UNIVERSITY OF CALIFORNIA,
IRVINE

Harvesting-aware and Quality-aware Energy Management for Solar-Powered Embedded Systems

THESIS

submitted in partial satisfaction of the requirements
for the degree of

Doctor of Philosophy

in Computer Science

by

Nga Dang

2015

DEDICATION

To

my mother and my husband

in recognition of their worth,
and their unconditional love and enormous support during my PhD

If, just once during his lifetime,
he could make one other person happy,
then he,
too,
would be happy.
- Yasunari Kawabata

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to thank those who have inspired and supported me during my PhD. First and foremost, I would like to thank my advisor, Prof. Eli Bozorgzadeh, who has constantly supported, guided me, and gave me countless valuable suggestions. I would also like to thank my committee, Prof. Nalini Venkatasubramanian for her inspiration and insights on wireless sensor networks, Prof. Moonju Park for teaching me real-time system principles and Prof. Marco Levorato for introducing me to Markov Chain.

I wish to thank my parents and my family, especially my mother and my husband, Rosario Cammarota for their precious love, constant support, and care which give me strength every day. But more than anything, their working ethic, passion, and dedication inspired my journey.

I wish to thanks my best friend at UC Irvine, Siripen Pongpaichet, her humility reminded me to work harder and her care was important during my last 2 years of PhD. I wish to thanks all colleagues that have co-authored some of the work presented in this dissertation. And thanks to many friends I met at Irvine who brought joy to my life.

Through this PhD journey, I have learnt many lessons. I feel thankful for this opportunity to understand myself, my strengths and my weaknesses, and to become a better person.

ABSTRACT OF THE THESIS

Harvesting-aware and Quality-aware Energy Management
in Solar-Powered Embedded Systems

By

Nga Dang

Ph.D in Computer Science

University of California, Irvine, 2015

Associate Professor Eli Bozorgzadeh, Chair

Embedded system market growth is fast in the last couple of decades, especially with the current demand for embedded processing to support IoT services. However, to power billions of IoT devices and embedded systems is a challenge. Energy harvesting has emerged as a promising power supply alternative for embedded systems, enabling systems to convert renewable energy sources in the surrounding environment (e.g., solar, wind, thermal, kinetic energy) to electrical energy. Nevertheless, these renewable energy sources often exhibit temporal and spatial variations, which cause uncertainties and fluctuations in the energy supply of the systems. Furthermore, the complex non-ideal characteristics of harvesting circuit components such as converters and the energy storage make it even more challenging for energy management. In this thesis, I propose a harvesting-aware and quality-aware energy management middleware framework for solar-powered embedded systems, modeling them as complete Cyber Physical Systems, optimizing performance (Quality of Service), and tackling their mentioned challenges.

The energy management middleware framework exploits energy harvesting history, patterns, and prediction algorithms to extrapolate solar harvesting in the next harvesting period. The prediction is used by an offline algorithm to assign QoS for each time slot in the next harvesting period. The objective is to maximize total QoS while meeting energy harvesting constraints. An online adaptation phase is employed to adjust QoS assignment if actual harvesting profile deviates from offline prediction. This thesis explores the notion of application QoS and QoS adaptation in two middlewares for two types of systems: communication-intensive and computation-intensive. In communication-intensive systems, such as wireless sensor networks, where majority of tasks (and energy consumption) are sending and receiving messages, I exploit the tolerance to data inaccuracy in data transmission. In computation-intensive systems such as real-time systems, application QoS can be expressed as quantity and distribution of real-time task completion. In particular, I target firm real-time systems with adaptive QoS.

Furthermore, energy harvesting embedded systems require energy storage to smooth out fluctuations and to store energy for long term operation. Two types of commonly used energy storage elements are batteries and supercapacitors. Frequent charging and discharging activities accelerate the aging of batteries while supercapacitors suffer from high leakage. Therefore in this middleware, in orchestration with QoS adaptation, I propose algorithms to keep battery aging under threshold and reduce leakage from supercapacitors.

Lastly, as an alternative to prediction-based deterministic approach, I explore stochastic modeling using Finite State Markov Chain and propose a novel unified stochastic model and optimization for solar-powered embedded systems. Experimental results show that stochastic model improves the systems' ability to capture and adapt to variations in the energy harvesting supply and application demand.

Chapter 1 Introduction

Embedded Systems growth is tremendous in the last couple of decades. Numbers reported by BCC research showed that billions of revenue was generated from worldwide embedded system sale [1]. Embedded System growth was double in all sectors between 2004 and 2009, from consumer appliances, automotives, medical devices, to industry/military equipments. The emergence of Internet of Things (IoT) gives even a greater boost to embedded systems growth. It is predicted by Cisco that there will be 50 billion of IoT devices connected to the Internet by 2020 [2] and the root of all these IoT services will be embedded systems with sensing, processing, and communicating ability [3].

A basic but important question is how we could power these 50 billion of IoT devices and embedded systems. Sustainability, including energy sustainability is a key challenge for embedded systems. A sustainable embedded system is endurable, providing continuous service or with minimal disruption during its long lifetime. To achieve this goal, embedded systems need to rely on various methods, such as infrastructure resilience, energy sustainability, to application adaptation or all of them. My thesis focuses on energy sustainability aspect of embedded systems. For other aspects of sustainability, such as infrastructure resilience or application adaptation, the readers are referred to [4, 5].

Energy sustainable systems must have a reliable and dependable energy supply subsystem which could be the characteristics of the energy source itself or could be provided through a good energy management layer. The power grid is a dependable energy source as the chance of power outage nowadays is small in many countries. However, the power grid is not a suitable power option for embedded systems. Embedded systems are often portable or embedded into the physical world. An elaborate effort as wiring to a power plug is intrusive and limits the portability or outreach of embedded systems.

Traditionally, to achieve energy sustainability while being non-intrusive, embedded systems rely on non-rechargeable batteries. However, batteries have limited capacity and this poses another challenge for embedded systems: their lifetime is bounded by the capacity of batteries. For example, if a wireless sensor mote runs at full duty cycle, it lasts only several weeks. Wireless sensor motes are therefore usually set to operate at low duty cycle, from 1% to 5% to prolong the system lifetime while maintaining minimum acceptable quality of services. Still their batteries last from 6 months to 1 year. As soon as a wireless sensor mote runs out of battery, a technician must replace its exhausted battery to avoid temporary shutdown in their services or lose access to a physical area under monitoring by this sensor mote. It is costly to replace batteries (think of the labor cost to replace 50 billion of batteries). In scenarios where changing battery is impossible due to inaccessibility or high risk such as volcano or battlefield monitoring, out-of-battery sensor motes must be discarded and new sensor motes are deployed. Despite many efforts to prolong system lifetime and maintain system performance under power-efficient or energy-efficient management schemes [15, 16], the repetitive high maintenance cost still remains a burden for embedded systems.

There is a need for a new alternative energy supply for embedded systems that can alleviate these concerns. A promising power supply for embedded systems is renewable energy. Renewables are energy sources that keep replenishing themselves such as solar, wind, kinetic, and thermal energy. Their obvious advantages are that they are free, unlimited, no noise, odorless, and completely clean. With the advancement of energy harvesting technologies and system development, embedded systems are now able to transform renewable energy in the surrounding environments into electrical power [5, 17, 18]. Energy harvesting embedded systems last perpetually only subject to hardware aging and failures. This emerging technology makes it feasible to build embedded systems that are truly energy sustainable, nonintrusive, with low maintenance cost and long lifetime. There are existing systems integrated with energy harvesting technologies from the macro level down to the micro level, from solar farms, smart grids, solar-powered houses, cars, to solar-powered wireless sensor motes and surveillance cameras. My thesis focuses on micro-scale energy harvesting systems. I define the concept of micro-scale energy harvesting embedded systems as opposed to macro-scale energy harvesting systems (such as solar or wind farms) as follow:

"**Energy Harvesting Embedded Systems are small low power embedded devices which have energy harvesting subsystems capable of harvesting energy from the surrounding environment to fully or partially power the whole system, hence attain almost perpetual system lifetime.**"

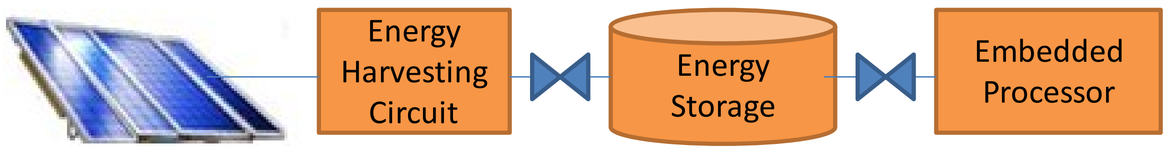


Figure 1.1 Solar-Powered Embedded Systems

Figure 1.1 shows the hardware architecture of a solar-powered embedded system. This thesis focuses on solar harvesting because of its high harvesting potential and accessibility. Each embedded system is equipped with a solar panel (or an array of solar panels) to harvest energy from solar irradiance. The solar panel voltage is controlled by an energy harvesting unit which employs a Maximum Power Point Tracking (MPPT) method to optimize the generated power. Harvested energy is stored in the energy storage. Two typical energy storage elements are batteries and supercapacitors. To isolate solar panel optimal voltage from energy storage voltage, a buck/boost converter is added to the energy harvesting unit. Energy storage supplies power to the embedded processor through another DC-DC converter which matches supply voltage with the embedded processor’s required operating voltage. Embedded processors run applications that provide services or information to users.

