School of Engineering

Dynamic Voltage Scaling Algorithms for Soft and Hard Real-time System

Lue Ik Hong

This thesis is presented for the Degree of
Master of Philosophy(Electrical and Computer Engineering)
of
Curtin University

December 2012

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made. This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Abstract

Real-time system is all around us, especially in 21st century era, mobile devices are becoming one of the most important real-time systems. However, the energy consumption has always been a challenge in designing a real-time system. Although several common power management techniques have been widely proposed, Dynamic Voltage Scaling (DVS) has not been investigated completely for further minimizing the energy consumption of microprocessor and prolong the operational life of real-time system. In this dissertation, several algorithms of DVS are proposed, including the workload prediction based DVS for soft real-time system and offline Convex Optimized based DVS for hard real-time system. For Workload Prediction based DVS, two adaptive algorithms are proposed based on the weighted least square error and the minimum mean square error methods, respectively. The algorithms of soft real-time system are implemented on a small scaled wireless sensor network (WSN), while the algorithms of hard real-time system are implemented on a simulation model. The experimental results shows that workload prediction based DVS have a good predction performance with a condition of good choice in number of workload samples and forgetting factor. As for Convex Optimized based DVS, it perform a good energy saving of CPU while guaranteeing deadline. The computational performance of the proposed method in term of time complexity is improved with the modified of problem formulation.

Acknowledgement

My deepest gratitude goes to my supervisor, Dr. Wong Kiing Ing, who has been tirelessly guiding, offering professional suggestion and motivating my work through all these months. I am able to finish my Project with his great patience, intelligence and nonstop encouragement during my whole study.

I thank to my parents, for their lovely tender cares, support, selfless love and direction. This dissertation will only be started and finished with them being excellent role models.

My appreciation goes to all my friends who have given me spiritual supports and the courage to finish this dissertation. I appreciate the unlimited love from my family members and love ones who have given me a beautiful life.

Contents

1.0 INTRODUCTION

1.1 Hard and soft real-time system

In 21st Century, the devices and machines have been integrated into real-time system. However, what is a real-time system? A real-time system is a system required to complete its work and deliver its services on a timely basis [1]. Each of the electrical systems such as electronic, control, power, signal processing, and telecommunication has varieties of real-time systems in order to deliver its services in timely manners. For example, an electronic device—smartphone, can be used for internet browsing, multimedia playing and gaming apart from the usual mobile phone functionalities. Thus, each of these functionalities is processed in a timely manner by using a real-time system.

Real-time systems can be categorized into two types, depending on its characteristics such as soft or hard real-time system. Generally, all kinds of work (such as signal processing, data acquisition and etc.) processed by real-time systems are called a *job*. Sometimes, real-time systems are overloaded with excessive jobs, therefore some of the jobs have to be scheduled for later execution. There are two main time definitions in this context: release time and deadline. The release time of a job is defined as the instant of time at which the job becomes available for execution. Meanwhile, the deadline of a job is defined as the instant of time by which its execution is required to be completed. A hard real-time system is a system which must finish executing the job before the specified deadline once it has been released. Failing to meet the deadline will bring a catastrophe performance.

An example of a hard real-time system is the automatic train braking controller, jobs assigned to the CPU has to be completed within the specified deadline. When the stop signal is triggered from sensors, the braking action must be activated in a distance away from the signal post at which the train must stop. The braking distance depends on the speed of the train and the safe value of deceleration. Implementing these information, the controller is able to compute the best time for the train to start braking. In the real-time system point of view, it is important to impose a constraint on the response time of the jobs. If the system cannot finish the jobs on time or not responsive to the incoming jobs, it will cause a catastrophe to happen in which human lives are endangered.

Soft real-time system on the other hand, can afford to miss some deadlines occasionally or tolerated abort of execution. In fact, the overall performance of the system will become less responsive due to the extension of time completion for the jobs. However, the designer or developer of soft real-time system is rarely required to prove the real time guarantees on that system. For instance, a frame of a video must be delivered in every 1/13 second time. The time difference between the displayed video frame and the presented sound should be less than 80msec. In fact, each new video stream that is transmitted through a network, is subjected to a timing constraint. If the network could not transmit the video frame within the timing constraint, the new stream will be rejected and the admission is requested again, with the condition of not violating the constraints of existing streams. However, users are willing to tolerate a few glitches of the video, and for that reason, timing constraints of most multimedia systems are measured on a statistical basis (e.g., the average number of lost frames per minute). In this case, the quality-of-service guarantee, validation requirement and the timing constraints defining the video quality are essentially "soft".

1.2 Energy consumption of real-time system

Since real-time system has become a major concern in each design, thus its energy consumption is also directly proportional crucial. Many applications impose severe power or energy constraints on embedded systems, such as Wireless sensor network (WSN). Although WSN has several benefits such as low cost, wireless, fast deployment characteristics and has been used for remote monitoring applications, its energy consumption remains as a major issue. This is because WSN platforms are powered by batteries, and they are inaccessible once they are deployed in the physical environment. Therefore, a good energy saving scheme for WSN nodes is essential in order to provide a long operational life, especially for at least two years.

1.3 Previous Works

Below are some of the most frequent approaches in minimizing the power consumption of WSN:

- Dynamic Power Management (DPM)

Dynamic Power Management (DPM) schemes have been presented in the literatures [2, 3, 4, 5]. The main idea is to switch off the electronic components (such as the radio transceiver) when it is not used, and switch back on again when it has been assigned for a task. DPM has been a very popular approach for WSN due to its significant energy saving performance and ease of implementation.

- Energy Harvesting

Energy harvesting is another approach for achieving the longevity of operational life, where the electrical energy of sensor node is harvested from the natural energy nearby, such as solar and water flow current [6, 7, 8, 9, 10, 11].

- Energy Aware Routing Protocol

Energy aware routing protocols have been presented in [12, 13] which the data aggregation route from node to node is adjusted based on the battery conditions of the sensor nodes around the network. This will prevent early battery drain of a particular node (dead node) due to rapid data aggregation, hence prolong the operational life of the whole system.

- Dynamic Voltage Scaling (DVS)

Although Dynamic Voltage Scaling (DVS) has been widely used in embedded system applications due to its effective energy saving for CPU, it has not fully investigated on WSN. Most microprocessor systems have time varying computational load, including WSN, therefore, by scaling the clock rate of a CPU corresponding to its CPU loads will result in lower power consumption.

1.4 Motivation

Although several common power management techniques have been widely proposed, Dynamic Voltage Scaling (DVS) has not been investigated broadly. DVS is a new trend for minimizing the energy consumption of microprocessor. Speed scheduling on real-time system is not a trivial task, as it has known to be NP-hard problem [14]. The potential of good DVS scheme could significantly save the energy usage of processor, especially WSN and mobile devices.

1.5 Objectives

The energy consumption of both hard and soft real-time systems based on Dynamic voltage scaling (DVS) is investigated in this thesis. Several algorithms of DVS are proposed and their performances are evaluated and discussed. For soft real-time system based algorithms, the experiment is carried out on a small scale WSN, while for hard real-time system experiment is carried out on simulation model.

1.6 An outline of the dissertation

This dissertation is organized in six chapters. The following Chapter discuss the introduction of real-time systems and CPU power model. Chapter 3 includes the related work of DVS. Meanwhile, Chapter 4 and Chapter 5 comprises of the presentation of workload prediction based DVS and offline convex optimization based DVS, respectively. The experimental results and discussion for the proposed DVS algorithms are discussed in Chapter 6. Lastly, conclusions and the outline of future work are detailed in Chapter 7 and Chapter 8.

2.0 REAL-TIME SYSTEM AND CPU POWER MODEL

In this chapter, the introduction of real-time system model and CPU power model are presented. In this dissertation, DVS algorithms are designed for real-time system processing periodic tasks. A periodic task is referred to as a task with regular arrival time, such as to read sensor data and to update the state of the real-time system on a regular basis. Therefore, a periodic task model is also included in this chapter.

2.1 Temporal parameters of real-time workload

The workload on microprocessors consist of several jobs *J*. Each job, J_i, is a unit of work to be allocated to a microprocessor time and other resources, respectively. A set of related jobs that execute to support a function of the system is denoted as a task, T_i.

The parameters of a hard real-time job and task are identified at all times, otherwise the system will be impossible to ensure whether it meets the hard real-time guarantee. The following are the frequently used terms of temporal parameters of a job J_i:

- **Release time (or arrival time) r_i**: is the instant of time at which the job becomes eligible for execution. The respective release time of the job may be jittery (sporadic). This means that r_i is in the range of $[r_i-, r_i+]$ in which only the range of r_i is known but not the actual value of r_i. More elaboration of this term is discussed in section B.
- **Absolute deadline d_i**: d_i is the instant of time by which the job must complete.
- **Relative deadline D_i**: D_i is the maximum allowable response time of the job.
- **Execution time e_i**: e_i is the amount of time required to complete the execution of J_i when it executes alone and has all the resources it requires. The execution time of J_i may vary in range such that e_i is in the range of $[e_{i-}, e_{i+}]$. This corresponding range is known but not the actual value of e_i. Some models assume that J_i always executes for the maximum amount of time, e_{i+}; or in other words, Worst Case Execution Time (WCET).
- **Pre-emptivity:** A job is pre-emptible if its execution can be suspended at any time to allow the execution of other jobs and, later on, can be resumed from the point of suspension. A job is non-pre-emptible if it must be executed from start to completion without being interrupted.

2.2 Jittered, fixed, and sporadic release times

In most of the systems, the time when a job is released is unknown. In other words, the actual release time r_i of each job J_i is not known; only the range of r_i $[r_{i-}, r_{i+}]$ can be identified. r_{i-} is denoted as the earliest release time, and r_{i+} is denoted as the later release time possible. The range of r_i is called jitter, or jittered release times.

However, the jitter sometimes is so small compared with the values of other temporal parameters. It can therefore be ignored. For practical purposes, the actual release time of each job can be approximated by either the earliest or the later release time. We called this kind of release time to be fixed.

Almost every real-time system is required to respond to the external events which occurred at random instants of time. When such an event occurred, in response, the system will execute a set of jobs. The release times of these jobs are not known until the event triggering them occurs. Such jobs are called sporadic jobs or aperiodic jobs because they are released at random time Instants.

2.3 Periodic task model

In the periodic task model, tasks are executed repeatedly in regular time intervals to comprise a function of the system. Specifically, each periodic task is denoted as T_i. The period p_i of the periodic task T_i is the minimum length of all time intervals between release times of consecutive jobs in T_i. Meanwhile, the execution time, e_i is the maximum execution time of all jobs in T_i.

The *n*-th number of tasks in the system is denoted by $T_1, T_2, \ldots, T_n$. And each individual jobs in a task T_i is denoted by $J_{i,1}, J_{i,2}, \ldots, J_{i,k}$ in which $J_{i,k}$ being the *k*-th job in T_i. The release time, $r_{i,1}$ of the first job $J_{i,1}$ in each task T_i is called the phase of T_i where it is denoted as ϕ_i, and $\phi_i = r_{i,1}$, respectively. In general, different task has different phases. However, there exist some tasks which are in the same phase.

The maximum number of jobs, *N* in each hyper-period is equal to

$$\sum_{i=1}^{N} H/p_i. \quad (2.1)$$

where *H* is the hyper-period of the periodic task and *p* is the period of the tasks. For example, the length of a hyper-period of three periodic tasks with periods 3, 4, and 10 is 60. Thus, the total number of jobs, *N* in the hyper-period is 41.

The utilization of the task T_i, is denoted as u_i, where $u_i = e_i/p_i$ is the fraction of time a truly periodic task keeps a processor busy. The total utilization denoted as $u_{total} = \sum_{i=1}^{n} e_i/p_i$, respectively. For instance, if the execution times of the three periodic tasks are 1, 1, and 3, and their periods are 3, 4, and 10, thus their corresponding utilizations will be 0.33, 0.25 and 0.3. Meanwhile, the total utilization of the tasks is 0.88.
A job in T_i which is released at *t* must complete D_i units of time after t; D_i is the (relative) deadline of the task T_i.

