important to adjust the scheduling in such a way that less number of jobs misses its deadline. If we don’t remove the job before execution, we waste some energy in executing the job which didn’t meet its deadline thus wasting some valuable energy. On the other hand, if we remove the job which gives us maximum profit we overcome the energy constraint efficiently.

Therefore, a job should be chosen which helps to increase the number of instances of the tasks meeting its deadline. In addition, we should also consider the period of task while choosing the job to be removed. This means that a task with a smaller period will have high frequency of occurrence. Thus removing this instance of the task will be more beneficial than tasks which have a large period as this task might occur very rarely. For example, let’s assume we have a wireless sensor placed in the middle of some deserted area for weather monitoring to record the wind speed, wind direction, temperature, pressure and rainfall. Suppose the periods of all the tasks are 2, 4, 10, 20 and 25 minutes.
respectively. It is better to remove the task with period=2 as it occurs every 2 minutes and if we miss the job for the present period we can record another data for wind speed shortly. On the other hand, if we miss the deadline of rainfall which occurs after every 25 minutes, it can be more hazardous for the system. Algorithm 3 outlines the solution for the conditions where Ed<0 (Energy is less than predicted by EWMA).

**Algorithm 3: Adaptive scheduling when Ed<0**

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>M periodic task set with their corresponding start and finish time as scheduled by initial scheduling algorithm.</td>
</tr>
<tr>
<td>Number of jobs which missed the deadline in initial scheduling (ct1)</td>
</tr>
</tbody>
</table>

Require: The difference in energy predicted (Ed) by AFP and EWMA for a particular slot.

I. Execute step II while ct1>1 or Ed>0
II. Find the job which when removed from queue minimizes the deadline miss rate

1. Consider all jobs in the current slot in a for loop
2. Consider another for loop from (i+1) to the last job in the slot
3. Each time remove the i\(^{th}\) job and schedule the other jobs in the queue
4. If j=(i+1), we schedule the job with start time=max(arrival time, start time of i\(^{th}\) task)
5. For other jobs following (i+1) we decide start time using max(arrival time, finish time of previous job)
6. End for
7. End for
8. Find the total number of jobs missing the deadline in the present slot
9. The job or jobs which give minimum deadline miss rate is chosen for further operations
10. An array of jobs which provide minimum deadline miss is formed and the job with highest frequency of occurrence is chosen to be removed from the queue to increase system reliability.
11. Count total number of jobs still missing the deadline
12. End while
4.2.2.1 Motivational example

Figure 15 shows all the jobs in two hyper period scheduled using initial scheduling having the same attributes as used in Section 4.1.1. This is a situation where scarce energy is predicted by AFP and thus the initial scheduling needs to be adjusted. In the example shown below 4 jobs missed its deadline. While calculating, it was found that jobs 4, 5, 13 or 14 when removed from the queue minimize the deadline miss rate. The period of jobs 4 and 13 is 5 which is smaller than the period of jobs 5 and 14 which equals to 20. Thus, removing job with a period of 20 is not a good idea as its frequency of occurrence is very small. So Job 4 was chosen to be removed from the queue. The algorithm continues until the number of deadline miss is greater than one.

![Figure 15: Tasks scheduled using initial (top) and adaptive (bottom) scheduling when Ed <0 for two hyper period](image)
4.2.3 Assumptions on the size of the capacitor

The initial scheduling was done keeping the available energy in mind which is the sum of energy harvested during the time of execution and the remaining energy in the capacitor at the end of execution of previous job. EWMA predicts energy for smaller slots like per minute which made it easier for us to schedule all tasks but with a larger error. On the other hand, AFP predicts energy with a much smaller error percentage but for a larger slots say 30 minutes. Considering, a big slot as in case of AFP, we know that there might be many jobs running in that slot.

In AEMS scheduling, we find the difference in prediction algorithms and compensate for this difference in the initial scheduling. The capacitor was not taken into consideration in this algorithm. We didn’t consider the fact that the AFP slots are larger and the difference in energy might not be available at the particular place in the slot where we are utilizing it. The energy might be harvested any minute or minutes in the present AFP slot. In order to avoid such kind of situations, the capacitor should be large enough to store that extra amount of energy harvested as in case for $Ed>0$.

One way to overcome this problem is to consider a smaller storage capacity in initial scheduling. As our initial scheduling aims in slowing down the tasks according to the energy available, we also take care that the energy is not completely consumed by the present job as this might result in energy deficiency for future jobs. The normalized remaining energy throughout the algorithm was found to be close to one at the end of maximum jobs as shown in Figure 9. Thus we schedule the tasks in initial scheduling
assuming the capacitor to be smaller than in AEMS algorithm so that we reserve some additional place in the capacitor for the extra energy predicted by AFP.