1.1 Energy Sustainability Approaches for Smart Spaces

In this section, I present several views and approaches of energy sustainability in a larger scope, beyond embedded system, WSNs, or IoT, i.e., energy sustainability for smart spaces. I show that among these approaches, energy harvesting is still the most suitable approach for sustainable embedded systems. Smart spaces mostly refer to enclosed well-structured areas such as a building, a house, a meeting room whose physical infrastructure is well integrated with computing, communication technologies, and energy systems. Smart spaces are extended to larger scale and more sophisticated (and/or open) spaces such as an instrumented campus composed of several smart buildings, smart parking lots, and smart outdoor spaces (e.g., Responsphere infrastructure on UCI campus [6]).

Approaches for energy sustainability in smart spaces should consider the entire energy system in a holistic way, not only reducing total energy usage, peak load, or enabling Smart Grid architecture integration but also considering any side-effects on occupant behaviors, and possible pollutants. To achieve energy sustainability in smart spaces, there have been works proposed in the following categories:

Energy awareness increase: It is important to make building occupants aware of their energy usage by leveraging sensing systems together with visualization and other forms of media (such as smart phones, apps, social networks) to convey relevant information to users. This can make an impact and influence their behavior towards a more parsimonious usage of utilities including electricity, gas, heating, water, etc.

Smart buildings, smart apps: Novel approaches are needed to predict, monitor, and actuate the systems in smart spaces in order to reduce energy consumption. The SYNERGY Labs at UC San Diego performed multiple research projects on smart buildings [7, 8]. The team employed occupancy sensors that can tell when a room is empty, and had these detectors communicate with a smart controller to adjust the existing HVAC system in real time, reducing their energy waste [9]. The wireless occupancy sensors are claimed to be easy-to-use and do not require any alteration to existing energy systems. These sensors achieve accuracy of 96% and are calibrated to never assume a room is empty when someone is around.

Smart meters: A smart meter is an electrical meter that records consumption of electric energy in houses in intervals of an hour or less and communicates that information at least daily back to a utility plant for monitoring and billing purposes [10]. Such an advanced metering infrastructure (AMI) differs from a traditional automatic meter reading (AMR) in that it enables two-way communications between a meter and a central system. Novel infrastructure and communication standards are needed for collecting data from smart meters and energy-related information in smart spaces. However, there are security and privacy issues related to smart meters which must be addressed [11].

Smart materials: Smart materials are those that can adapt themselves to the environment conditions and user needs. A smart material such as smart glass (also called smart windows or switchable windows for homes, skylights, and transportation vehicles) refers to electrically switchable glass which changes light transmission properties when a voltage is applied to it [12, 13]. Current smart glass technologies include electrochromic devices, suspended particle devices, and liquid crystal display. When activated, the glass changes from transparent to translucent, partially blocking light while maintaining a clear view through the glass. The use of smart glass can save cost for heating, air conditioning, and lighting, and avoid the cost of installing and maintaining motorized light screens, blinds, or curtains. Smart glass however increases installation costs and requires the use of electricity and a control system for dimming and changing transparency. Smart glass is just one example of smart materials for smart spaces; other smart materials such as smart lighting to increase energy efficiency in smart spaces are still in research and at initial production.

Smart Grid: Smart Grid is a digitally enabled electrical grid that gathers, distributes, and acts on information about the behavior of all participants (suppliers and consumers) in order to improve the efficiency, reliability, economics, and sustainability of electricity services. One example is the Irvine Smart Grid Demonstration project [14] whose goals are: 1) Reducing customers' utility bills by shifting usage loads to off-peak hours or using an energy storage to buffer energy at low price. 2) Optimizing the performance of the electric grid, renewable generation and energy storage, and 3) Scalability. They applied various cutting-edge technologies including smart meters, smart appliances, solar panels, electric vehicles, and smart electric distribution circuits. A project concept was successfully deployed around University Hills housing community on the UC Irvine campus.

Renewable energy: An alternative to achieve energy sustainability in smart spaces is energy harvesting. Energy harvesting technologies help systems to capture and transform renewable energy in the surrounding environment (such as solar irradiance, wind, kinetic, or thermal energy) to electrical energy to power systems themselves. One characteristic which makes energy harvesting approach unique and different from other sustainability approaches is that energy harvesting does not attempt to reduce the energy consumption of the system since it does not withdraw energy from the grid. It instead relies on the environment to sustain. Furthermore, it can be integrated into Smart Grids and become a cooperative part in these emerging systems. There should be innovative tools to model and visualize energy expenditure, production, and utilization of renewable energy.

Many above approaches address sustainability of various components in smart spaces but only benefit embedded systems indirectly. For example, smart glasses could be integrated with energy harvesting technologies to enhance harvested power for embedded systems. Smart Grid could help to reduce energy concern of embedded systems but explicit wiring from its infrastructure to embedded systems would defeat their non-intrusiveness goal. Among sustainability approaches for smart spaces presented above, renewable energy and energy harvesting techniques have a significant direct impact on energy sustainability of embedded systems. Energy harvesting embedded systems augment traditional sensing, processing, communicating systems with energy harvesting capability to reduce energy concern and prolong their lifetime. However, beside many benefits, energy harvesting embedded systems have their own challenges that will be summarized in the next section. Energy harvesting technologies revolutionize traditional embedded systems but their inherent challenges demand novel solutions at all levels of systems, from hardware to software in order to realize their promising energy sustainability.

1.2 Energy Harvesting as a Promising but Challenging Solution for Embedded Systems

The benefits that energy harvesting technologies bring to embedded systems include but not limited to:

Energy sustainability: Energy harvesting embedded systems operate autonomously on their own renewable energy sources which potentially sustain the system for unlimited time subject only to hardware aging and failures.

Environmental friendliness: Energy harvesting utilizes clean energy sources from the surrounding environments instead of non-rechargeable batteries which must be discarded after use and might not be recyclable. In inaccessible locations, human cannot replace batteries and are forced to discard the whole embedded systems while deploying new ones, resulting in a high amount of unrecyclable system wastes. On the other hand, renewable energy sources are clean, odorless, and noise-free (or low noise).

Energy scalability: As the energy harvesting embedded systems in a smart space grows in size, energy resource is scalable if each embedded system has access to renewable energy sources in the environment. Each system's renewable energy source is largely independent from the others (such as solar irradiance) although they could share the same environment. Because of this independency, energy resource is scalable.

Low generation and maintenance cost: Users have less concern on installation, energy generation, and maintenance of energy harvesting embedded systems. No wire or alteration to the current energy infrastructure of smart spaces is needed to support energy harvesting. Except for the installation cost, the generation of energy in energy harvesting systems virtually costs nothing. Cost in maintenance is much lower compared to traditional battery-powered systems.

Pervasiveness: Because of all the characteristics above, energy harvesting embedded systems can be easily deployed in smart spaces as plug-and-play devices. They can spread and integrate into everyday life's activities effortlessly and non-intrusively.

On one hand, energy harvesting technologies bring many benefits to embedded systems including energy sustainability. On the other hand, energy harvesting embedded systems face several challenges listed below and they require smart management framework in order to attain the aforementioned benefits. Some of the challenges are due to the inherent nature of renewable energy sources; some are general challenges in embedded systems that are exaggerated by the dynamic energy sources. I provide a brief overview of these challenges below:

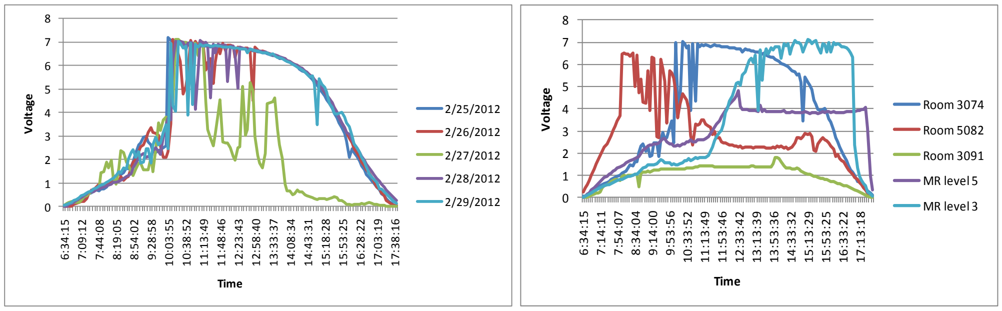


Figure 1.2 a) Temporal Variations and b) Spatial Variations in Solar Energy Profile

Temporal variations: Renewable energy exhibits both spatial and temporal variations. For example, harvesting energy through solar cells highly depends on time of the day, season, and weather conditions that affect exposure to direct sunlight at a specific location. Figure 1.2 shows our measurement in a building on campus at UC Irvine. Figure 1.2a shows the solar energy profiles at the same location for several days in a week. It shows large variations in the energy profiles within each day and of adjacent days.

Spatial variations: Spatial variations refer to the difference in energy harvesting profiles of different locations, even ones in close proximity to each other. For solar irradiance, this is due to geological differences (latitude, longitude) and solar panel tilts. In addition, static objects such as trees and buildings cast different shadow effects on solar panels at different locations. Figure 1.2b shows the harvested energy profiles at several locations around a building on a same day. Solar panels facing east have peak harvesting in the morning while solar panels facing west have peak harvesting in the afternoon. Solar panels that do not have direct sunlight the whole day receive indirect sunlight with lower energy potential. Sudden increases/drops in the profiles in Figure 1.2b was due to objects blocking direct sunlight such as walls or trees. In conclusion, variations in scavenging energy from environment, leading to an uncertainty in energy availability during system operation, challenge the sustainability in embedded systems.