2.4 Real-Time Scheduling Algorithms and Validation Tests

A scheduling algorithm in real-time systems allocates processor time to tasks in order to meet the timing constraints. There are numerous publications on the real-time scheduling algorithm [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]. The commonly used terms in the publications are summarized below:

Feasibility, optimality & schedulability

A task set *T* is said to be *feasible* on a hardware platform if there exist some way of scheduling that meets all the deadlines of the corresponding platform. *T* is said to be *schedulable* on a hardware platform by algorithm *A*, provided *A* is capable for correct scheduling on that platform such that A can meet all the deadlines of *T*. Furthermore, *A* is said to be an *optimal* scheduling algorithm if ***A*** can correctly schedule every feasible task system for every hardware platform.

Maximum schedulability utilization

A system is schedulable by an algorithm if the algorithm could always produces a feasible schedule. A feasible and schedulable system is valid if it is schedulable by some algorithm. However, now the question is—how large is the total utilization of a system can be in order for the system to be surely schedulable, in other words, what is the maximum schedulability utilization of the algorithm.

Schedulability tests

It is a validation tests for the purpose of validating that the given application system can indeed meets all its hard deadlines when scheduled accordingly to the chosen scheduling algorithm. If a schedulability test is efficient, it then can be used as an on-line acceptance test.

Priority-driven scheduling approach

Priority-driven scheduling algorithms refer to a large class of scheduling algorithms that never leave any resource idle intentionally. A resource will stay idle only when there is no job ready for execution. Scheduling is decided based on events such as occurrences of releases and completions of job.

Priority-driven algorithms are classified into two types: fixed priority and dynamic priority. A fixed-priority algorithm assigns the same priority to all the jobs in each task. In other words, the priority of each periodic task is fixed relative to other tasks. Conversely, a dynamic-priority algorithm assigns different priorities to the individual jobs in each task. Hence the priority of the task with respect to that of the other tasks changes as jobs are released and completed. Both examples of well-known dynamic and fixed priority algorithms are discussed below. For the rest of the dissertation, we are going to refer a periodic task T_i with period p_i, execution time e_i by the 2-tuple (p_i, e_i).

For example, (3, 6) is a periodic task whose period is 10, and execution time is 3. By default, the phase of each task is 0, and its relative deadline is equal to its period.

Dynamic-priority driven

The most well-known dynamic-priority algorithm—Earliest-deadline-first (EDF) algorithm assigns priorities to individual jobs in the tasks according to their absolute deadlines. Figure 2.1 shows the example of EDF schedule, with the corresponding tasks $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$.

- At time 0, the first jobs $J_{1,1}$ and $J_{2,1}$ of both tasks are ready. The absolute deadline of $J_{1,1}$ is 2 while the deadline of $J_{2,1}$ is 5. Consequently, $J_{1,1}$ has a higher priority to execute first. Thus when $J_{1,1}$ completed, $J_{2,1}$ begins to execute.
- At time 2, $J_{1,2}$ is released, and its deadline is 4, earlier than the deadline of $J_{2,1}$. Hence, $J_{1,2}$ is placed ahead of $J_{2,1}$ in the ready job queue. $J_{1,2}$ pre-empts $J_{2,1}$ and executes.
- At time 2.9, $J1_{,2}$ completed. The processor then executes $J_{2,1}$.
- At time 4, $J_{1,3}$ is released; its deadline is 6, which is later than the deadline of $J_{2,1}$. Hence, the processor continues to execute $J_{2,1}$.
- At time 4.1, $J_{2,1}$ completed, the processor starts to execute $J_{1,3}$, and so on.

Note that the priority of T_1 is higher than the priority of T_2 from time 0 until time 4.0. T_2 starts to have a higher priority at time 4.0. When the job $J_{2,2}$ is released, T_2 again has a lower priority. Hence, the EDF algorithm is a task-level dynamic-priority algorithm.

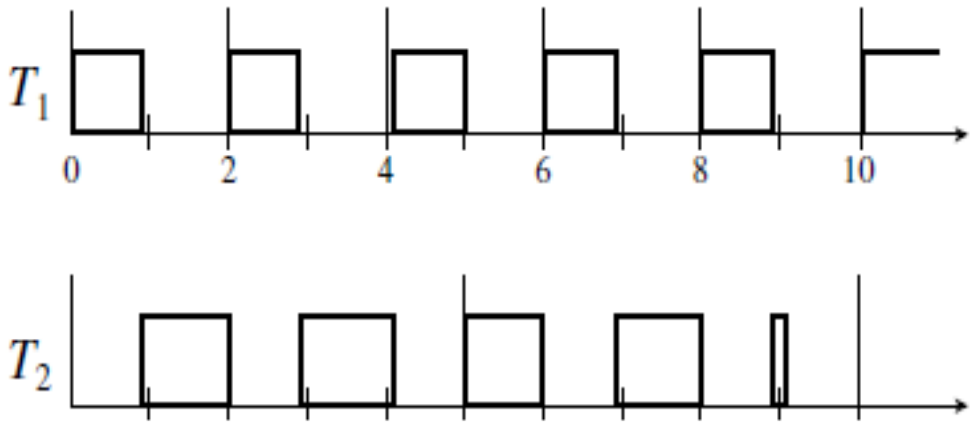

Figure 2.1 An earliest-deadline-first schedule of (2, 0.9) and (5, 2.3).

Static-priority driven

A well-known fixed-priority algorithm is the *rate-monotonic* algorithm. This algorithm assigns priorities to tasks based on their periods: the shorter the period, the higher the priority.

The *rate* (of job releases) of a task is the inverse of its period. Hence, the higher its rate, the higher its priority are. We will refer to this algorithm as the RM algorithm for short and a schedule produced by the algorithm as an RM schedule.

For an example, the schedule in Figure 2.2 is for the tasks T_1 = *(*2, 0.9*)* and T_2 = *(*5, 2.3*)*. The tasks are in phase. According to the RM algorithm, task T_1 has a higher priority than task T_2. Consequently, every job in T_1 is scheduled and executed as soon as it is released. The jobs in T_2 are executed in the background of T_1.

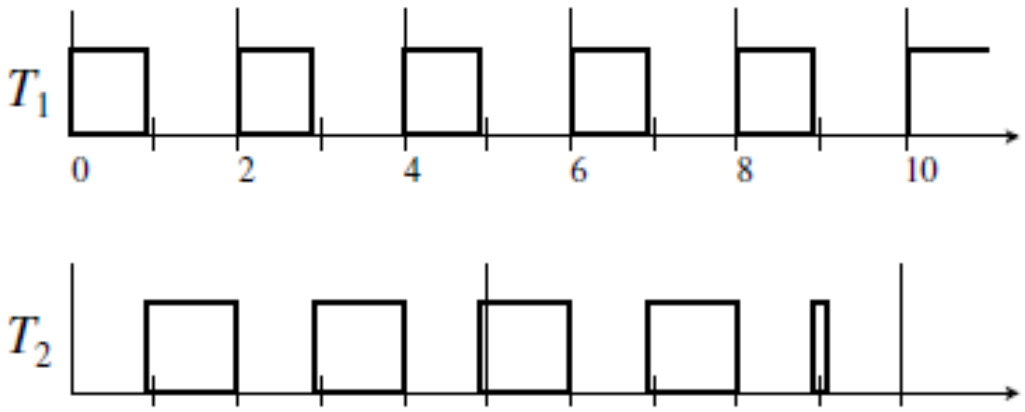

Figure 2.2 RM schedule of T1 = (2, 0.9) and T2 = (5, 2.3).

EDF and RM algorithm comparison

For a uniprocessor, the maximum schedulable utilization of RM for periodic task systems is $U_{RM} = N \cdot (2^{1/N} - 1)$ [1]. Notice that from the equation, the maximum utilization decreases when number of task, N increases, and will eventually converge to $ln\ 2 \approx 0.69$ when N is ∞. But the schedulable utilization of EDF is 1.0 for all N. Thus, EDF has better performance than RM. Furthermore, since there is no algorithm which could correctly schedule any task system with total system utilization more than 1, EDF is considered optimal among all scheduling algorithms.

2.5 CPU Power Model

In CMOS devices, power consumption is proportional to the voltage square, which is given by [17]:

$$P_{cmos} = C_L N_{SW} V_{DD}^2 \tag{2.2}$$

Where

- C_L is the circuit output load capacitance
- N_{sw} is the number of switches per clock cycle
- V_{DD} is the supply voltage
- f is the clock frequency

The gate delay in the system is followed by:

$$T = k\frac{V_{DD}}{(V_{DD}-V_T)^2} \tag{2.3}$$

Where

- k is technology dependant constant
- V_T is threshold voltage

From the observation of both equations above, the speed of the processor is proportional to f, and inversely proportional to the gate delay. Thus, speed versus power curve can be derived in Figure 2.3.

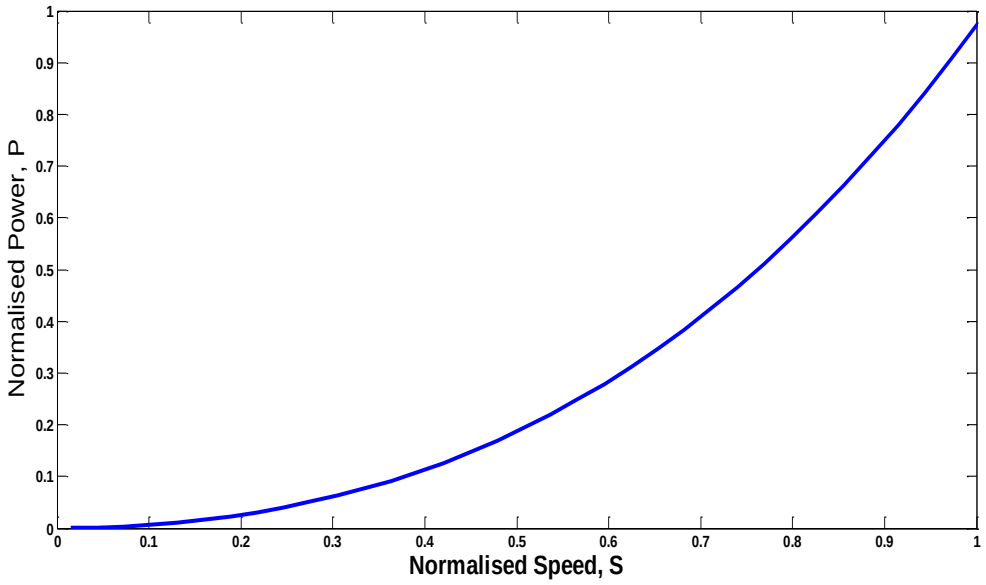

Figure 2.3 Power *vs* Speed of CMOS Device

In this dissertation, V_T is set as 0.8V and maximum supply as 5.0V. The normalized speed of the processor at maximum frequency, and the normalized power consumption at maximum frequency and maximum supply voltage were both set to be 1. By combining the gate delay equation and the Power of CMOS device, the normalized power function (P) as a function of speed (S) is shown as below:

$$P(S) = 0.248S^3 + 0.256 * S + (0.014112S^2 + 0.0064S)(\sqrt{311.16S^2 + 282.24S}) \quad (2.4)$$

3.0RELATED WORK

There are two categories of DVS which are known as inter-task and intra-task. An inter-task DVS is basically a scheme to assign one specified speed to the task, and it is not changed during the task execution. For an intra-task DVS, different speeds can be assigned to a same task along the task execution. Some of the research papers on DVS are presented in [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44].

An extension of static solutions on periodic and aperiodic task models were proposed in various papers [45- 47, 75]. Pering et al proposed a DVS algorithm on a microprocessor [45]. The result showed that DVS allows a microprocessor to save energy by operating at optimal voltage for a given task. In [46], the authors proposed both aperiodic and periodic tasks on variable voltage processor to optimize power consumption while ensuring all periodic tasks meet their deadlines and to accept as many periodic tasks as possible, as long as it is not compromising the feasibility of the system.

In paper [47], the authors proposed a non-pre-emptive power aware scheduling and also the synthesis technique that addresses the selection of the processor core. Meanwhile, the determination of the instruction, data cache size and configuration so as to fully exploit dynamically variable voltage hardware. And it is the only paper that considers the aspect of instruction and data cache size that will further improve the energy saving of the processor.

In paper [48], it was the first paper addresses the further voltage scaling by allocating slack time of the early finished task to other available tasks. The proposed run-time mechanism is simple to implement in a system kernel while resulted as significantly energy saver. Thus, there goes the attention of researchers to further improve on this approach.

In [49], the authors proposed a reclaiming algorithm (Cycle-Conserving EDF) and a speculation-based algorithm (Look-Ahead EDF). These inter-task DVS algorithms are based on updating and predicting the instantaneous utilization of the periodic task set.