If the above mentioned solution is not applied and the tasks are scheduled without considering the exact position at which the energy was harvested, we might encounter additional deadline miss rate. Some of the task will be able to meet its deadline by utilizing the energy stored in the capacitor but few tasks might encounter energy deficiency in reality, thus adding more deadline miss of the tasks for the given system and lowering the system reliability.
5. Results and Discussions

Solar energy was chosen as the energy harvesting unit in our simulation. The irradiation profile was obtained from [21]. EWMA prediction algorithm was used to predict the energy harvested in smaller slots. With this information, tasks were scheduled using the initial scheduling algorithm with four different environment of the energy harvesting unit as shown in Figure 8.

The deadline miss rate of the initial scheduling depends on the maximum energy a capacitor can store and the energy harvested profile. The following graph in Figure 16 shows how the deadline of the tasks varies with $E_{max}$ and $E_H$. Here, $E_{max}$ is normalized and the simulation is done for $E_{max} = 0$ to $E_{max} = 5000$ at an interval of 400. We consider 4 different energy profiles from 0 to 3 as shown in Figure 8, 0 being the least energy harvested throughout the day (cloudy day) and 3 being the highest (sunny day). It is clearly visible that the deadline miss rate decreases with the increase in amount of energy a capacitor can store. It is also seen that deadline miss rate is inversely proportional to the amount of energy harvested throughout the day.

In few cases like $E_H = 2$, we observe that the deadline miss rate does not go down to zero with the increase in $E_{max}$. This is a situation of timing constraint in which the task even when executed with highest speed cannot meet its deadline. This is because the tasks of higher priority than the present task where slowed down to a great extent which prohibits the present task in meeting its deadline.
After implementing AEMS algorithm on the given task set, it was found that the algorithm was successfully able to compensate any extra amount of energy harvested as predicted by AFP. This extra energy was utilized by the job which gives maximum profit, thus minimizing the overall deadline miss rate of the system. The graph in Figure 17 shows the change in % deadline miss rate as Ed and utilization increases. The processor utilization can be defined as:

$$U = \sum_{i=1}^{M} \frac{W_i}{P_i} \quad \ldots (22)$$

After a certain specific utilization and extra energy predicted, the algorithm was successfully able to decreases the % deadline miss rate to zero. The graph clearly represents an increase in deadline miss rate as the utilization increases and a decrease as the extra amount of energy predicted increases.
Figure 17: Graph showing the change in % deadline miss rate as Ed and utilization increases (assuming Ed>0 for present slot)

The next graph Figure 18 illustrates the increase in deadline miss rate as the amount of energy predicted by AFP decreases and utilization increases. Here, the deadline miss rate includes any job which has been removed from the queue due to the insufficient energy available. The process of removing the jobs from the queue continues till there is more than one job missing its deadline as assigned by the initial scheduling or the total energy dissipation by the removed tasks is less than the amount of energy deficiency predicted by AFP. The deadline miss rate increases with the decrease in energy predicted by AFP. As the processor’s utilization increases, the overall deadline miss also increases.
Figure 18: Graph showing the increase in % deadline miss rate as Ed decreases and utilization increases (assuming Ed<0 for present slot)
6. Conclusions and Future Work

This thesis proposed an adaptive scheduling algorithm for real-time energy harvesting embedded systems. This algorithm considers both energy and timing constraint of the energy harvesting systems unlike most of the scheduling algorithms. An AFP prediction algorithm was also proposed for a better energy prediction for each slot. Based on this information the initial scheduling, which was designed using the information given by EWMA prediction algorithm, was rescheduled. The purpose was successfully achieved by compensating the extra/less energy harvested from the environment in such a way so that system wide efficiency can be achieved. Using adaptive scheduling we were successfully able to decrease the deadline miss rate of the tasks up to 15-30% in addition to the results accomplished by initial scheduling depending on the amount of energy harvested.

There are number of places where we can work in order to enhance the above mentioned algorithm further. The prediction algorithm can be tested with inputs of larger variations. The capacitor constraint explained in Section 4.2.3 should be accounted for in the algorithm to get better results. A realistic capacitor model should be considered and energy loss due to charging and discharging process should be taken in account. It will be interesting to implement the algorithm in hardware platform and obtain some experimental results. It is also important to compute the other parameters of the system such as computational complexity of the algorithm which minimizes the cost of hardware and enhance the performance of the AEMS algorithm.
REFERENCES


[26] “Cruso SE Processor TM5800 Data Book v2.1,”