Non-ideal behavior of harvesting circuit components: In Figure 1.1, we see that there are two converters, one matching the voltage of solar panel and energy storage and another matching the voltage of energy storage and the embedded processor. These converters have varying efficiency from *10-40%* depending on the input voltage, the output voltage, and the output current. This significant loss from voltage converters must be considered in energy management. It adds complexity as estimating converter loss accurately requires tracking voltages and currents closely. During runtime, this tracking is possible but expensive. During offline planning, it is difficult to predict solar panel's output voltage and current, energy storage voltage, and load voltage and current as they all vary dynamically in an energy harvesting embedded system.

Energy storage management: Two commonly used elements for energy storage are batteries and supercapacitors. Frequency charging/discharging activities in energy harvesting embedded systems have adverse effects on the energy storage. Such frequent activities of different magnitudes accelerate battery aging and shorten system lifetime. It may also increase supercapacitor leakage and reduce system energy efficiency.

These characteristics make it challenging to realize all the benefits of an energy harvesting embedded system. Therefore, to achieve dependable energy source, an effective energy management is required to tackle mentioned challenges. Furthermore, there are general challenges for embedded systems that must be taken into consideration by the energy management.

1.3 Application Quality of Services and Variations

Embedded processor runs applications that provide services or information to users. Their performance is hence often measured by the Quality of Services. In smart spaces, unsupervised events and environmental changes trigger various applications with sensing, processing, or communicating tasks to run on the embedded systems. Different applications and different demands of the same application in varying scenarios lead to variations in quality of services requirement and urgency of response. While aiming for the highest application quality and fastest response can resolve the issue, it incurs high energy cost, over design and complexity in implementation. Thus, this is often not feasible in practice.

Application demand and QoS variations: The variations in the application demand appear both in time and space domains. For example, the frequency of events such as people entering the main entrance of a building changes over time (day vs. night). Frequency of events also changes across locations, e.g., the entrance of a building as opposed to a back-door or side-door. Surveillance cameras in each of these locations will have variations in their sensing, processing, or communicating demands according to event occurrences.

As the application demand and Quality of Services vary, the energy demand also varies. This is a general challenge for embedded systems. However, in an energy harvesting embedded system whose energy supply also varies, the problem becomes even more challenging.

1.4 Focus of the Dissertation

This dissertation specifically targets harvesting-aware and quality-aware energy management in energy harvesting embedded systems, in particular, solar-powered embedded systems. Solar-powered embedded systems have high accessibility to solar irradiance almost everywhere outdoor and even indoor (such as light bulb or through windows). Compared to other renewable energy sources, solar irradiance also generates significant higher amount of energy (see Table 2.1).

I propose an energy management middleware framework called SQUARES (Smart Quality-aware Renewable Energy Systems) as shown in Figure 1.3. The objective of the framework is to optimize the system performance, measured by application QoS under energy harvesting and system constraints. The framework is aware of challenges in energy harvesting (section 1.2) as well as application demand and QoS (section 1.3).

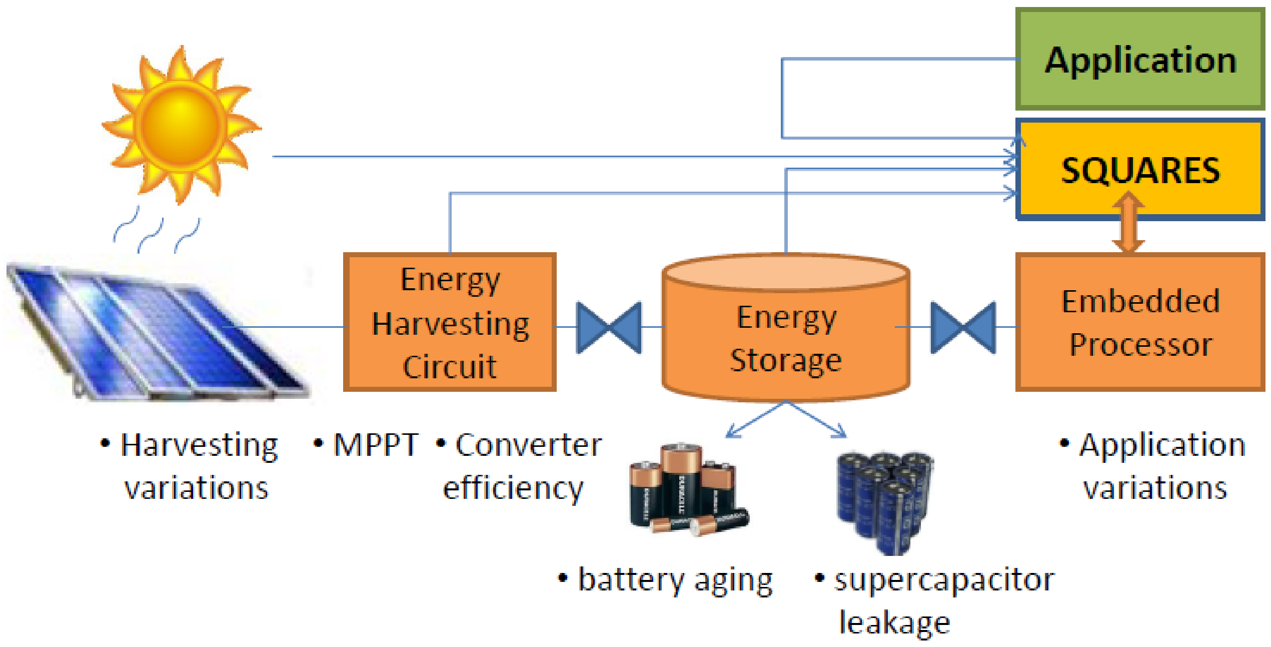


Figure 1.3 SQUARES as a Middleware

SQUARES middleware framework is a software component sitting on top of the embedded processor. Its task is to collect information about each embedded system component including harvesting history, converter efficiency model, energy storage model, embedded processor, and applications to build a unified model for solar-powered embedded systems. SQUARES utilizes this unified model in its algorithms to make control decision for the systems with the objective to maximize application QoS.

In this dissertation, I developed four middlewares based on SQUARES for energy management in solar-powered embedded systems. These middlewares are summarized as follow:

1. **Energy management for Communication-Intensive systems**: This middleware targets communication intensive systems such as wireless sensor networks and data collection application running on such networks. As a preliminary version, this middleware assumes a simple energy storage model with maximum capacity and does not consider converter efficiency. It exploits prediction algorithms and deterministic planning to tackle harvesting variations.

2. **Energy management for Computation-Intensive systems:** This middleware targets computation-intensive systems such as real-time systems with periodic tasks. As an improved version, it takes into consideration practical energy storage constraints, i.e. supercapacitor and its leakage.

3. **Hybrid Energy storage management:** This middleware targets solar-powered embedded systems with hybrid storage, containing both batteries and supercapacitors. It orchestrates the objective of the energy management, optimizing application QoS with energy storage lifetime and efficiency.

4. **Stochastic Model and Optimization:** This middleware takes a different approach from previous three middlewares in tackling energy harvesting variations. Instead of using prediction and deterministic planning, it employs a stochastic model and optimization method based on Finite State Markov Chain to capture the variations in harvesting and application QoS.

There are many approaches to energy management in solar-powered embedded systems. This dissertation focuses on tackling energy harvesting variations and optimizing application QoS simultaneously. This dissertation is organized into chapters as follows. Section 1.5 presents the overview of our SQUARES middleware. Chapter 2 presents the background of energy harvesting systems.

Chapter 3 introduces communication-intensive systems whose energy consumption is mainly spent on data transfer. It presents my energy management middleware for communication-intensive systems focusing on wireless sensor networks and data collection application.

Chapter 4 introduces computation-intensive systems in which a large portion of energy consumption is spent on task processing. An example of such systems is real-time systems with powerful processors for real-time task processing. Chapter 4 presents my energy management middleware for computation-intensive systems with a focus on soft real-time systems. The middleware exploits task scheduling and DVFS as knobs to tune application QoS and energy consumption in order to tackle energy harvesting variations.

Chapter 5 shows the need for energy storage management in solar-powered embedded systems with hybrid storage, batteries and supercapacitors. Hybrid energy storage combines the advantages of both energy storage elements and uses one element's strength to complement the other's weakness. Ideally, hybrid energy storage helps to prolong storage lifetime and increase energy efficiency. However, energy management for such complex storage subsystem is not simple and in this chapter, I show my middleware for hybrid storage management in solar-powered embedded systems.

Chapter 6 shows my unified stochastic model and optimization middleware framework based on Finite State Markov Chain. Prediction-based and deterministic planning approach works well in situation where energy harvesting and application demand variations are not significant. In scenarios where harvesting and application demand is hard to predict accurately, stochastic model and optimization show its strength.

Finally, I conclude my work in chapter 7 and show possible directions for future work. In the next section, I give an overview of my proposed middleware framework, SQUARES.

1.5 SQUARES – Smart Quality-aware Renewable Energy System Middleware Framework Overview

SQUARES focuses on solar energy management and its objective is optimizing application QoS under system constraints which include energy harvesting variations, harvesting circuit efficiency, and energy storage limitations. SQUARES is a middleware framework sitting on top of the embedded processor (as shown on Figure 1.3). Since the tasks of this middleware framework is fairly complicated and it is computational expensive to construct models and run long-term planning algorithms on the embedded processor, I divide SQUARES into two phases as shown in Figure 1.4: an offline modeling and planning phase running on a server or cloud with unlimited power and resources, and an online adaptation and control phase running on the embedded processor itself.