The most popular DVS method on hard real-time system is published in [50]. Their proposed method includes three components: a static(offline) solution to compute the

optimal speed assuming worst case execution time (WCET) for each arrival, an online speed reduction mechanism to reclaim energy by adapting to the actual workload, and the online, adaptive and speculative speed adjustment mechanism to anticipate early completions of future executions based on the average-case workload information. The main idea behind their proposed online DVS is to the slack time from early-finished task will be given to strictly lower priority task. The method is found to be efficient while guarantee feasibility for EDF scheduling.

In [63], the authors proposed a heuristic algorithm on DVS by using just two voltage levels to achieve energy saving consumption while still guarantee to meet tasks' deadlines.

In [51], the authors presented an analytical model of general tasks for DVS assuming job timing information is known only after a task release. It models the voltage scaling process as a transfer function-based filtering system, which facilitates the design of two efficient scaling algorithms—time-invariant and time variant scaling algorithm.

In paper [52], the authors proposed a scheduling scheme for reduced energy of hard real-time tasks with fixed priorities assigned in a rate monotonic schedule. The authors targeted to scale down the voltage based on offline and online decisions. For their offline scaling method, the authors employed stochastic data of the task to derive energy efficient schedules based on the probability theory.

In [53], the authors presented an algorithm to compute a near optimal constant slowdown factor for a hard real-time system assuming the deadline of the task can be lower than its period. The authors proposed the bisection method to compute constant static slowdown factors and an algorithm based on the ellipsoid method to compute uniform slowdown factors.

In paper [54], Lin et al. proposed ALT-DVS which is a DVS that take consideration of the leakage and temperature of CPU chips. The algorithm leverages the on-chip temperature sensor and the established system-level thermal model to determine the appropriate operating voltage and frequency for a given task. Such that each task can be completed before its deadline while the system does not overheat.

In [55], the authors issued that the memory-bound multimedia applications are becoming popular in handheld devices, thus the DVS policies should consider the so-called ‘memory wall’ problem to maximize the energy gain. The authors proposed a memory-aware DVS (M-DVS) technique that takes the memory wall problem fully into consideration. The experimental results on a PDA shows that M-DVS can reduce 8% of additional power consumption, compared to the conventional DVS without any QoS degradation for handling multimedia clips.

Although many DVS algorithms have been developed for real-time systems with periodic tasks, none of them can be used for a system with both periodic and aperiodic tasks because of the arbitrary temporal behaviours of aperiodic tasks. In [56], Shin at el. Describes dynamic voltage scaling (DVS) algorithms for real-time systems with both periodic and aperiodic tasks. The paper proposed off-line and on-line DVS algorithms that are based on existing DVS algorithms. The proposed algorithms utilize the execution behaviours of scheduling servers for aperiodic tasks. Since there is a trade-off between the energy consumption and the response time of aperiodic tasks. The proposed algorithms focus on bounding the response time degradation of aperiodic tasks even though the response time is delayed by stretching the task execution to get high energy savings in mixed task sets.

Jacob et al. proposed PACE (Processor Acceleration to Conserve Energy)—a New Approach to Dynamic Voltage Scaling in [57]. The proposed formula specifies increased speed as the task progresses. The optimal formula depends on the probability distribution of the task’s work requirement with condition that the speed be varied continuously. They also presented methods for estimating the task work distribution and evaluate how effective the authors are on a variety of real workloads. The authors showed how to approximate the optimal continuous schedule with one that changes speed a limited number of times.

In [58], the authors proposed a novel workload estimation technique for DVS. This technique is based on the Kalman filter and can estimate the processing time of workloads in a robust and accurate manner by adaptively calibrating estimation error of

feedback. Experimental results showed that this approach can reduce energy consumption by 57.5% on average, only with negligible deadline miss ratio (DMR) around 6.1%, 11.7% in the worst case.

In [62], the authors proposed DVS by predicting the incoming workload traces through adaptive filtering theory. Few filter coefficient have been proposed and compared, and Least Mean Square have been chosen due to its light weight computation. In [58], the authors proposed a DVS scheme for jittered controlled real-time scheduling. They transformed jittered periodic real-time tasks into a tree structure.

For intra-task DVS, Dongkun et al proposed a novel intra-task dynamic voltage scheduling (IntraDVS) framework for low-energy hard real-time applications [59]. Based on a static timing analysis technique, the proposed approach controlled the supply voltage within an individual task boundary. By fully exploiting all the slack times, a scheduled program by the proposed technique always completes its execution near the deadline, thus achieving a high energy reduction ratio. The problem formulation of IntraDVS is first presented and two heuristics are proposed: one based on worst-case execution information and the other on average-case execution information.

In [60], the authors presented a novel light-weighted energy-efficient EDF schedulers for processors. The proposed EDF scheduler performed an online intra-task and inter-task frequency scaling at the same time.

In paper [61], the authors addressed the problem of variable execution time in tasks under the Fixed-Priority scheduling with pre-emption threshold (FPPT). FPPT allows a task to disable pre-emptions from tasks up to a specified pre-emption threshold priority. Tasks with a priority greater than the pre-emption threshold priority are still allowed to pre-empt. The pre-emption threshold scheduling model has been shown to reduce the run-time costs by eliminating unnecessary task pre-emptions. The authors thus proposed an intra- task voltage scheduling that exploit the stochastic data of the tasks completion tasks.

4.0 WORKLOAD PREDICTION BASED ON LINEAR PREDICTION

In this chapter, a workload prediction based on linear prediction for a soft real-time system is proposed. Due to the nature of WSN in which few deadline misses are acceptable, WSN is used for the study of a soft real time system. The proposed workload prediction method will be experimented on a small-scale WSN for evaluating its performances.

4.1 Operating System for WSN

To determine whether an Operating System (OS) is suitable to provide real-time guarantee, it depends on the scheduling mechanism and execution models of the OS. Real-time guarantee is essential for an OS used in the real-time systems. This is because real-time systems have its deadlines, thus missing the deadline will cause system failure in a worst situation.

An OS that is eligible to provide real-time guarantees should have a real-time scheduler policy (i.e., Earliest-Deadline-First) and a good execution model that could handle task and being responsive in time constrained environment (i.e., thread based driven model). In [65], a survey regarding OS for WSN is made. Out of seventeen OS that have been surveyed, only five OS have real-time guarantees. Since the majority of the OS based in WSN are not eligible to apply on real-time constrained applications, therefore WSN can be considered as soft real-time system.

4.2 WSN System Model

A system model of DVS for wireless sensor node is shown in figure 3.1. The main components of a wireless sensor node consist of a radio transceiver, microcontroller, Flash memory and a number of sensors. Each of the components has their own tasks-waiting for the microprocessor to execute. In OS environment, the tasks will be queued and scheduled by an OS scheduler. An OS scheduler's job is to determine which task should be executed first or pre-empted, based on its scheduling policy. Assume each of *n* components generates task at a specified rate r_k (k = 1, 2, … , n). The tasks rate that arrives at processor is $r = \sum r_k$. Then the OS kernel will compute the workload, *w* over an observation frame, *T*. The workload is computed using the following equation:

$$w = \frac{T - idle\_cycle}{T} \tag{4.1}$$

where *idle_cycle* denotes the idle cycle of the processor in the observation timeframe. Based on the historical workloads that have be calculated, a future workload can be predicted. The DVS module then adjusts the voltage, *v* and the frequency, *f* for the microprocessor, based on the predicted future workload, w_p. A direct current or direct current converter is used to adjust the desirable voltage level from the fixed voltage supply that corresponding to the current workload state of the processor, as shown in figure 4.

4.3 Workload prediction algorithm

4.3.1 Ergodic Process

A stochastic process is assumed to be ergodic if its statistical properties can be deduced from a single, sufficiently long sample of the process. For example, a discrete process *x(n)* has mean,

$$\mu = E[x(n)] \tag{4.2}$$

And autocovariance,

$$r_x(\mathrm{K}) = E[(x(n) - \mu)(x(n + \mathrm{K}) \tag{4.3}$$

that does not change with time, is assumed to be ergodic [66].

In this paper, ergodic process is proposed as the context for workload prediction. It is reasonable because in most WSN applications, the event is periodic nature. For example, a remote monitoring system will acquire sensing data and relay it across

network once for every time interval. Hence, it is understandable to assume that the workload will have the same mean and autocovariance across time.

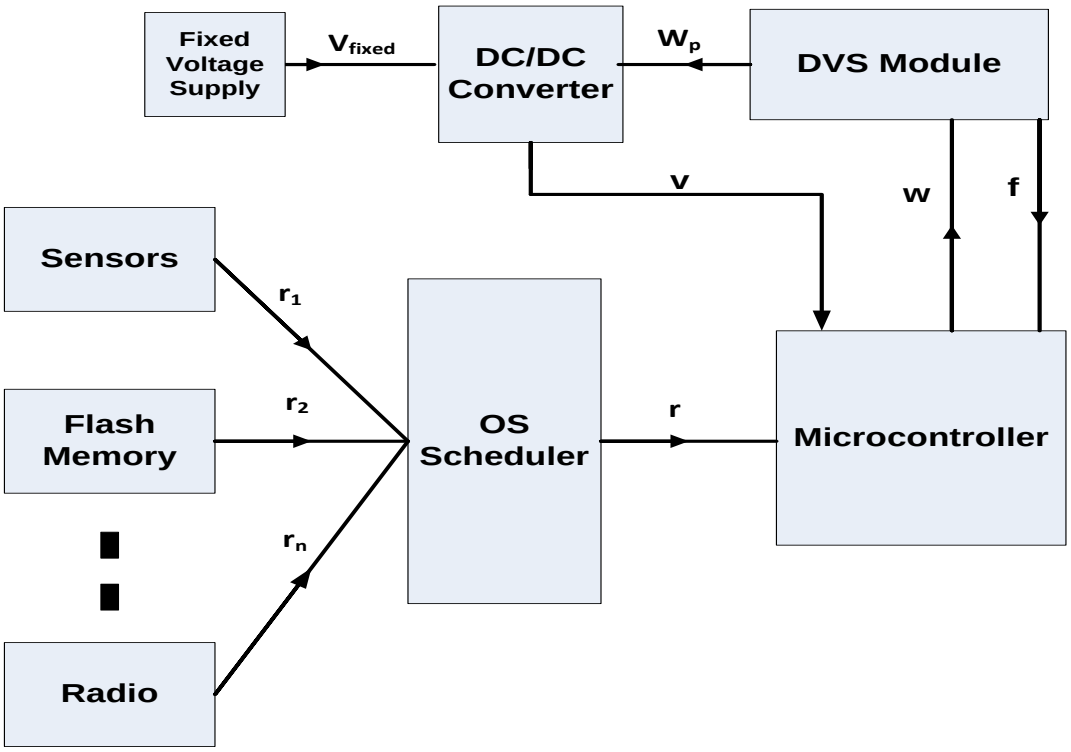

Figure 4.1 DVS System Model for Sensor Node

4.3.2 Linear Prediction

To predict a future workload, observation of the previous workload history, *w* has to be done. Assume the previous workload samples are *w(n – 1), w(n – 2),….., w(n – M)*, where M is the number of past samples used to predict the future workload, $\widehat{w}$(n). We may assume the predicted workload as a linear function of the given M samples. Hence, the equation for predicting the future workload is given by [67]

$$\widehat{w}(n|n-1, n-2, \dots, n-M) = \sum^{M} a_k w(n-k) \tag{4.4}$$

Where a_k are constant coefficients for the predictor. Different methods had been proposed to determine a_k [67]. In this paper, two algorithms are considered: Minimum Mean Square Error and Weighted Least Mean Square.