APPENDICES

Appendix A: Code for initial scheduling using MATLAB

Emax=3000;
Ethres=200;
fileName='Book1.xls'; % reads the energy predicted by EWMA
y=0.7.*xlsread(fileName);
S=[1 0.7 0.5 0.3 0.2 0.1];
PD=[300 200 90 50 10 5
400 250 100 50 25 10
650 400 200 90 50 40];
a=[0 0 5 10 0 10 15 20 20 25 30 20 30 35];
d=[5 10 10 15 20 20 25 30 30 35 40 40 40 40];
w1=[1 2 1 1 3 2 1 1 3 2 1];
t=14;
Ec=Emax;
Ec1=0;
ED1=zeros(1,t);
ED=zeros(1,t);
w=zeros(1,t);
starts=zeros(1,t);
finish=zeros(1,t);
add1=zeros(1,t);
add=zeros(1,t);
Ec2=zeros(1,t);
D=zeros(1,t);
tc=zeros(1,t);
time=0;
count=0;

for i = 1:t
    waste=0;
    if w1(i)==1
        temp=1;
    elseif w1(i)==2
        temp=2;
    else
        temp=3;
    end
    % with max execution speed
    starts(i) = max(time,a(i));
    for j=ceil(starts(i)):1:floor(starts(i)+w1(i)-1)
        add1(i)=add1(i)+y(j+1);
    end
    ED1(i)=PD(temp,1)*w1(i);
    % slowing down tasks
if \( E_{c+\text{add}1(i)}-E_{D1(i)} \geq \text{Ethres} \) && \( E_{c+\text{add}1(i)}-E_{D1(i)} < \text{Emax} \)

\[ S_{\text{mean}} = S(\text{ceil}((1+6)/2)); \]

for \( z = \text{ceil}((1+6)/2):-1:1 \)

if \( \text{starts}(i)+w_{1(i)}/S(z) < d(i) \)

\[ w(i) = w_{1(i)}/S(z); \]

\( t_c(i) = z; \)

break;
end

\[ w(i) = w_{1(i)}/S(1); \]

\( t_c(i) = 1; \)
end

elseif \( E_{c+\text{add}1(i)}-E_{D1(i)} \geq \text{Emax} \)

\[ w(i) = w_{1(i)}; \]

\( t_c(i) = 1; \)
end

elseif \( E_{c+\text{add}1(i)}-E_{D1(i)} < \text{Ethres} \) && \( E_{c+\text{add}1(i)}-E_{D1(i)} \geq 0 \)

for \( z = 6:-1:1 \)

if \( \text{starts}(i)+w_{1(i)}/S(z) < d(i) \)

\[ w(i) = w_{1(i)}/S(z); \]

\( t_c(i) = z; \)

break;
end

\[ w(i) = w_{1(i)}/S(1); \]

\( t_c(i) = 1; \)
end
end

elseif \( E_{c+\text{add}1(i)}-E_{D1(i)} < 0 \)

for \( z = 2:6 \)

\( E_{\text{new}} = \text{PD(temp,z)}*w_{1(i)}/S(z); \)

if \( E_{c+\text{add}1(i)}-E_{\text{new}} > 0 \)

if \( \text{starts}(i)+w_{1(i)}/S(z) > d(i) \)

\[ w(i) = w_{1(i)}/S(z); \]

\( t_c(i) = z; \)

break;
else

for \( k = 6:-1:z \)

if \( \text{starts}(i)+w_{1(i)}/S(k) < d(i) \)

\[ w(i) = w_{1(i)}/S(k); \]

\( t_c(i) = k; \)

break;
end
end
end
end

\[ w(i) = w_{1(i)}/S(6); \]

\( t_c(i) = 6; \)
end
\[ ED(i) = PD(temp, tc(i)) \times w(i); \]
\[ finish(i) = starts(i) + w(i); \]

\begin{verbatim}
for k = ceil(starts(i)):1:floor(starts(i)+w(i)-1)
    add(i) = add(i) + y(k+1);
end

if Ec + add(i) - ED(i) > 0 && Ec + add(i) - ED(i) < Emax
    Ec1 = Ec;
    Ec = Ec + add(i) - ED(i);
    Ec2(i) = Ec;
elseif Ec + add(i) - ED(i) > Emax
    Ec1 = Ec;
    Ec = Ec + add(i) - ED(i);
    waste = Ec - Emax;
    Ec = Emax;
    Ec2(i) = Emax;
end

if finish(i) > d(i) || Ec1 + add(i) - ED(i) < 0
    D(i) = 0;
    count = count + 1;
else
    D(i) = 1;
end