The offline phase of SQUARES first uses a prediction algorithm to extrapolate future energy harvesting for the next harvesting period. Prediction algorithms exploit harvesting history, diurnal and seasonal patterns, and weather forecast. This prediction is used to plan energy budget and QoS assignment for each slot in the next harvesting period. This plan is built with the knowledge of application QoS model and system model. The server then communicates with the embedded processor about slot-based QoS assignment via wireless.

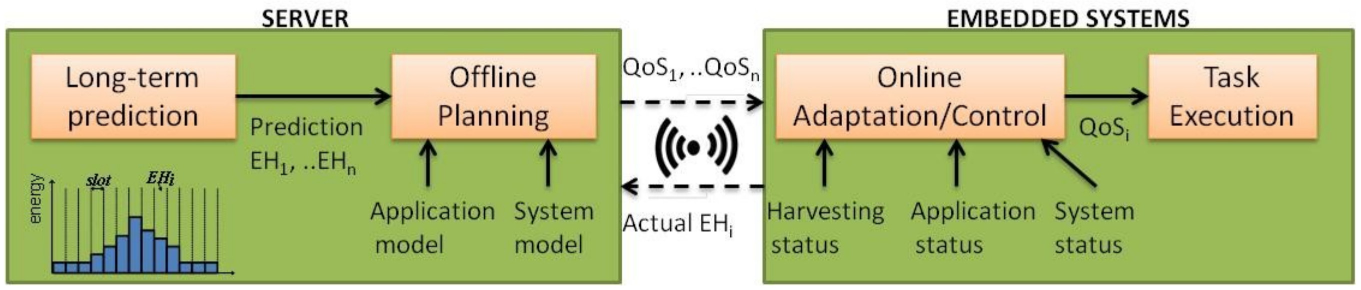


Figure 1.4 SQUARES Overview

The online adaptation on the embedded processor receives this offline planning and applies any necessary adaptation according to energy harvesting status, energy storage status, and application QoS status tracking at run-time. This updated QoS assignment control the tasks execution on the embedded processor. Tasks here can refer to sensing, processing, or communicating, or a combination of those depending on the nature of the application. Different system configurations or application options might be set in order to achieve the desired QoS level, such as frequency of sensing and communicating, or the speed at which tasks are processed on the embedded processor.

This is the overview of SQUARES middleware framework. In chapter 2, I will introduce the background of energy harvesting systems. In chapter 3-5, SQUARES middleware framework is applied in three different systems to solve problems of harvesting variations, application variations, and energy storage challenges in solar-powered embedded systems.

References

[1] http://cordis.europa.eu/ist/embedded/facts_figures.htm, Source "Future of Embedded Systems Technology". BCC Report G-229R

[2] "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", White Paper, by Dave Evans, https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[3] "What the Internet of Things (IoT) Needs to Become a Reality", White Paper, http://www.freescale.com/files/32bit/doc/white_paper/INTOTHNGSWP.pdf

[4] "Resilient dependable cyber-physical systems: a middleware perspective", by Grit Denker, Nikil Dutt, Sharad Mehrotra, Mark-Oliver Stehr, Carolyn Talcott, Nalini Venkatasubramanian

[5] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks ," in IPSN 2005, pp. 463-468

[6] Responsphere infrastructure test bed, 17 March 2011, http://www.responsphere.org/index.php

[7] Jan Kleissl and Yuvraj Agarwal, "Cyber-Physical Energy Systems: Focus on Smart Buildings" in DAC 2010

[8] Yuvraj Agarwal, Bharathan Balaji, Seemanta Dutta, Rajesh K Gupta, Thomas Weng, "Duty-Cycling Buildings Aggressively: The Next Frontier in HVAC Control", in IPSN/SPOTS 2011

[9] Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng, "Occupancy-Driven Energy Management for Smart Building Automation", in Buildsys 2010

[10] Shang-Wen Luan , Jen-Hao Teng ; Shun-Yu Chan ; "Development of a smart power meter for AMI based on ZigBee communication", in International Conference on Power Electronics and Drive Systems, 2009. PEDS 2009.

[11] P. McDaniel, S. McLaughlin, "Security and Privacy Challenges in the Smart Grid", IEEE Journal of Security and Privacy, volume 7, issue 3, 2009

[12] C.M. Lampert, "Switchable glazings for the new millennium", Proceedings of the Eurosun, Copenhagen, Denmark, 2000

[13] C.M. Lampert, "Functional coatings—displays and smart windows", in: H.A. Meinema, C.I.M.A. Spee, M.A. Aegertner (Eds.), Proceedings of the Third International Conference on Coatings for Glass, Maastricht, NL, 2000

[14] Irvine Smart Grid Demonstration Project,
http://www.sustainability.uci.edu/Resources1/ISGDOverview.pdf

[15] U. Cetintemel, A. Flinders, Y. Sun, "Power-efficient Data Dissemination in Wireless Sensor Networ",in MobiDE'03

[16] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, B. Krogh, "Energy-Efficient Surveillance System Using WirelessSensor Network", in MobiSYS'04

[17] C. Park and P. H. Chou, "AmbiMax: Autonomous Energy Harvesting Platform for Multi-Supply Wireless Sensor Nodes," in 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON), Reston, VA, 2006, pp. 168-177.

[18] Farhan Simjee and Pai H. Chou, "Everlast: Longlife, Supercapacitor operated Wireless Sensor Node" in ISLPED 2006

Chapter 2 Energy Harvesting Embedded System Background

In this chapter, I present a system model for energy harvesting embedded systems, highlight some system prototypes and related work at all layers of hardware and software. There are four main components in an energy harvesting embedded system: (a) energy transducer (harvesting devices), (b) energy harvesting circuit, (c) energy storage subsystem, and (d) the load. Figure 2.1 shows the hardware/software layer overview of an energy harvesting embedded system. The load can be a sensor board, a micro processor or a general purpose processor running a software stack and applications. Hardware components of the load also include sensor(s) and radio chip for wireless communication. The software stack comprises of the network layer, OS, middleware, and the applications.

Energy harvesting from the surrounding environment are subject to availability of energy sources both in time and in space. Classification and characteristics of energy harvesting sources are presented in Section 2.1. Energy goes through several transformation steps before being used by the load; each step has its own efficiency: conversion efficiency, harvesting efficiency, buffering efficiency, and consumption efficiency [1].

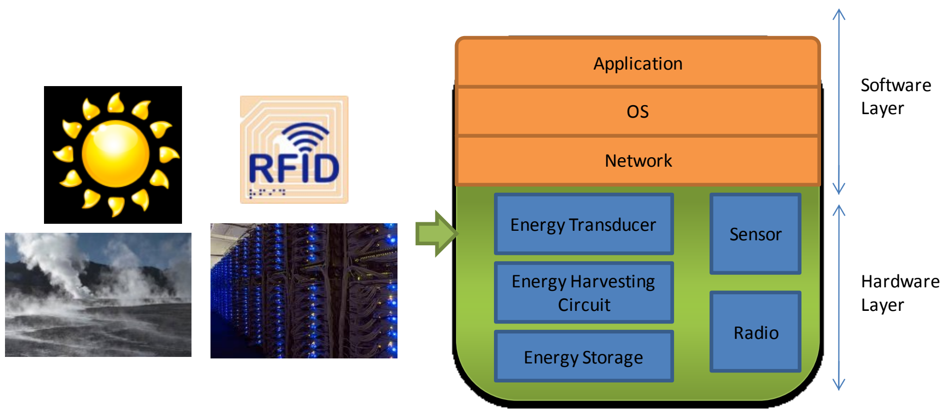


Figure 2.1 Energy Harvesting System Overview

Depending on the location where sensors are deployed, cost, size constraints of the system, and power demand of applications, a feasible energy source(s) is identified and suitable energy transducers are chosen. In order to select and set up the right configuration for harvesting devices, designers must understand harvesting circuit component characteristics and their configuration options. I discuss hardware components including energy harvesting circuit and energy storage subsystem, their characteristics, and configuration options in Section 2.2.

A good hardware layer for energy harvesting embedded system is capable of efficiently harvesting energy and storing energy in a storage subsystem. However, as the energy storage replenishes at a varying rate, it is necessary to have an energy management scheme to address the challenges of sustainability and efficiency, to guarantee continuous services and application requirement satisfaction. The software stack for energy management in energy harvesting embedded systems will be discussed in Section 2.3

2.1 Energy Harvesting Sources

Energy harvesting sources are those available in the surrounding environment. Energy harvesting sources can be classified into two groups according to characteristics of their source generation:

- Natural sources are those available readily from the environment such as sun light, wind, and geothermal heat.

- Artificial sources are those generated from human or system activities. They are not part of the natural environment. Examples are human motion, pressure (on shoe inserts) when walking or running, and system vibration or heat when operating.

System designers need to take energy harvesting source type into account for two reasons. First, natural sources are influenced by natural factors such as weather, temperature, season while artificial sources are influenced by schedule of human and machine systems. This will impact, for example, prediction mechanism for each harvesting source type. Second, natural sources do not cost extra energy to generate. There could be effects on the environment through harvesting natural sources at large scale which is outside the scope of our study of energy harvesting embedded systems. Artificial sources, on the other hand, require human/machine systems to expend energy in order to generate harvestable energy. This generating energy should not be considered as a cost if it is used mainly for other purposes such as lighting a room or running a computer system. The available harvestable energy is thus just a side effect of these processes. However, if the generating energy is mainly used to generate harvestable energy it is considered a cost. This could happen, for example when a light bulb is turned on for some extra hours just to charge a sensor equipped with solar panels; or radio spectrum is generated to charge a RFID harvesting sensor.