Minimum Means Square Error (MMSE) – It is obvious that we need to minimize the error between predicted value and actual value. In Probability theory, a common measure for prediction error is the Root Mean Square (RMS) Error. So, to minimize prediction RMS error, Weiner- Hopf equations came out as following,

$$\boldsymbol{R}\mathbf{a} = \boldsymbol{r} \tag{4.5}$$

where,

$$\mathbf{a} = [\mathrm{a}_1 \quad \mathrm{a}_2 \quad \mathrm{a}_3 \ \ldots \ \mathrm{a}_3]^T \tag{4.6}$$

$$\boldsymbol{r} = [\, R_{xx}(1) \quad R_{xx}(2) \quad R_{xx}(3) \quad \ldots \quad R_{xx}(M)]^T \tag{4.7}$$

$$\boldsymbol{R} = \begin{bmatrix} R_{xx}(0) & R_{xx}(-1) & \ldots & R_{xx}(1-M) \\ R_{xx}(1) & R_{xx}(0) & \ldots & R_{xx}(2-M) \\ R_{xx}(2) & R_{xx}(1) & \ldots & R_{xx}(3-M) \\ . & . & \ldots & . \\ . & . & \ldots & . \\ R_{xx}(M-1) & R_{xx}(M-2) & \ldots & R_{xx}(0) \end{bmatrix} \tag{4.8}$$

$R_{xx}(k)$ denotes the autocorrelation function of the sequence w(n) for a lag k. Note that $R_{xx}(-k) = R_{xx}(k)$, since we assume the process to be stationary. Therefore, to solve the corresponding coefficient **a**, $R_{xx}(k)$ up to order M have to be determined.

To determine $R_{xx}(k)$ for the input sequence w(n), we need to have apriori knowledge of the entire process. In previous section, we have assumed the workload process to be ergodic. Hence $R_{xx}(k)$ is derived as follow,

$$R_{xx}(k) = \frac{1}{N-k}\sum^{N} w(l)w^*(l-k) \tag{4.9}$$

Weighted Least Square Error (WLSE) - unlike MMSE, autocorrelation of the sequence is not needed. Hence, WLSE is more practical for real time implementation compared to the former since priory knowledge of the process is no needed anymore.

The idea is to minimize the sum of square error in which the error is defined as the difference between actual value and the predicted value. Thus, we have

$$e(n) = w(n) - \hat{w}(n) \tag{4.10}$$

And the cost function that we need to minimize, as a function of e(n) and dependent on the predictor's coefficients. We have

$$\mathrm{C}(\mathbf{a}) = \frac{1}{2}\sum_{n=1}^{k} \alpha_n e^2(n) \tag{4.11}$$

where $\alpha_n=\alpha^{k-1}$ is the "forgetting factor", which will give the lesser weights to the older error samples, and $0 < \alpha < 1$. New set of predictor's coefficients will be computed at each time t = k. In order words, it will adaptively change the coefficient in order to minimize the cost function in every time interval.

WLSE algorithm is outlined as below,

For k = 2 to ∞,

a) Calculate the current predicted workload

$$\hat{w}(n) = \mathbf{a}_M^T(k-1)u(k) \tag{4.12}$$

where

$$u(k) = [w(k-1) \quad w(k-2) \quad \ldots \quad w(k-M)]^T \tag{4.13}$$

b) Update the coefficient vector

$$\mathbf{a}_M(k) = \mathbf{a}_M(k-1) + \frac{P(k-1)u(k)}{\alpha + u^T(k)P(k-1)u(k)}[e(k)] \tag{4.14}$$

where

$$e(k) = w(k) - \hat{w}(k) \tag{4.15}$$

c) Update the P matrix

$$P(k) = \frac{1}{\alpha}\left\{P(k-1) - \frac{P(k-1)u(k)u^T(k)P(k-1)}{\alpha + u^T(k)P(k-1)u(k)}\right\} \tag{4.16}$$

5.0 OFFLINE OPTIMAL INTER TASK DVS

In the previous chapter, it shows that DVS by predicting the future workload based on its history profile is not a promising method, especially in hard real-time system. In this chapter, an offline optimal inter task DVS is presented.

The algorithm is offline because the optimal speed of each task is allocated based on prior knowledge of each task's parameters, (C_i, P_i). The prior knowledge of execution time, C_i is assumed to be Worst case execution time (WCET). This method ensures deadline guarantee of the tasks even though CPU speed is slowing down. Hence, the DVS proposed in this chapter can be applied on hard real-time system.

Although several offline DVS have been presented, an investigation is carried out on whether the problem formulation presented in [43] that has been commonly used, can be improved. Moreover, a modified problem formulation to improve the computation performance is proposed in the dissertation as well.

5.1 Problem formulation

We want to compute task slowdown factors that minimize the energy consumption of the processor. The total energy consumption of CPU when task is executed at a normalized speed, S_i (highest normalized speed is equal to 1) is given by:

$$e(S) = \sum^{n} \frac{H}{P_i} \frac{C_i}{S_i} P(S_i)$$

- H is hyper-period of the tasks
- P_i is period of the task
- C_i is WCET of the task
- P (S_i) is the power consumption of CPU at a given slowdown speed.

Based on the EDF scheduling policy, a task-set of *n* independent periodic tasks is feasible at speed S_i, for task T_i, if the utilization under slowdown is not more than 1,

which is stated at (5.2) and (5.3). S_i has to be more than or equal to minimum speed, S_{min} and less than or equal to 1, stated at (5.4).

$$U_{tot} \leq 1 \tag{5.2}$$

$$U_{tot} = \sum_{i=1}^{k} \frac{C_i}{S_i} \cdot \frac{1}{P_i} \tag{5.3}$$

$$S_{min} \leq S_i \leq 1 \tag{5.4}$$

However, it can be proven that the inequality constraint of (5.2) can be written as equality constraint, shown in (5.5), without affecting the optimization performance and results. The proof of the statement is written below.

$$U_{tot} = 1 \tag{5.5}$$

Proof: From equation (2.1) in chapter 2, power consumption $P(S_i)$ is convex function of speed, S_i. Hence, it is understandable that, in order to minimize the energy consumption of CPU, it is better for the speed allocation of each task to be lower. Now, considering the inequality constraint of equation (5.2) and (5.3). The total utilization bound U_{tot} is inversely proportional with S_i and it is upper bounded by 1. In other words, value S_i should be as low as possible, as long as U_{tot} does not exceed 1. Hence the optimal solution S^* can be found when the utilization constraint is equal to 1.

The reason of transforming inequality constraint to equality constraint is to reduce the feasible solution space. This is because, the solution space is directly proportional to the computation throughput. The larger the solution space, the more iteration and larger the computation throughput will be. The performance comparison between equality constrained and inequality constrained is further discussed in the experimental result and discussion chapter.

Now a question arises. Will the minimization function performs better if C_i of each task is further thin sliced into smaller pieces and different speed is allocated on each of them? Consider the following example with task set $\{T_1, T_2 \ldots T_n\}$, assuming each task has C_i.

If each C_i are thin sliced into m pieces, so we have $C_{i,1}$, $C_{i,2}$,…$C_{i,m}$ such that $C_i = \sum_{j=1}^{m} C_{i,j}$. Thus, we have

$$e(S) = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{H}{P_i} \left[\frac{C_{i,j}}{S_{i,j}} P(S_{i,j}) \right] \tag{5.6}$$

Therefore, in order to answer the question whether equation (5.6) is better function formulation than (5.1), we look at the following scenario. Says T_1 has optimal solution, $\mathbf{S}^*$ with S_1 is equal to 0.6, and C_1 has been thin sliced into 2 pieces (m=2) such that $C_{1,1}$ and $C_{1,2}$. Next, we need to find the optimal solution for $S_{1,1}$ and $S_{1,2}$ subject to constraint of utilization bound below:

$$\frac{C_{1,1}}{S_{1,1}} + \frac{C_{1,2}}{S_{1,2}} \leq \frac{C_{1,1} + C_{1,2}}{0.6} \tag{5.7}$$

Based on the proof mentioned, the inequality constraint can be written into equality constraint, thus we have

$$\frac{C_{1,1}}{S_{1,1}} + \frac{C_{1,2}}{S_{1,2}} = \frac{C_{1,1} + C_{1,2}}{0.6} \tag{5.8}$$

Now, simplify the equation above, we have

$$C_{1,1}S_{1,1}S_{1,2} + C_{1,2}S_{1,1}S_{1,2} = 0.6C_{1,1}C_{1,2} + 0.6C_{1,2}C_{1,1} \tag{5.9}$$

From the observation, you get $S_{1,1}$ =0.6 and $S_{1,2}$ =0.6. Note that, the optimal solution of S_1^* =$S_{1,1}$ = $S_{1,2}$. In other words, equation (5.6) produces the same optimal solution as equation (5.1). Hence, we are going to use the equation (5.1) as the minimization function for the remaining work.

To summarize from the above, the finalized optimization problem can be stated as below:

$$\textbf{Minimize:} e(S) = \sum_{i=1}^{n} \frac{H}{P_i} \frac{C_i}{S_i} P(S_i) \tag{5.10}$$

$$\textbf{Subject to:} \quad \sum_{i=1}^{n} \frac{C_i}{S_i} \cdot \frac{1}{P_i} = 1, \tag{5.11}$$

$$S_{min} \leq S_i \leq 1 \quad , i=1, \dots, n \tag{5.12}$$

5.2 Introduction of Convex optimization

Convex minimization is a subset of optimization. It studies the minimization problem that is convex functions over convex sets. The convexity property makes the optimization easier to solve compared to the general cases, because any local minimum found in the problem must be a global minimum. In other words, a global solution of the problem is guaranteed to be found [68, 69].

The minimization problem of energy consumption that was discussed above, is considered a convex nature, due to the convexity of the Power function, $P(S_i)$. The graph of $P(S_i)$ is shown in Chapter 2.

5.2.1 Interior point method algorithm

Interior point methods are a certain class of algorithms to solve linear and nonlinear convex optimization problems [70, 71].

The main idea of interior point method is to solve a sequence of approximate minimization problems. Consider the original problem below:

$$\textbf{Minimize:} e(S) = \sum_{i=1}^{n} \frac{H}{P_i} \frac{C_i}{S_i} P(S_i) \tag{5.13}$$

$$\textbf{Subject to:} \; h(S) = \sum_{i=1}^{n} \frac{C_i}{S_i} \cdot \frac{1}{P_i} - 1 = 0, \tag{5.14}$$

$$S_i - 1 \leq 0 \quad , i = 1, \dots, n \tag{5.15}$$

$$-S_i \leq -S_{min} \quad , i = 1, \dots, n \tag{5.16}$$

$$g(S) = \begin{bmatrix} S_i - 1 \leq 0 & , i = 1, \dots, n \\ -S_i \leq -S_{min} & , i = 1, \dots, n \end{bmatrix} \tag{5.17}$$

Note that (5.14) is the same as (5.11), while (5.15) and (5.16) is the same as (5.12). For the remaining chapter, *g(S)* is denoted as the vector of inequality constraints (5.15) and (5.16), *h(S)* is denoted as the vector of equality constraint (5.14). Next, we add slack variables, w into *g(S)* to become equality constraint:

$$\begin{aligned} g(S) + w &= 0, \\ w &\geq 0, \end{aligned} \tag{5.18}$$

w represents the vector of w_p, with $p = 1, \dots, 2n$. Note that the slack variables are as many as inequality constraints.