end

time = finish(i);
end
\end{verbatim}

**Appendix B: Code for adaptive scheduling for Ed>0 using MATLAB**

\[ EH_{EWMA} = 0; \]
\[ \text{for } j = 1:40 \]
\[ \quad EH_{EWMA} = EH_{EWMA} + y(j); \]
\[ \text{end} \]
\[ D1 = \text{zeros}(1,t); \]
\[ EH_{AFP} = 6000; \]
\[ Ed = EH_{AFP} - EH_{EWMA}; \]
\[ \text{count1} = 0; \]
\[ \text{if } Ed > 0 \]
\[ \quad \text{for } i = 1:14 \]
\[ \quad \quad w_temp = \text{zeros}(1,t); \]
\[ \quad \quad \text{finish_temp} = \text{zeros}(1,t); \]
\[ \quad \quad \text{starts_temp} = \text{zeros}(1,t); \]
\[ \quad \quad \text{end} \]
\[ \quad \text{end} \]
\[ \text{end} \]
SI=zeros(1,t);
diff=zeros(1,t);

if D(i)==0
    task=i;
    for j=1:i
        if w1(j)==1
            temp=1;
        elseif w1(j)==2
            temp=2;
        else
            temp=3;
        end
        if starts(j)>=a(task) || finish(j)>a(task)
            if tc(j)>1
                for new=tc(j):-1:1
                    ED_new=PD(temp,new)*w1(j)/S(new);
                    if Ed<ED_new-ED(j)
                        NEW1=new+1;
                        break;
                    end
                end
                NEW1=new;
            end
            starts_temp(j)=starts(j);
            w_temp(j)=w1(j)/S(NEW1);
            SI(j)=NEW1;
            finish_temp(j)=starts_temp(j)+w_temp(j);
        for s=j+1:i
            starts_temp(s)= max(finish_temp(s-1),a(s));
            finish_temp(s)=starts_temp(s)+w(s);
        end
        diff(j)=finish(i)-finish_temp(i);
    end
end
[c,index]=max(diff);

w(index)=w_temp(index);
finish(index)=starts(index)+w(index);
for s=index+1:i
    starts(s)= max(finish(s-1),a(s));
    finish(s)=starts(s)+w(s);
end
if w1(index)==1
    temp1=1;
elseif w1(index)==2
    temp1=2;
else
    temp1=3;
end
if c>0 && SI(index)>tc(index)
    if ED(index)<(PD(temp1,SI(index))*w(index))
Ed = Ed - ((PD(temp1, SI(index)) * w(index)) - ED(index));
end
end
if finish(i) > d(i)
    D1(i) = 0;
    count1 = count1 + 1;
else
    D1(i) = 1;
end
end
av1 = count1 / t * 100;
stem(EH_AFP - EH_EWMA, av1)
hold on

Appendix C: Code for adaptive scheduling for Ed<0 using MATLAB

EH_EWMA = 0;
for j = 1:40
    EH_EWMA = EH_EWMA + y(j);
end
D1 = zeros(1, t);
EH_AFP = 5.6926e+003;
Ed = EH_AFP - EH_EWMA;
count1 = 0;

    D2 = zeros(1, 13);
    ct1 = 4;
    ct2 = 0;
    Ed = abs(Ed);

w1_temp = w1;
while ct1 > 1 || Ed > 0
    for i = 1:13
        finish_temp = finish;
        starts_temp = starts;
        if finish(i) == 0
            D2(i) = 100;
        else
            for j = i + 1:14
                if j == i + 1
                    starts_temp(j) = max(a(j), starts(i));
                    finish_temp(j) = starts_temp(j) + w(j);
                else
                    starts_temp(j) = max(a(j), finish_temp(j - 1));
                end
            end
        end
    end
    if Ed > 0
        Ed = abs(Ed);
        count1 = count1 + 1;
    end
end

hold on
```matlab
finish_temp(j)=starts_temp(j)+w(j);
end
end

for j=1:14
if j==i
    D2(i)=D2(i);
else
    if finish_temp(j)>d(j)
        D2(i)=D2(i)+1;
    end
end
end
end
[c1,index1]=min(D2);
in=find(D2==c1);
[c2,index2]=min(w1_temp(in));
task=in(index2);
starts(task+1)=max(a(task+1),starts(task));
finish(task+1)=starts(task+1)+w(task+1);

for j=task+2:14
    starts(j)=max(a(j),finish(j-1));
    finish(j)=starts(j)+w(j);
end
starts(task)=0;
finish(task)=0;
w(task)=0;
w1_temp(task)=100;
ct1=0;
for j=1:14
    if j==task
        D(j)=1;
    else
        if finish(j)>d(j)
            D(j)=0;
            ct1=ct1+1;
        else
            D(j)=1;
        end
    end
end
Ed=Ed-ED(task);
end
```
for k=1:14
    if finish(k)==0
        ct2=ct2+1;
    end
end
stem(abs(EH_AFP-EH_EWMA),(ct2/14*100))
hold on