Table 2.1 Energy Harvesting Sources

Energy source	**Type**	**Typical power**
Outdoor solar light	Natural	100 mW/cm^2 (outdoor),
Indoor office light	Artificial/Natural	100μW/cm^2 (artificial light) – 10mW/cm^2 (filtered solar light)
Ambient radio frequency	Artificial	0.001μW/cm^2(WiFi) - 0.1μW/cm^2 (GSM)
Thermoelectric	Artificial	60 μW/cm^2
Vibration	Artificial	4 μW/cm^3 (human motion) 800 μW/cm^3 (machines)
Ambient airflow	Natural/Artificial	1 mW/cm^2
Acoustic noise	Natural/Artificial	960 nW/cm^3

Table 2.1 shows different energy sources for harvesting, their source type, and typical harvesting power. Energy harvesting sources can be classified into four groups based on two characteristics: controllability and predictability [2]. Controllability means whether an energy harvesting system has full/partial control over its energy harvesting sources. Predictability means the degree to which the energy harvesting source can be modeled and predicted. The four groups are:

- Uncontrollable but predictable: Natural sources are typically uncontrollable but some sources exhibit or follow predictable patterns such as solar energy. For artificial sources, the schedule and impact of the generating systems or human can be known beforehand or be predicted so energy harvesting availability is predictable to a certain degree. However, they often operate independently from the harvesting systems and hence they are not controllable.
- Uncontrollable and unpredictable: Natural sources can be uncontrollable and behave in a random way. For instance, in mobile systems, the surrounding energy harvesting sources are uncontrollable and unpredictable due to the stochastic mobility of the systems.
- Controllable and predictable: Artificial sources can be fully controlled if a central control system, which is authorized to and is capable of coordinating both the generating system and the harvesting system, exists. For example, a control system schedules turning on/off the lights or charging wirelessly via radio frequency to create harvesting opportunities for harvesting systems. It is also possible to predict the availability of artificial sources to some extent given the schedule and impact of human and generating system activities.
- Partially controllable: Artificial sources can be partially controlled by human or systems but with uncertain result in energy harvesting.

Among these four groups, the first group has so far yielded the most research interest, the energy sources cannot be controlled but its behavior can be modeled to predict energy harvesting availability. Degree of predictability varies according to energy sources and the granularity of prediction. Coarse-grained prediction such as on a daily basis in general yields higher accuracy than fine-grained prediction such as at minute or second intervals. Nevertheless, prediction at various time intervals is still important. For example, long-time coarse-grained prediction is sufficient for offline planning while short-time fine-grained prediction is more useful for online adaptation. All in all, a thorough understanding of the target energy harvesting source is crucial in building an efficient harvesting system. In the following sections, I describe the hardware components and software stack in energy harvesting embedded systems.

2.2 Energy Harvesting Circuit Components

In this section, I discuss essential hardware components that build up an energy harvesting embedded system. I present different options for each component and trade-offs between system cost, size, efficiency, lifetime, and other important factors to embedded system design.

2.2.1 Energy Transducer

Energy transducers are hardware devices transforming energy harvesting sources into electrical power. Harvesting devices have different cost and power conversion efficiency depending on the materials used to build the transducer. For example, monocrystalline silicon, polycrystalline silicon, or thin films are alternative materials to build solar panels. Typical solar cell efficiency is around 18% [3] but it is expected to improve with new technologies.

Existing work in literature [4, 5, 6] focus on design consideration for energy harvesting embedded systems. Taneja et al. [4] gave some guidelines for selecting hardware components and their corresponding size. Daily energy requirement for the system is computed based on empirical estimation of load power. This requirement and estimation of energy harvesting are important factors to drive the process of choosing solar panel size and energy storage elements. For example, in the case of Hydro-Watch system deployed in a forest [4], each node has only about half an hour sun light each day. Given efficiency of hardware components, they suggested that solar panel size in this case should be sufficient large to produce harvesting power of *15* times load power requirement during its limited exposure time to direct sun light. This over design ensures that the system has enough energy in storage to sustain its operation during non-harvesting periods.

2.2.2 Energy Harvesting Circuit

Energy harvesting circuit is one of the most crucial parts of the hardware in an energy harvesting embedded system. It calibrates and maximizes the output from the energy transducer, routes the energy to power the load directly, or deposits into the energy storage subsystem. Energy harvesting circuit monitors the transducer output and energy storage status, and possibly makes this information available to upper software layers. Each of these tasks is often handled by individual hardware circuit. In this chapter, I call them in general the energy harvesting circuit.

Maximum Power Point Tracking (MPPT): Most energy harvesting source has a special voltage–current (I–V) characteristics curve. In Figure 2.2, I show the model of I–V curve for a solar panel. For energy harvesting embedded systems, empirical characterization and manual calibration of energy transducer are often required to obtain this I–V curve model. Measuring, modeling, and understanding I–V curve is important since it reveals the Maximum Power Point at which the highest power is attained by the harvesting circuit. Without MPPT, the efficiency loss can range from *30%* to *90%* of the available power [1].

However, the I–V curve for a harvesting device is dynamic as it is sensitive to ambient factors such as temperature and solar irradiance level. Figure 2.2 shows the I–V curves under various temperature and irradiance condition (extracted from a model of Sunpower A300 solar cell). It shows that Maximum Power Point changes under different environmental conditions. For AC sources such as vibration, Maximum Power Point is related to the resonant frequency of vibrating devices and magnitude of the physical oscillation. Operating energy transducers at Maximum Power Point gains significantly more power than sub-optimal points. Therefore harvesting circuits should employ Maximum Power Point Tracking methods to improve their efficiency.

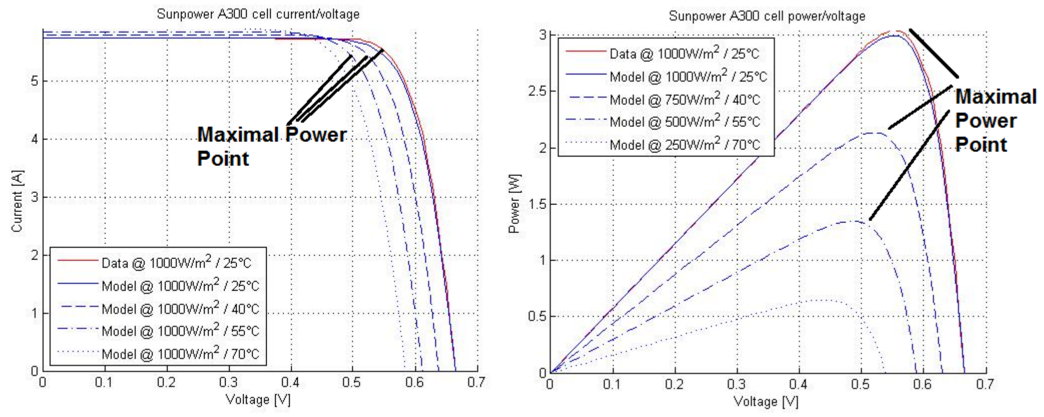


Figure 2.2 I-V Characteristics of an Example Solar Cell

Maximum Power Point Tracking is a practice to maximize harvesting power by adjusting the impedance load of the harvesting transducers. Many methods have been proposed for macro-scale harvesting systems, a survey is given in [7]. Chou and Kim [8] classified MPPT approaches for energy harvesting embedded systems (with limited memory and stringent energy consumption requirement) into two categories: load matching and supply side MPPT. In the load matching approach, the load is adjusted by duty cycling, dynamic power management (e.g., DVFS), or other power management techniques to match with energy transducer load and hence, maximizing the harvesting power. This MPPT approach is very application-specific and can only be managed at run-time by software stack.

The second approach, supply side MPPT, is further divided into two types: sensor-driven MPPT and perturbation-based MPPT. In the sensor driven MPPT, sensors are employed to measure environment conditions. Sensor values are then used to determine the optimal impedance load corresponding to Maximum Power Point from a look-up table. This method is simple to implement and it runs fast. However, it blocks a portion of the limited memory to store the look-up table. In addition, sensors consume energy and they are subject to aging and other forms of deterioration which might require recalibration.

Perturbation-based MPPT techniques are open circuit voltage, short circuit current, hill climbing, and I–V curve sweeping. One method is called Fractional Open-Circuit Voltage in which $V_{mpp}=KV_{oc}$ where V_{mpp} is voltage at Maximum Power Point and V_{oc} is open circuit voltage. K is a constant, typically between 0.71 and 0.78 for photovoltaic modules [9, 10]. This approximation method has low energy overhead and does not require many sensors or calibration at the trade-off of lower accuracy. In the curve sweeping method, the I–V curve is profiled at run-time using sensors and calibration if needed. This method is more robust at the trade-off of a complex MPPT circuit and higher overhead both in time and energy consumption.

MPPT methods can be implemented in hardware (analog circuit) or software (running on the Main Control Unit, MCU). Software implementation is reusable in any system but it is unable to re-calibrate energy transducers. On the other hand, implementation in analog circuit consumes very low power and allows continuous and quick response to Maximum Power Point changes. The hardware for such MPPT, however, must be designed for each specific harvesting system.