The constraints (5.18) are then replaced with a logarithmic barrier term in the objective function, resulting in the transformed problem below:

$$\begin{aligned} \text{Minimize:} \quad & e(S) - \mu \textstyle\sum_{p=1}^{2n} \ln(w_p) \\ \text{subject to:} \quad & h(S) = 0, \\ & g(S) + w = 0. \end{aligned} \tag{5.19}$$

Incorporating the equality constraints into objective function using Lagrangian Multipliers to form Lagrange function:

$$L_u(S, w, \gamma, \beta) = e(S) - \mu \textstyle\sum_{p=1}^{2n} \ln(w_p) - \gamma^T (g(S) + w) - \beta^T h(S) \tag{5.20}$$

The first order conditions for a minimum are:

$$\nabla_s L_\mu(S, w, \gamma, \beta) = \nabla e(S) - \gamma^T \nabla g(S) - \beta^T \nabla h(S) = 0 \tag{5.21}$$

$$\nabla_w L_\mu(S, w, \gamma, \beta) = -\mu W^{-1} \varepsilon - \gamma \varepsilon = 0, \tag{5.22}$$

$$\nabla_\gamma L_\mu(S, w, \gamma, \beta) = g(S) + w = 0, \tag{5.23}$$

$$\nabla_\beta L_\mu(S, w, \gamma, \beta) = h(S) = 0 \tag{5.24}$$

Where W is a diagonal matrix whose diagonal elements are wp, ε is the p-vector of all ones, modify equation (5.22) by multiplying W, resulting as follows:

$$\nabla e(S) + \gamma^T \nabla \mathrm{g(S)} + \beta^T \nabla h(S) = 0, \tag{5.25}$$

$$-\mu\varepsilon - WY\varepsilon = 0, \tag{5.26}$$

$$g(S) + w = 0, \tag{5.27}$$

$$h(S) = 0 \tag{5.28}$$

Taking Y is the diagonal matrix with diagonal elements γ_i. Applies Newton's method in order to solve the system above. Substitute $S = S + \Delta S$, $w = w + \Delta w$, $\gamma = \gamma + \Delta\gamma$, and $\beta = \beta + \Delta\beta$ into the system, we have

$$\begin{bmatrix} H(S,\gamma) & 0 & -\nabla g(S)^T & -\nabla h(S)^T \\ 0 & Y & W & 0 \\ \nabla g(S) & I & 0 & 0 \\ \nabla h(S) & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta S \\ \Delta w \\ \Delta\gamma \\ \Delta\beta \end{bmatrix} = \begin{bmatrix} -\nabla e(S) + \gamma^T g(S) + \beta^T \nabla h(S) \\ -\mu\varepsilon - YW\varepsilon \\ -g(S) - w \\ -h(S) \end{bmatrix}, \tag{5.29}$$

$$H(S,\gamma) = \nabla^2 e(S) - \sum_{p=1}^{2n} \nabla^2 g_p(S)\gamma_p - \sum_{p=1}^{2n} \nabla^2 h_p(S)\beta_p.$$

Once the search direction ΔS, Δw, $\Delta\gamma$, $\Delta\beta$ have been computed, the algorithm loop iteratively, and compute the new estimates to the optimal solution as follows:

$$\begin{aligned} S^{(k+1)} &= S^{(k)} + \alpha \Delta S^k \\ w^{(k+1)} &= w^{(k)} + \alpha \Delta w^k \\ \gamma^{(k+1)} &= \gamma^{(k)} + \alpha \Delta \gamma^k \\ \beta^{(k+1)} &= \beta^{(k)} + \alpha \Delta \beta^k \end{aligned} \tag{5.30}$$

where α is the step length.

6.0 EXPERIMENTAL RESULT AND DISCUSSION

6.1 Workload Prediction based on Linear Prediction

Two kinds of methods are used to test for the prediction of performance evaluation. The first method is a Wireless sensor node system model which was built on Matlab Simulink SimEvent, and its workload state was captured for algorithm testing. In the simulation system model, five components generate tasks at each different rate. The execution time for each task will be uniformly distributed between [BCET, WCET], where BCET is best case execution time, and WCET is worst case execution time, with ratio WCET/BCET of 5. The scheduler policy for the system model is First-In First-Out (FIFO).

Figure 6.1 and 6.2 show the predicted workloads that were done by MMSE and WLSE. Both MMSE and WLSE are compared with proposed LMS algorithm [63], in terms of the corresponding prediction performance. Root mean square error (RMSE) is used for prediction performance metrics, and their results are shown in figure 6.4. As seen in figure, both MMSE and WLSE have better performance compared to LMS with lesser RMSE. MMSE has the least RMSE among other algorithms with 0.0928 when the number of samples, M is 5. WLSE has a 0.1 RMSE when M is 6, which is slighly higher than MMSE. LMS has the highest RMSE among all, which was 0.1172 when M is 3. Note that, if the number of past samples, M is too little, the prediction will be noisy and if it is too much, there exist excessive low pass filtering. Both will results in high RMS error. Thus, choosing a right number of past samples has to be wise in order to get the best performance.

For WLSE algorithm, in order to optimize the prediction performance, choosing a right forgetting factor, α is crucial. Figure 6.3 shows the surface plot of RMSE for WLSE with respects of M and forgetting factor, α. When M is 6 and α is 0.8, the prediction performance is most optimal.

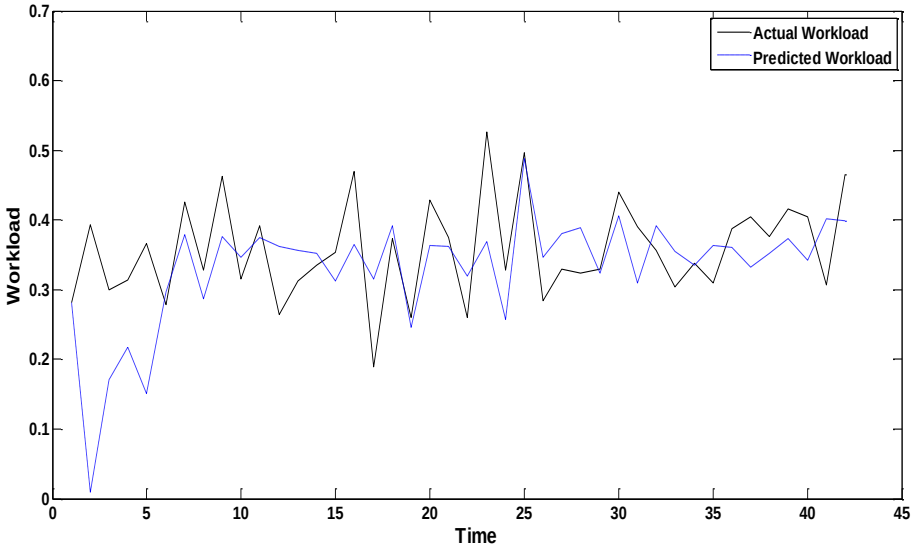

Figure 6.1 MMSE workload prediction (Simulink model generated)

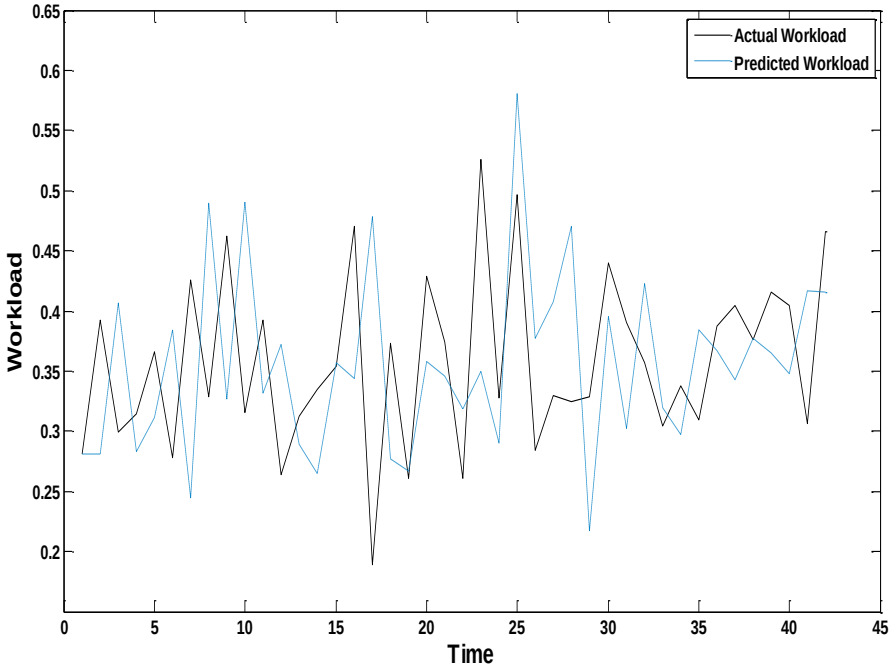

Figure 6.2 WLSE workload prediction (Simulink model generated)

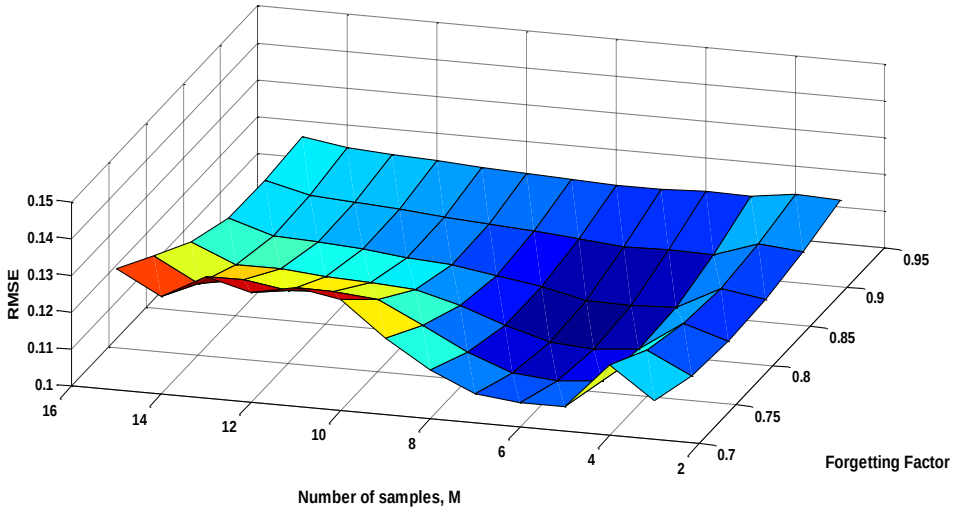

Figure 6.3 Root mean square error of WLSE respects to number of samples and forgetting factor,α

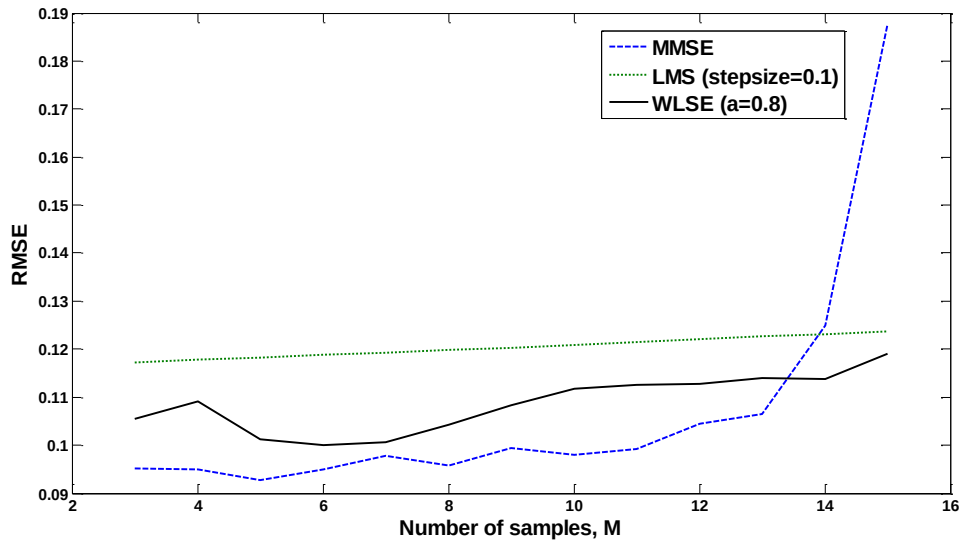

Figure 6.4 Root mean square error comparison between MMSE, WLSE, and LMS

As for the other evaluation method, workload prediction performance is tested on a small network consists of six commercial sensor nodes, namely Tinynode [72]. Tinynode operates at 8MHz, and it has a MSP430 Texas Instruments microcontroller, a 512kb External Flash, Semtech XE1205 ultra-low power multi-channel wireless transceiver and temperature sensor on board. We choose TinyOS—a component based

embedded operating system, to operate on the sensor node. TinyOS employs FIFO schedule policy and event driven execution model. A simple application is deployed on the sensor nodes, where temperature sensing data of every node was acquired, and then relayed back to the sink node by using a multi-hop network protocol. The sink node's workload was captured and we use MATLAB to do the prediction algorithms testing.