From another point of view, Taneja et al. [4] argue that energy transducer such as solar panel can be chosen to operate near its Maximum Power Point given the combination of the load and energy storage. Hence they do not use MPPT in their design of Hydro-Watch harvesting circuit but select solar panel that best matches the system load and energy storage sub-system.

Voltage regulators: There is usually a gap between the transducer output voltage and energy storage voltage or between energy storage voltage and the load voltage. Voltage regulators are necessary to bridge this gap. The options are either linear regulators or switching regulators and the trade-off is between their conversion efficiency and generation of clean, stable output power. Switching regulators have been mainly used in most energy harvesting embedded systems. Switching regulators could be diodes, bucks, boosts, or a combination of buck-boost regulators such as pulse frequency modulation (PFM) regulators. Bucks perform voltage step-down while boosts perform voltage step-up. Among these options, PFM regulators are considered most effective since each regulator has a switching capacitor regulator to avoid wasting energy in diode at low voltage and a buck converter to prevent shorting the input and output. Other components which might be needed are DC-AC or AC-DC adapters depending on the types of current generated by the energy harvesting process and the types of current accepted by the load.

Multi-dimensional array of harvesting transducers: Multiple harvesting devices can be combined in series or in parallel to form an energy harvesting sub-system. Harvesting arrays are reconfigurable at run-time, allowing systems to adjust output voltage or current dynamically. However, such arrays of harvesting transducers are subject to size and packaging constraints of embedded systems.

Heterogeneous/homogeneous harvesting systems: If one source of energy harvesting is not sufficient to provide energy for system operation, several energy sources could be harvested at the same time to complement each other in term of harvesting availability and to increase overall energy supply. This adds complexity to the harvesting circuit because each energy harvesting source requires a separate energy transducer and a different MPPT method. Each source might require independent energy storage with a specific charging algorithm for efficiency purpose. Because of this separation and independence of resources, heterogeneous harvesting systems often have an energy harvesting subsystem for each energy harvesting source [6, 11].

2.2.3. Energy Storage Subsystem

Systems could be powered directly from energy harvesting sources but large variations in the energy sources will make the systems unstable. Therefore, to smooth out the variation effect, energy storage is often used to buffer harvested energy. There are currently two choices for energy storage in an energy harvesting embedded system: rechargeable batteries and supercapacitors (also called ultracapacitors or electrochemical double layer capacitors). These two types of energy storage elements are significantly different from each other as summarized below.

Batteries have higher energy density (more storage capacity for a given volume or weight). Rechargeable batteries can be recharged multiple times but they are subject to aging effect and rate capacity constraints. Characteristics of four types of rechargeable batteries are presented in Table 2.2. Sealed Lead Acid (SLA) and Ni-cadmium are used less often because of their low energy density and low power density. According to [5], there are several trade-offs between Nickel-Metal Hybrid batteries (NiMH) and lithium-ion batteries (Li-ion). Li-ion batteries are often used because they more efficient and have longer lifetime. However, they are more expensive and require a more complicated charging circuit. They might not accept charging at low rate which often happens in energy harvesting circuits. Characteristics of any battery type could vary according to operating temperatures. This is important for energy harvesting systems since they are usually exposed to harsh conditions like strong wind, direct sunlight, and thermal heat.

Table 2.2 Comparison of Rechargeable Battery Types (adapted from [4] and [12])

Battery Type	Energy Density (MJ/kg)	Power Density (W/kg)	Efficiency (%)	Discharge Rate (% per month)	Recharge Cycles
Sealed Lead Acid	0.11-0.14	180	70-92	3-4	500-800
Ni-cadmium	0.14-0.22	150	70-90	20	1500
NiMH	0.11-0.29	250-1000	66	20	1000
Li-ion	0.58	1800	99.9	5-10	1200

Supercapacitors do not have aging problem, they offer long lifetime with unlimited or high charge-discharge cycles. They have high power density but low energy density. High leakage which increases exponentially with supercapacitor voltage (even when idle) is a disadvantage. Another problem with supercapacitors is the cold start, which was addressed in [13] using a feed forward PFM.

Chou and Kim [8] suggested hybrid schemes for energy harvesting storage in which batteries and supercapacitors compensate each other and utilize advantages of both energy storage technologies. More of this hybrid energy storage architecture will be discussed in chapter 6.

2.2.4. Case Study of Energy Harvesting Embedded Systems

Recent research has enabled embedded systems, in particular, wireless sensor motes to have capability of harvesting energy from surrounding environment. Many prototype platforms have been built successfully including Heliomote [15], Prometheus [3], Everlast [15], Ambimax [6, 16]. Beside solar and wind, technologies have made it feasible to harvest from other renewable sources such as vibration, RFID, geothermal, human motion [17–19]. I present here several working prototypes of energy harvesting embedded systems, many of them have been deployed in real- life applications.

Prometheus [3] is one of the first designs for solar-powered embedded systems. It focuses mainly on the energy storage subsystem and there is no MPPT. It argues that in most latitudes we only expect a few hours of direct sunlight, therefore the systems need large buffers to store harvested energy and to power themselves through the night. In their design, there are a primary buffer and a secondary buffer. Supercapacitor is chosen as the primary buffer for its longer lifetime and capability of frequent pulse charging. However, larger supercapacitors have greater leakage current. Given charging, discharging, and leakage rate of a harvesting system, Prometheus finds the theoretical optimal capacitance of its supercapacitor. It also proposes configuration for supercapacitors such as connecting two supercapacitors in series, in order to reduce leakage and match with solar output voltage instead of using MPPT methods.

Prometheus chooses Lithium battery as the secondary buffer which is neither charged nor discharged frequently but holds backup energy for an extended time. A rechargeable battery is more suitable than a supercapacitor for secondary buffer because of its low leakage, high energy density, and high voltage for a single cell. In their experiments, they run a simple driver program controlling a power switch (either drawing load power from the primary buffer or the secondary buffer) and sending energy harvesting statistical information to a base station at *1%* duty cycle.

Heliomote was designed at UCLA [15]. In the first version, Raghunathan et al. [5] argued that the Maximum Power Point changes slightly within a day. Therefore they avoid using MPPT circuit by carefully selecting a suitable battery. They use NiMH batteries as energy storage because the charging circuit is simplified compared to Li+ batteries. In Helimote3, its harvesting circuit is integrated with a MPPT method to actively learn the solar panel's I–V characteristics and to reconfigure itself at run-time. Notably, the platform is equipped with an on-board measurement system providing voltage and current output of the solar transducer and the battery terminal voltage which could be used in tuning or optimizing overall system performance.

The overall efficiency of their energy harvesting and storage subsystem is *80–84%*. It is tested by running an application of ecosystem sensing at James Reserve Mountain [63]. The measurement results show the system can run at *20%* duty cycling which is a very promising result compared to a typical non-rechargeable battery system running at *1–5%* duty cycling.

Hybrid energy harvesting systems: Due to the intermittent nature of energy harvesting sources, several efforts explored the possibility of building hybrid energy harvesting systems with the goal to increase energy harvesting availability overall. Ideally, different sources complement each other for a stable power source in time and in space. Tan and Panda [20] designed a hybrid energy harvesting system consisting of both indoor ambient light and thermal energy harvesting circuits.

Ambimax was designed at UC Irvine by Park and Chou [6]. Ambimax is a hybrid energy harvesting system with both a solar panel and a wind generator. Each energy source is managed separately by individual energy harvesting subsystems with source-specific MPPT methods and efficiently charging different supercapacitors. A PWM (pulse width modulation) is put between the energy harvesting transducer and the supercapacitor. This isolation keeps the supercapacitor from degrading energy harvesting efficiency and allows harvesting when $V_{source} < V_{supercapacitor}$.

This PWM switching regulator combined with a comparator and sensors creates a perturbation-based MPPT circuit. In comparison with Prometheus, Ambimax shows a charging time 12.5x faster. An improvement of Ambimax was presented in Duracap [13] which includes three supercapacitors to improve system reliability during the cold booting phase. Carli et al. [11]

proposed a similar architecture with independent energy harvesting subsystems but emphasized on fully analog implementation of MPPT, charging algorithm, and power management.

Indoor energy harvesting systems: Indoor environment has many potential sources for harvesting, each with intensity and availability different from the same sources outdoor. The most accessible source is light in offices and hallways. Hande et al. [21] devised a system to harvest energy from fluorescent light in hospital hallways to support routing of patient data in clinics using Micaz motes. Tan and Panda [20] carried out an extensive indoor energy harvesting measurement over 16 months in different settings. EnHantTag [22, 23] are small ultra-low- power devices harvesting both light and RFID, and supporting novel applications such as tracking personal items and locating disaster survivors. Other energy sources such as kinetic, vibration, magnetic can be harvested as well. inDOOR Energy Harvester was a project at New York University [24] that converted kinetic energy from opening and closing a door to electrical energy.

Simulator: In addition to hardware prototypes, designers need tool chain such as simulators to support the design process. Simulators allow designers to explore the design space, to evaluate performance of a design candidate in a modeling environment with reproducible inputs and conditions, and to choose optimal configuration options for their energy harvesting embedded systems. It can also be used for comparison of designs, topologies, and algorithms.

A solar power simulator S# was developed by Li and Chou [25]. The simulator is a programmable power supply used to simulate or emulate electronic behavior of a solar panel. In the simulation mode, S# takes a sunlight profile as inputs, looks up the built-in solar power model and generates a simulated power output trace. The simulation model could take locations and configuration of solar panels and weather condition as inputs and to generate customized power output.