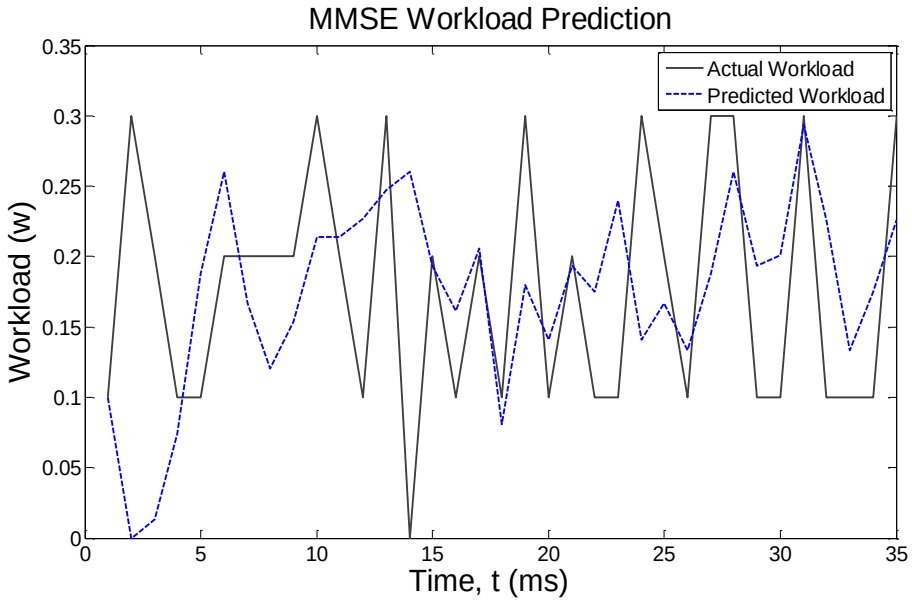

Figure 6.5 MMSE workload prediction (WSN generated workload)

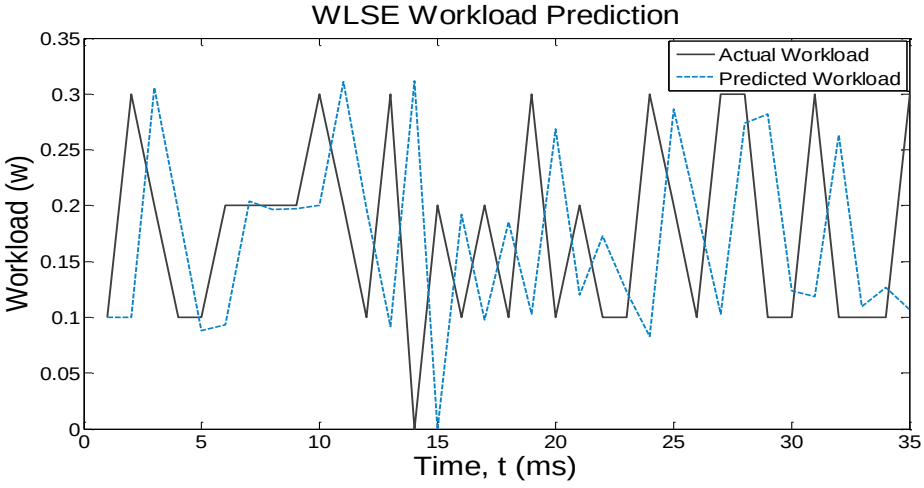

Figure 6.6 WLSE workload prediction (WSN generated workload)

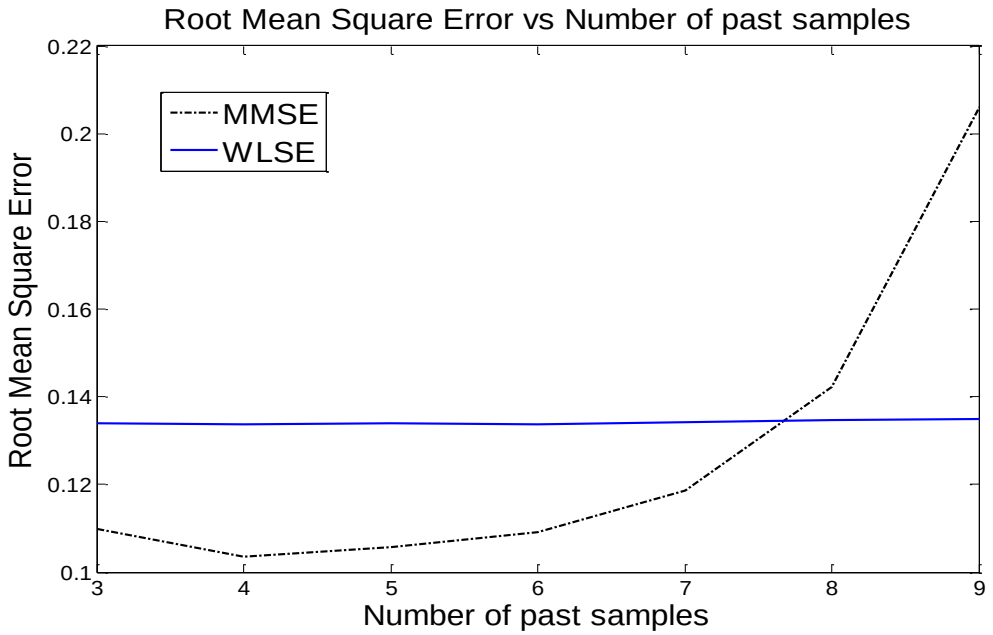

Figure 6.7 Root means Square Error comparison for both MMSE and WLSE

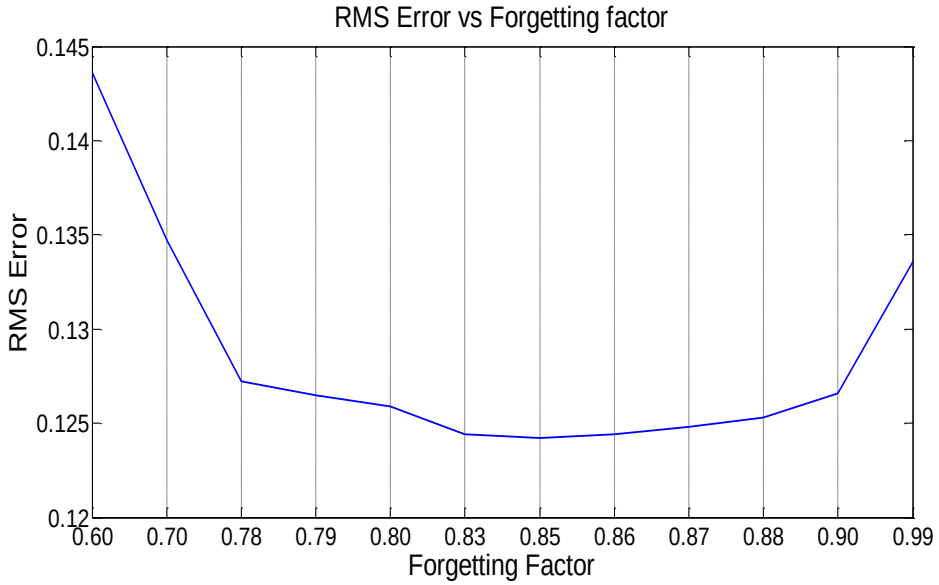

Figure 6.8 Root mean square error of WLSE respects forgetting factor, α

Figure 6.5 and Figure 6.6 picture both MMSE (M = 4) and WLSE (M = 6, α = 0.99) based future workload prediction. The performance for both of the algorithms are judged in terms of RMSE as well, which is shown in Figure 6.7. MMSE produced a

better prediction compared to WLSE, when M = 4, it generates the least RMS error, which is 0.1035.

In figure 6.8, it shows the RMS error of WLSE vs. forgetting factor, α with M = 6. Same as previous, choosing too large or too little α will result in bad predicting performance. As seen in figure, when α = 0.85, it gets smaller RMS error, which is 0.1242.

For prediction wise, MMSE has better performance compared to WLSE. But in computational wise, WLSE uses much lesser computation compared to MMSE. Notice that, in MATLAB environment, MMSE algorithm used 55ms to complete the running while WLSE used only 1.2ms. It means that MMSE needs almost 46 times more computation compared to WLSE, and this is a huge advantage for WLSE. It is obvious why there exist a huge computation difference because MMSE needs to compute the autocorrelation function while WLSE does not need so. Hence, in resource constrained environment for WSN, it is more practical to choose WLSE rather than the other one.

6.2 Convex Optimization of DVS

Simulation experiments were performed by using Simulink SimEvent to evaluate convex optimization DVS. Several task sets are considered, each with 10 randomly generated tasks. Tasks were assigned with a random period within range of [2, 40], and WCET within a period of [0.01, 3].

Power model which is shown in Chapter 2, equation (2.1) is used for the corresponding experiments. Threshold voltage is 0.8V and maximum voltage is 5V. The speed has been normalized in the range of [0.1, 1] at the steps of 0.001.

For interior point method search stopping criteria, the condition is set when maximum iteration is 10000, or when the solution is stopped to improve with the improved range of $1X10^{-6}$.

The Energy consumption for both conventional method, Static Voltage Scaling [49] with convex optimization of DVS is compared. Both methods are compared under scenario of identical power characteristic and varying task power characteristic.

Identical Power Characteristic

For the simulation result of identical power characteristic shown in figure 6.9, both static voltage and proposed DVS method generate the same optimal solution. Hence, theorem shown in [74], in which it stated that the optimal speed allocation for each tasks for minimizing energy consumption should be constant and equal to total utilization, U_{tot} is proven.

Varying Power Characteristic

Tasks can have different power characteristics due to the diverse nature of the tasks in a system [43]. For the experimental results, we assumed the power characteristics to be linear. It means that the tasks have constant power coefficient, k. The equation is shown below:

$$P_{cmos} = kC_L N_{SW} V_{DD}^2 f \tag{6.1}$$

The power function coefficients of the tasks are uniformly distributed between [1, 10].

Figure 6.10 shows the results of both static voltage and proposed DVS. It clearly shows that Convex Optimization of DVS has better normalized energy saving compared to static voltage scaling except when the utilization equal to 0.1. Both have the same energy saving due to the bounded minimize speed. The energy saving decreases when utilization began to increase, because the speed allocation becomes higher, which contribute to more energy consumptions.

Convex Optimization inco-operate with Generic Dynamic Reclaiming Algorithm

Generic Dynamic Reclaiming Algorithm (GDRA) is an online DVS scheme that aims to further scale down the speed when a task finishes earlier than WCET [50]. The idea behind GDRA is to allocate the runtime slack introduced by an early finished task to a lower priority task. GDRA is proved to be deadline guarantee.

Convex Optimized DVS as offline DVS is combined with GDRA as online DVS for performance evaluation. We vary the execution time of a task in normal distributed manners within a range of [r, WCET]. Here, r is the ratio of WCET and best case

execution time (BCET), which is equal to WCET/BCET. The power characteristic coefficient in all the WCET/BCET ratio cases is equal to 10. Different of WCET/BCET ratio of experiment is performed, and Figure 6.11 shows the outcome. As seen earlier, we compare both static DVS with convex Optimized DVS. Notice that from the graph, Convex Optimized DVS outperform static DVS. The normalized energy saving increases when WCET/BCET ratio starts to increase. This is because higher WCET/BCET ratio implies more slack time introduction, hence higher potential of higher speed reduction.

Computational Performance

The computational performance of both inequality and equality constrained problem formulation were investigated. Figure 6.12 shows the number of iteration of computation for both equality constrained and inequality constrained problem. It shows that equality constrained has far lesser iteration compared to inequality constrained when the utilization increases. Equality constrained problem has lesser iteration than inequality constrained because of the smaller feasible search space.

Figure 6.13 shows the graph of number of task set versus the number of iteration of equality constrained problem. The iteration increases when the number of task set increases due to the increase of variable search space to compute as number of task set increases.