Another simulator for energy harvesting platforms is developed by Jeong [26] at UCLA. This Matlab-based simulator captures behavior of the main components of an energy harvesting system: solar irradiation, the solar panel, the energy storage, and the energy harvesting circuit including both input and output regulator. Solar irradiation is modeled using an astronomical model. It is further improved by integrating obstruction model from surrounding objects and a weather conditions such as cloud coverage and horizontal visibility. Other components of energy harvesting embedded systems are modeled based on their electronic properties and characterization.

In this section, I have presented state-of-the-art research in building efficient energy harvesting embedded systems from hardware perspective. In the next section, I present the software stack that works in concert with hardware components in order to realize the best benefits of energy harvesting and sustainability.

2.3 Software Stack

In traditional battery-powered systems, the conventional ultimate goal is to maximize system lifetime given the limited battery capacity. To address this problem, researchers proposed many energy-efficient and power-efficient approaches, from energy-efficient sensor placement [27–29], routing and communication protocols [30-32], low power MAC protocols [33], duty cycling techniques [34], DVFS [64], to adaptive data rate [35]. These approaches aim to minimize energy consumption to prolong system lifetime while barely meeting the requirements of applications.

These assumptions and goals must change in the context of energy harvesting embedded systems. Renewable energy sources regenerate automatically and they power the systems for indefinite time subject only to hardware longevity and failures. A widely used new constraint in energy harvesting systems is energy neutrality, proposed by Kansal et al. [2]. Energy neutrality means an energy harvesting system can sustain its desired operation level by not consuming more energy than the amount of energy harvested.

Reducing power consumption below the level needed for energy neutrality will not increase system lifetime or system utility. On the other hand, just barely meeting energy neutrality constraint might not utilize all harvested energy. The remaining energy must be stored in the energy storage which has limited capacity and even leakage. For example, running 5–10% duty cycling on an energy harvesting embedded system and maintaining energy neutrality is possibly feasible but not necessary optimal. A smart approach would be adjusting power consumption (e.g., duty cycling) according to energy harvesting profile, spending the right amount of energy to optimize system performance and storing the right amount for back-up at times of low or no harvesting activity. In order to set the right amount for energy consumption and storage, and to achieve the optimal performance, it is important to share information about energy harvesting condition and energy storage status among system layers and even across the network.

One of such important information is prediction of future energy harvesting availability. Before going into details of power management using software, I present related research on energy harvesting prediction which is used extensively in energy management schemes for energy harvesting embedded systems.

Energy Harvesting Prediction: Renewable energy sources such as solar energy show recurring patterns (diurnal and seasonal patterns) that can be utilized to predict future energy harvesting availability. The prediction of energy harvesting is very important for simulation, estimating system performance, and planning system activities. However prediction algorithms must be lightweight with small memory footprint if it is to run on limited-resource embedded systems.

There are several prediction algorithms to estimate future availability of energy harvesting at coarse-grain (slot-based) level, i.e., every *30* min or per hour. Hsu et al. [36] proposed a prediction algorithm based on Exponentially Weighted Moving Average (EWMA). EWMA is a method to compute weighted average of data with the weight factors decreasing exponentially. When it is applied for time-series data analysis, by adjusting weight factors, short-term fluctuations can be smoothed out and long term trend is emphasized. A harvesting period, typically a day, is divided into N slots. In this algorithm, N is chosen to be *48* for low memory overhead, each slot is *30* min. They assume that on a typical day, amount of energy harvested in a slot is similar to that of the previous days in the same time slot. The energy generated in a particular slot hence is predicted as weighted average of the energy received in the same time slots during previous recorded days. Their experiments show the absolute error between the predicted and the actual energy profile is from *2* to *10 mA* (out of *60 mA*), which is up to *16.6%* error.

$$x_k = \alpha x_{k-1} + (1-\alpha)x_k \qquad \textbf{(2-1)}$$

Recas et al. [36] noticed that EWMA algorithm proposed in [37] is only accurate if the weather is consistent or "typical." Hence, they introduced another prediction algorithm called Weather-Condition Moving Average (WCMA) that does not only take into account the weights in each time slot but also the changing condition in energy harvesting profiles throughout a day. A similar principle was exploited in another energy harvesting prediction scheme proposed by Noh et al. [38]. In [36], the predicted energy value on day *i*, sample *n + 1* is:

$$E(d, n+1) = E(d, n) + \ GAP_k(1-\alpha) \times MD(d, n+1) \qquad \textbf{(2-2)}$$

where *E(d, n)* is the previous sample on the same day and *MD(d, n + 1)* is the mean of *D* past days at the same time sample *n + 1*. GAP_K is a weight factor measuring the weather condition in the present day in relative to the previous days. WCMA claims lower average error of *9.8%* as compared to EWMA's average error of *28.6%* in an experiment consisting of 45 harvesting days.

Jeong [26] and Sharma et al. [39] leverage weather forecast and extract cloud coverage information to improve their energy harvesting prediction algorithms. The later one uses these formulations for solar and wind prediction:

$$Power(t) = MaxPower(1 - cloud\_coverage(t)) \qquad \textbf{(2-3)}$$

$$Power(t) = 0.01787485 \times WindSpeed(t)^3 - 3.4013 \qquad \textbf{(2-4)}$$

where MaxPower is maximum solar power derived from the typical solar radiation at a given latitude, altitude, and at a specific time of the year. Renner and Turau [40] proposed another method to actively learn energy harvesting profile and to adapt the number of slots and duration of each slot at run-time. Its goal is prediction accuracy with small memory footprint for energy harvesting prediction algorithms running on embedded systems.

There also exist commercial tools for predicting energy harvesting such as iPV, iSV, SolarShade apps for smart phones and SunEye, Solar Pathfinder devices. The smart apps are able to record obstruction while the user traces the phone along the skyline where the solar panel is deployed. It utilizes available sensors on smart phones such as compass and inclinometer to identify position and elevation of the obstructing objects and overlays them on the sun plot. Using the shading effect derived from the overlaid sun plot, a built-in weather station database, and solar panel model, these smart apps produce an estimate of monthly solar energy at the user's location. However, smart app documentation and verification of method are limited.

Cross-layer approaches have been proposed to exploit energy harvesting prediction to adapt systems and tackle energy harvesting challenges (section 1.2). I classify research work in cross-layer power management schemes for energy harvesting embedded systems into three groups: node layer, operating system layer, and application layer adaptation. These power management schemes consist of three steps: leaning and predicting the energy harvesting profile, adapting power consumption at each layer to match with harvested energy, and fine-tuning power scaling to account for battery non-idealities and prediction error.

Node layer: Node layer management refers to management of hardware components such as sensors, radio chips, processors, and possibly energy storage subsystem using software. Kansal et al. [2, 41] and Hsu et al. [36] presented several power management schemes at the node layer. Duty cycling between active and low power modes for the purpose of performance/power scaling is a good option since most embedded systems provide at least one low power mode in which the power consumption is negligible. Hsu et al. [36] proposed an algorithm to adapt duty cycling rate of systems to the changes in renewable sources.

OS layer: Operating system controls how tasks such as sensing, processing, and communicating are scheduled using a scheduling algorithm(s). There are works in task scheduling, multi-version scheduling, and dynamic voltage frequency scaling (DVFS) at the OS layer of energy harvesting embedded systems. Moser et al. [42, 43] proposed a task scheduling technique for energy harvesting systems called Lazy Scheduling, which delays task execution to harvest and store more energy until tasks must be executed to meet their respective deadlines. Liu et al. [44, 45] extended these task scheduling techniques with DVFS capability. Steck and Rosing [46] and Ravinagarajan et al. [47] adapted task utility of structural health monitoring applications (coupled with DVFS technique) to maximize accuracy of tasks while sustaining the system under energy neutrality constraint.

Application layer: At the application layer of energy harvesting embedded systems, there are related works in adapting data quality, data update frequency, and quality of services in order to meet energy neutrality constraint. Moser et al. [48–50] presented a system model with different abstract levels of quality. Each level is associated with an energy demand and a corresponding reward. They proposed an optimal solution using ILP and an approximation dynamic programming technique to allocate energy budget and assign a quality level to each time slot in a harvesting period. This assignment must meet energy-neutrality constraint and maximize the total reward at the same time. Noh et al. [38] proposed a minimum variance slot-based energy budget allocation for systems which prefer a steady level of operation.

For applications with varying requirements, more dynamic energy budget schemes are required. Software support for energy harvesting embedded systems is unstructured so far and it is difficult to guarantee all approaches work in concert with each other to produce the optimal result. A middleware layer providing software services and information about energy harvesting statistics, and battery status is desirable. Such middleware layer allows tuning parameters, e.g., changing duty cycle rate, selecting scheduling algorithm, choosing budget allocation scheme, and turning on/off database services. Middleware layer will play an important role to connect hardware and software layers, enabling cross-layer adaptation and system performance optimization.

In a broader scale, i.e., networks of energy harvesting embedded systems presented in the next section, middleware and network layers will have a crucial role. These two layers connect systems in the network, enable sharing energy harvesting information beyond a single system's boundary, and allow application layers on different nodes to communicate and coordinate for more complex activities.

2.4 Networked Energy Harvesting Embedded Systems

So far, I focused on individual energy harvesting embedded systems. In this section, I present the model and structure of energy harvesting networks and related research. Many applications in smart spaces leverage connection and information sharing in network to monitor properties of the environment, to detect unsupervised events, and to relay the processed information to a central base station(s).