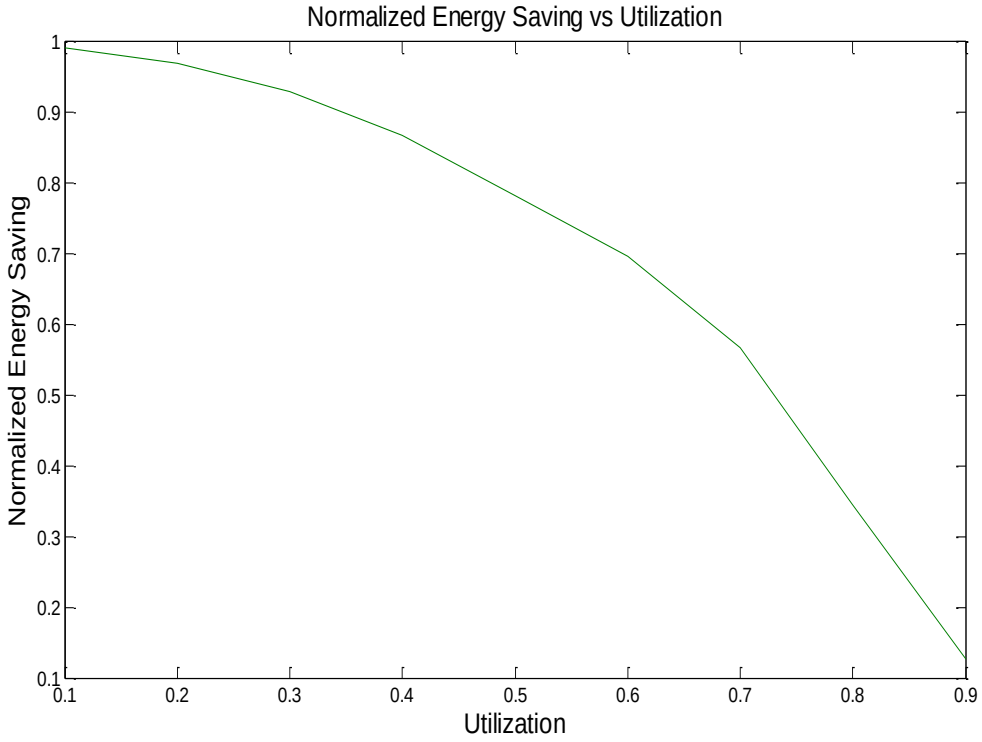

Figure 6.9 Normalized Energy Saving for both Equality and Inequality constrained problem optimization (identical power characteristic)

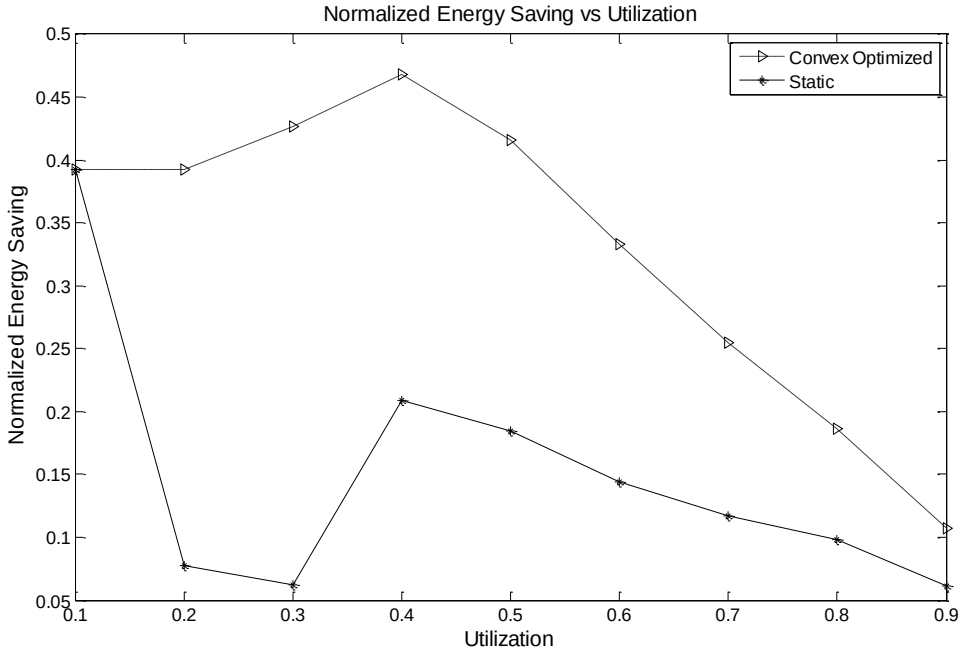

Figure 6.10 Normalized Energy Saving for task varying power characteristic

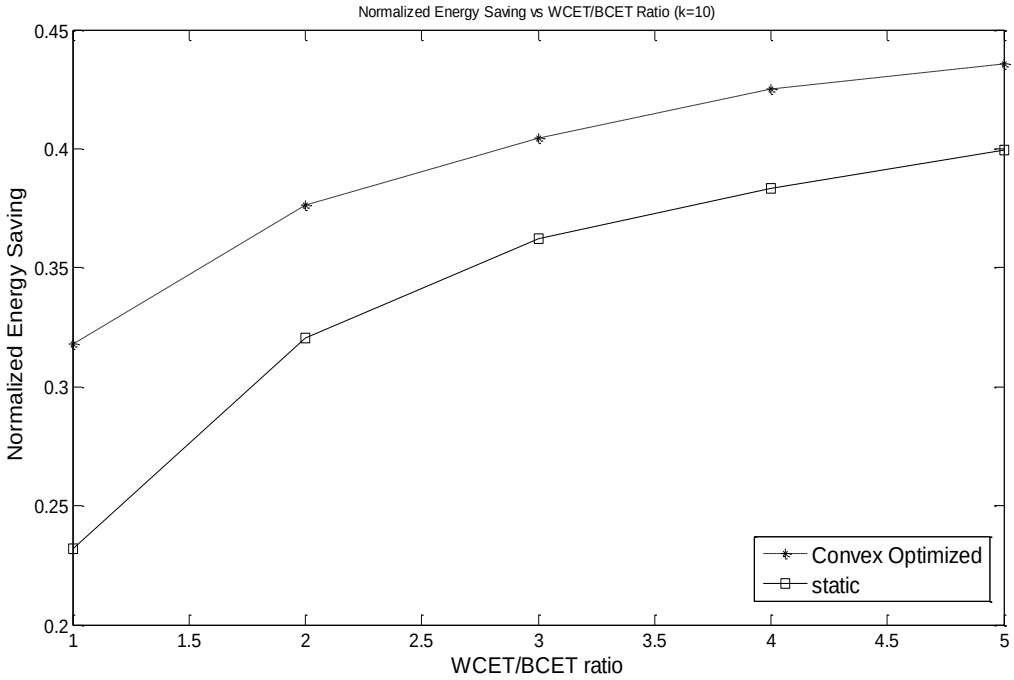

Figure 6.11 Normalized Energy Saving respects with WCET/BCET ratio in online condition

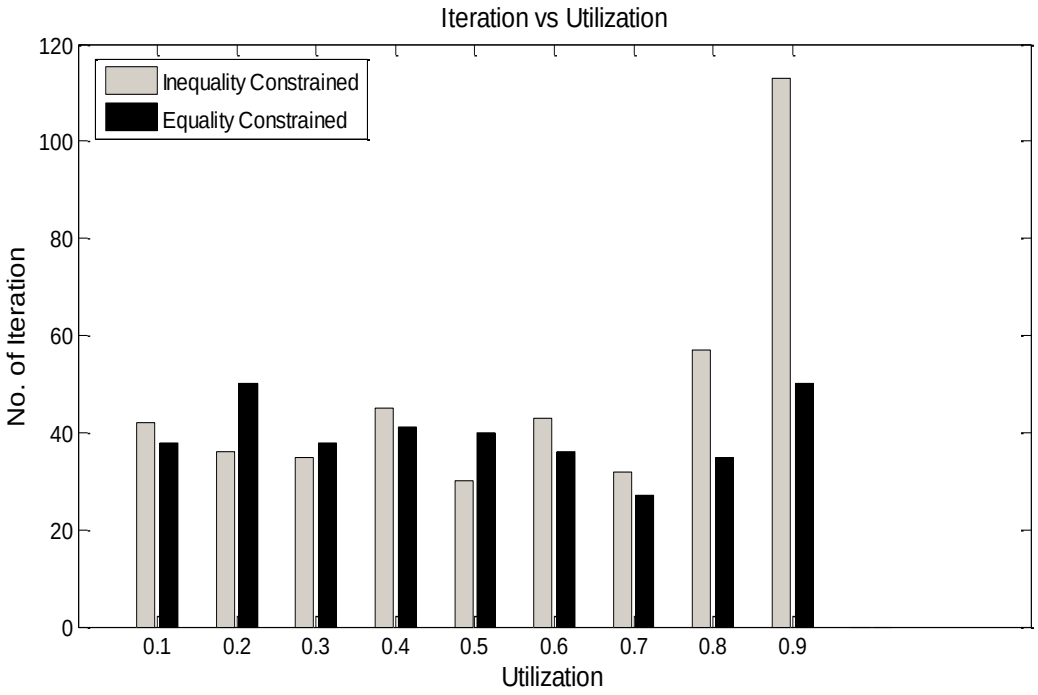

Figure 6.12 Number of iteration comparison between Equality and Inequality constrained problem optimization

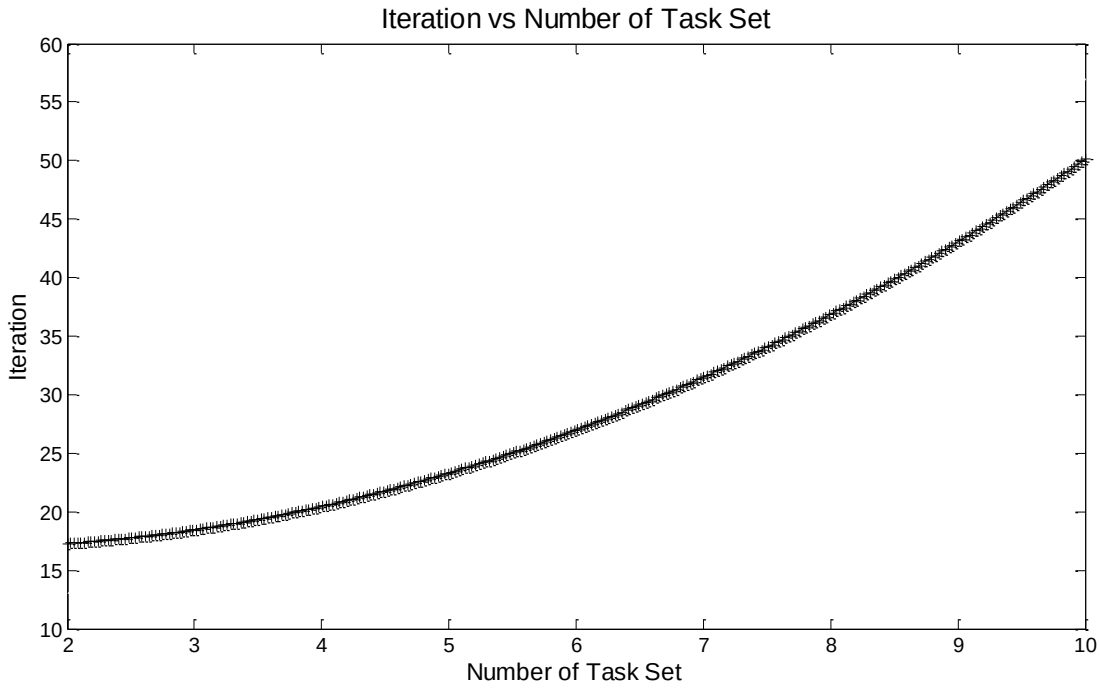

Figure 6.13 Number of iteration respects to number of task set

7.0 CONCLUSION

In this dissertation, the workload prediction based DVS for soft has been studied on WSN. It was found that other than conventional DPM method, DVS can be an effective technique to prolong the operational life of WSN. This dissertation proposed both online and offline DVS schemes to change the frequency and voltage. The workload prediction algorithm proposed is suitable for soft real-time system, since it does not consider on the total utilization of the system.

There are two types of algorithms applied in a predictor: MMSE and WLSE. MMSE generates a better prediction performance while WLSE generates a lesser computation. Hence, it is up to the users to decide whether MMSE with good prediction performance but large computation will be suitable for their application. Furthermore, for a better prediction result, choose three to seven past samples, M, would be a good choice.

As for hard real time system, an offline DVS is proposed. The equality constrained problem formulation is presented. The comparison between equality constrained problem and conventional inequality constrained problem has been made. The equality constrained problem has been proved to have better computational performance, compared with inequality constrained problem.

8.0 FUTURE WORK

Several extensions for the future work, a power aware task scheduling policy for overload system could be investigated, in order to perform a trade-off between minimizing numbers of deadline miss while maximizing the power saving performance.

Other than that, a speed scheduling for real-time system varying power characteristic tasks could be investigated in the future. As some system have different tasks with specific power characteristics, a power scheduling scheme for such case will be proposed in order to enhance power saving performance of the system.

REFERENCE

[1] J. W. S. Liu, *Real-Time systems*. Upper Saddle River, NJ: Prentice Hall, 2000.

[2] P. S. Sausen, J. R. B. Sousa, M. A. Spohn, A. Perkusich, and A. M. N. Lima, "Dynamic power management with scheduled switching modes in wireless sensor networks," in 15th IEEE MASCOTS, Istanbul, Turkey, pp. 1–8, Oct. 2007.

[3] Y. Wu, X. Li, Y. Liu, W. Lou, "Energy-Efficient Wake-Up Scheduling for Data Collection and Aggregation", *IEEE Transactions On Parallel And Distributed Systems*, Vol. 21, pp. 275-287 No. 2, Feb.2010.

[4] F. Salvadori, M. de Campos, Sausen, R. F. de Camargo, C. Gehrke, C. Rech, M.A. Spohn, A.C., Oliveira, "Monitoring in Industrial Systems Using Wireless Sensor Network With Dynamic Power Management", *IEEE Transactions on Instrumentation and Measurement*, vol. 58, pp. 3104 – 3111, no. 9, Sept. 2009.

[5] G. Anastasi, M. Conti, M. D. Francesco, "Extending the Lifetime of Wireless Sensor Networks Through Adaptive Sleep", *IEEE Transactions on Industrial Informatics*, Vol. 5, pp.351-365, NO. 3, Aug.2009

[6] G. W. Taylor, J. R. Burns, S. a. Kammann, W. B. Powers, and T. R. Welsh, "The Energy Harvesting Eel: a small subsurface ocean/river power generator," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 4, pp. 539–547, 2001.

[7] G. Li, "Research on a self-powered wireless ultrasonic flow sensor system," *2010 IEEE International Conference on Progress in Informatics and Computing*, pp. 522–526, Dec. 2010.

[8] V. Raghunathan, a. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pp. 457–462, 2005.

[9] M. A. Weimer, T. S. Paing, and R. A. Zane, "Remote area wind energy harvesting for low-power autonomous sensors," in *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*, 2006, pp. 1-5.

[10] K. Vijayaraghavan and R. Rajamani, "Novel Batteryless Wireless Sensor for Traffic-Flow Measurement," *Vehicular Technology, IEEE Transactions on*, vol. 59, pp. 3249-3260, 2010.

[11] J. Ahola, T. Ahonen, V. Sarkimaki, A. Kosonen, J. Tamminen, R. Tiainen, and T. Lindh, "Design considerations for current transformer based energy harvesting for electronics attached to electric motor," in *Power Electronics, Electrical Drives, Automation and Motion, 2008. SPEEDAM 2008. International Symposium on*, 2008, pp. 901-905.

[12] C. Cheng, C. K. Tse, F.C. Lau, "A Delay-Aware Data Collection Network Structure for Wireless Sensor Networks", *IEEE Sensors Journal*, Vol. 11, pp. 699-710, No. 3, Mar. 2011.

[13] S. Sharma, S. K. Jena, "A Survey on Secure Hierarchical Routing Protocols in Wireless Sensor Networks", *Proceedings of the 2011 International Conference on Communication, Computing & Security*, Feb. 2011.

[14] H.-S. Yun and J. Kim, "On energy-optimal voltage scheduling for fixed-priority hard real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, pp. 393–430, Aug. 2003.

[15] B. R. Swim, M. Benmaiza, M. Tayli, and M. C. Woodward, "Predictable distributed dynamic scheduling in RTDOS," *IEE Proceedings - Computers and Digital Techniques*, vol. 144, no. 3, p. 195, 1997.

[16] S. Insop, K. Sehjeong, and F. Karray, "A Real-Time Scheduler Design for a Class of Embedded Systems," *Mechatronics, IEEE/ASME Transactions on*, vol. 13, pp. 36-45, 2008.

[17] Z. Fengxiang and A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," *Computers, IEEE Transactions on*, vol. 58, pp. 1250-1258, 2009.

[18] a Shah and K. Kotecha, "Scheduling Algorithm for Real-Time Operating Systems Using ACO," *2010 International Conference on Computational Intelligence and Communication Networks*, pp. 617–621, Nov. 2010.

[19] K. Ramamritham and J. a. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 5, pp. 564–577, May 1987.

[20] S. Lorpunmanee, M. N. Sap, A. H. Abdullah, and C. Chompoo-inwai, "An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment," *International Journal of Computer and Information Engineering*, pp. 314–321, 2007.

[21] C. W. Chiang, Y. C. Lee, C. N. Lee, and T. Y. Chou, "Ant colony optimisation for task matching and scheduling," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, pp. 373-380, 2006.

[22] J. W.-S. Liu, "Algorithms for scheduling real-time tasks with input error and end-to-end deadlines," *IEEE Transactions on Software Engineering*, vol. 23, no. 2, pp. 93–106, 1997.

[23] P. Visalakshi, S. N. Sivanadam, "Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization," *Int J. Open Problems, Compt. Math*, vol. 2, no. 3, September, 2009.

[24] J. Eker and a. Cervin, "A Matlab toolbox for real-time and control systems co-design," *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*, pp. 320–327.

[25] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment," *JACM*, vol. 20, no. 1, pp. 46-61, 1973.

[26] C. Steiger, H. Walder, and M. Platzner, “Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks,” *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, Nov. 2004.

[27] a. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power CMOS digital design,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[28] R. Ravishankar and V. Sarma, "Energy-Optimal Speed Control of a Generic Device," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 25, pp. 2737-2746, 2006.

[29] V. Raghunathan, P. Spanos, and M. B. Srivastava, “Adaptive power-fidelity in energy-aware wireless embedded systems,” *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420),* pp. 106–115, 2001.

[30] R. Jejurikar and R. Gupta, "Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems," in *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, 2004, pp. 78-81.

[31] Y. Lin and Q. Gang, "Analysis of energy reduction on dynamic voltage scaling-enabled systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 24, pp. 1827-1837, 2005.

[32] T. a. AlEnawy and H. Aydin, “Energy-Constrained Scheduling for Weakly-Hard Real-Time Systems,” *26th IEEE International Real-Time Systems Symposium (RTSS’05)*, pp. 376–385, 2005.

[33] A. Shah and K. Kotecha, "Adaptive Scheduling Algorithm for Real-Time Multiprocessor Systems," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, 2009, pp. 35-39.

[34] K. Taewhan, "Application-Driven Low-Power Techniques Using Dynamic Voltage Scaling," in *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, 2006, pp. 199-206.

[35] W. Kim and S. L. Min, "Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pp. 788–794, 2002.

[36] V. Gutnik and a. P. Chandrakasan, "Embedded power supply for low-power DSP," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 425–435, Dec. 1997.

[37] J. Luo, N. K. Jha, and L.-S. Peh, "Simultaneous Dynamic Voltage Scaling of Processors and Communication Links in Real-Time Distributed Embedded Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 427–437, Apr. 2007.

[38] K. Kotecha and A. Shah, "Adaptive scheduling algorithm for real-time operating system," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 2109–2112, Jun. 2008.

[39] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic I/O device scheduling in hard real-time systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, pp. 847-858, 2003.

[40] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," *Proceedings of the 38th conference on Design automation - DAC '01*, pp. 828–833, 2001.

[41] L. B. Hormann, P. M. Glatz, C. Steger, and R. Weiss, “Evaluation of component-aware dynamic voltage scaling for mobile devices and wireless sensor networks,” *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1–9, Jun. 2011.

[42] F. Yao, a. Demers, and S. Shenker, “A scheduling model for reduced CPU energy,” *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 374–382, 1995.

[43] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Real-Time Systems, 13th Euromicro Conference on, 2001.*, 2001, pp. 225-232.

[44] K. Govil, E. Chan and H. Wasserman, “Comparing Algorithm for Dynamic Speed-Setting of a Low-Power CPU,” in *MobiCom ’95 Proceedings of the 1st annual international conference on Mobile computing and networking*, 1995.

[45] T. Pering and R. Brodersen, “Energy Efficient Voltage Scheduling for Real-Time Systems,” *Proc. Fourth Real-Time Technology and Applications Symp., WIP Session*, 1998.

[46] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processor," in *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, 1998, pp. 653-656.

[47] I. Hong, D. Kirovski, Q. Gang, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, pp. 1702-1714, 1999.

[48] Y. Shin and K. Choi, “Power conscious fixed priority scheduling for hard real-time systems,” *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pp. 134–139.

[49] P. Pillai and K.G. Shin, “Real-Time Dynamic Voltage Scaling for Low Power Embedded Operating Systems,” *Proc. 18th Symp. Operating System Principles*, Oct. 2001.

[50] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, pp. 584-600, 2004.

[51] X. Zhong and C. Z. Xu, "Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Real-Time Guarantee," *Computers, IEEE Transactions on*, vol. 56, pp. 358-372, 2007.

[52] F. Gruian, “Hard real-time scheduling for low-energy using stochastic data and DVS processors,” *Proceedings of the 2001 international symposium on Low power electronics and design - ISLPED ’01*, pp. 46–51, 2001.

[53] R. Jejurikar and R. Gupta, "Optimized Slowdown in Real-Time Task Systems," *Computers, IEEE Transactions on*, vol. 55, pp. 1588-1598, 2006.

[54] L. Yuan, “ALT-DVS: Dynamic Voltage Scaling with Awareness of Leakage and Temperature for Real-Time Systems,” *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pp. 660–670, Aug. 2007.

[55] J. Choi and H. Cha, "Memory-aware dynamic voltage scaling for multimedia applications," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, pp. 130-136, 2006.

[56] D. Shin and J. Kim, “Dynamic voltage scaling of mixed task sets in priority-driven systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 438–453, Mar. 2006.

[57] J. R. Lorch and a. J. Smith, “PACE: a new approach to dynamic voltage scaling,” *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 856–869, Jul. 2004.

[58] B. Sung-Yong, B. Kwanhu, Y. Sungroh, and C. Eui-Young, "Run-Time Adaptive Workload Estimation for Dynamic Voltage Scaling," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 1334-1347, 2009.

[59] D.-R. Chen and C.-C. Hsu, “Transition-Aware Dynamic Voltage Scaling for Jitter-Controlled Real-Time Scheduling: A Tree-Structured Approach,” *2009 International Conference on Parallel Processing Workshops*, pp. 27–34, Sep. 2009.

[60] S. Dongkun and K. Jihong, "Intra-task voltage scheduling on DVS-enabled hard real-time systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, pp. 1530-1549, 2005.

[61] T. Zitterell and C. Scholl, “Improving energy-efficient real-time scheduling by exploiting code instrumentation,” *2008 International Multiconference on Computer Science and Information Technology*, pp. 763–771, Oct. 2008.

[62] X. He, Y. Jia, and H. Wa, “Stochastic Voltage Scheduling of Fixed-Priority tasks with Preemption Thresholds,” *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–5, Oct. 2008.

[63] A. Sinha and A. P. Chandrakasan, "Dynamic voltage scheduling using adaptive filtering of workload traces," in *VLSI Design, 2001. Fourteenth International Conference on*, 2001, pp. 221-226.

[64] C. M. Krishna and Y. H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," *Computers, IEEE Transactions on,* vol. 52, pp. 1586-1593, 2003.

[65] L. M. Oliveira and J. J. Rodrigues, "Wireless Sensor Networks: a Survey on Environmental Monitoring", Journal of Communications, Vol. 6, No. 2, pp. 143-151, 2011

[66] B. Porat, Digital Processing of Random Signals: Theory and Methods, Prentice Hall, 2008, pp. 14.

[67] S. Haykin, Adaptive Filter Theory, Prentice Hall, New Jersey, 1986.

[68] I. Griva, G. N. Stephen and A. Sofer, Linear and Nonlinear Optimization, Second Edition. Society for Industrial Mathematics; 2 edition, December 3, 2008.

[69] D. P. Bertsekas, Convex Optimization Theory. Athena Scientific, 1st edition, June 30, 2009.

[70] Boyd, Stephen and L. Vandenberghe. Convex Optimization.Cambridge University Press, 2004.

[71] J. M. Borwein and A. S. Lewis, *Convex Analysis and Nonlinear Optimization, Theory and Examples*. New York: Springer-Verlag, 2000, Canadian Mathematical Society Books in Mathematics.

[72] H. Dubois-Ferrieri, R. Meier, L. Fabre, P. Metrailler, "TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications", Proceeding of The Fifth International Conference on Information Processing in Sensor Networks, pp. 358-365, Jul. 2006.

[73] TinyOS, http://www.tinyos.net.

[74] H. Aydin, "Enhancing Performance and Fault Tolerance in Reward-Based Scheduling," PhD dissertation, Univ. of Pittsburgh, Aug. 2001, http://www.cs.gmu.edu/~aydin/dissertation.

[75] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *Proc. 19th IEEE Real-Time Systems Symp.(RTSS '98)*, Dec. 1998.

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.