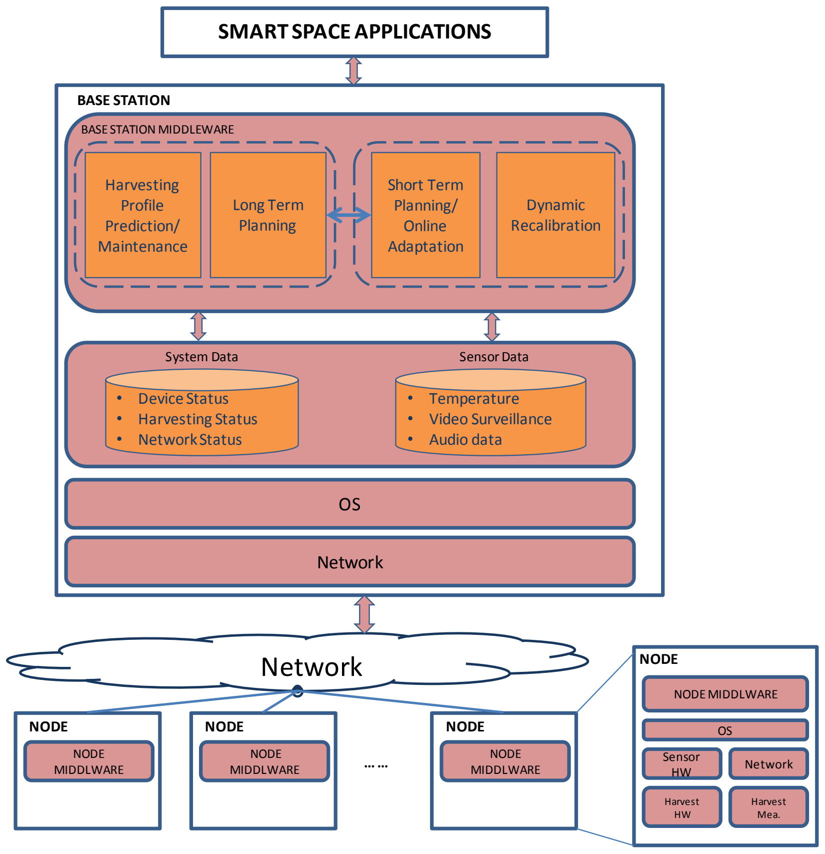


Figure 2.3 Architecture of an Energy Harvesting Network

Each energy harvesting node has hardware and software components working together as described in the previous section. A middleware layer is proposed on each node to enable sharing energy harvesting information, cross-layer optimization and in-network optimization. Data from nodes are sent to the base station(s) and vice versa through application and network protocols. Collected data is stored in a database at the base station and/or sent to the smart space applications. With unlimited resources (power, computation units), middleware layer on the base station can do computational-intensive tasks such as energy harvesting prediction, long-term and short-term planning, hardware and software recalibration.

Network layer: The network layer manages communication at packet level. Packets are sent from source to destination according to network protocols. In energy harvesting networks, packets should be routed along paths that do not only ensure delivery but also maintain energy sustainability. Each node sends its own sensor data and also forwards packets of other nodes in the network. Therefore, energy budget for communication on each node must consider both these internal and external data streams. Routing is an important challenge since there are both communication vs. computation trade-off on each node and data traffic balancing among nodes according to harvesting capability of each node in the network.

Voigt et al. [51] and Islam et al. [52] modified LEACH, a cluster-based routing protocol for sensor networks to take advantage of energy harvesting. Lattanzi et al. [53], Lin et al. [54, 55], Zeng et al. [56], Hasenfratz et al. [57], and Jakobsen et al. [58] modified existing energy-efficient routing protocols to exploit both temporal and spatial variations of renewable energy and to maximize data delivery for sensors. Different from traditional battery-residual-based routing cost, Kansal et al. [2] proposed an enhanced routing cost metric that takes into consideration both the harvesting potential of a node and its residual battery level. Once routing cost for each link is established, Bellman Ford algorithm, shortest path algorithms, and variants of these algorithms are deployed to find minimum cost routes between sources and destinations.

Middleware layer: As discussed, energy harvesting nodes in a network should communicate to share energy harvesting statistics for power management and in-network optimization. Kansal and Srivastava [59] proposed a framework which actively learns the properties of the renewable energy sources, predicts future energy availability, and distributes this information in the network for power management. It suggested several uses of this framework including topology management, clustering, leader election, load balancing, transmission power control, and network routing.

Networking application layer: For applications such as storage services in a wireless sensor network, Wang et al. [62] proposed an adaptive technique to turn on and off the storage services based on different energy thresholds. Fan et al. [60] and Zhang et al. [61] attempted to maximize data rate and utility-based data rate for data collection applications in energy harvesting WSNs.

In conclusion, I have provided a thorough background for energy harvesting embedded systems in this chapter, from characterization of renewable sources, energy harvesting hardware, to software, from individual harvesting systems to a network of harvesting systems. The literature provided here however is not in any way complete as energy harvesting systems are improving and new works are proposed. The readers are therefore encouraged to explore further.

References

[1] V. Raghunathan and P. H. Chou, "Design and Power Management of Energy Harvesting Embedded Systems ," in ISLPED'06. Proceedings of the 2006 International Symposium on Low Power Electronics and Design, 2006., Tegernsee, 2006, pp. 369-374

[2] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," ACM Transaction on Embedded Computomg Systems, vol. 6, no. 4, pp. 1539-9087, Sep. 2007

[3] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks ," in IPSN 2005, pp. 463-468

[4] J. Taneja, J. Jeong, and D. Culler, "Design, Modeling, and Capacity Planning for Micro-solar Power Sensor Networks ," in IPSN '08. International Conference on Information Processing in Sensor Networks, 2008., St. Louis, MO, 2008, pp. 407-418

[5] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani Srivastava. "Design considerations for solar energy harvesting wireless embedded systems", IPSN 2005

[6] C. Park and P. H. Chou, "AmbiMax: Autonomous Energy Harvesting Platform for Multi-Supply Wireless Sensor Nodes," in 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON), Reston, VA, 2006, pp. 168-177.

[7] Trishan Esram, Patrick L. Chapman, "Comparison of Photovoltaic Array Maximum Power Point Tracking Techniques", IEEE Transaction on Energy Conversion, volume 22, issue 2, June 2007

[8] Pai H. Chou and Sehwan Kim, "Techniques for Maximizing Efficiency of Solar Energy Harvesting Systems", ICMU 2010

[9] Bekker B, Beukes HJ. Finding an optimal panel maximum power point tracking method. In: 7th African IEEE Conf.; 2004. p. 1125–9.

[10] K. Kobayashi, H. Matsuo, and Y. Sekine, "A novel optimum operating point tracker of the solar cell power supply system," in Proc. 35th Annu. IEEE Power Electron. Spec. Conf., 2004, pp. 2147–2151.

[11] Davide Carli, Davide Brunelli, Luca Benini and Massimiliano Ruggeri, " An Effective Multi-Source Energy Harvester for Low Power Applications", DATE 2011

[12] Sujesha Sudevalayam and Purushottam Kulkarni, "Energy Harvesting Sensor Nodes: Survey and Implications", IEEE Communications Surveys and Tutorials, volume 13, issue 3, 2011

[13] Chien-Ying Chen, Pai H. Chou, “DuraCap: a Supercapacitor-Based, Power-Bootstrapping, Maximum Power Point Tracking Energy-Harvesting System”, ISLPED 2010

[14] K. Lin, et al., "Heliomote: enabling long-lived sensor networks through solar energy harvesting," in SenSys '05, New York, 2005, pp. 309—309

[15] Farhan Simjee and Pai H. Chou, “Everlast: Longlife, Supercapacitor operated Wireless Sensor Node” in ISLPED 2006

[16] Davide Carli, Davide Brunelli, Davide Bertozzi and Luca Benini, “A high-efficiency wind-flow energy harvesting using micro turbine”, International Symposium on Power Electronics Electrical Drives Automation and Motion (SPEEDAM), 2010

[17] S. Meninger, J. O. Mur-Miranda, R. Amirtharajah, A. Chandrakasan, and J. H. Lang, "Vibration-to-electric energy conversion," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9, no. 1, pp. 64-76, Feb. 2001.

[18] H. A. Sodano, G. E. Simmers, R. Dereux, and D. J. Inman, "Recharging Batteries using Energy Harvested from Thermal Gradients," Journal of Intelligent Material Systems and Structures, vol. 18, no. 1, pp. 3-10, 2007

[19] D. W. Harrist. (2004) Wireless Battery Charging System Using Radio Frequency Energy Harvesting. Master's Thesis, University of Pittsburgh

[20] Y. K. Tan and S. K. Panda, "Energy Harvesting From Hybrid Indoor Ambient Light and Thermal Energy Sources for Enhanced Performance of Wireless Sensor Nodes," IEEE Transactions on Industrial Electronics, vol. 58 , no. 9, pp. 4424-4435, Sep. 2011

[21] A. Hande, T. Polk, W. Walker, and D. Bhatia, "Indoor solar energy harvesting for sensor network router nodes," Microprocessors and Microsystems, vol. 31, no. 6, pp. 420--432, Sep. 2007

[22] Maria Gorlatova, Peter Kinget, Ioannis Kymissis, Dan Rubenstein, Xiaodong Wang, Gil Zussman, “Challenge: Ultra-Low-Power Energy-Harvesting Active Networked Tags (EnHANTs)”, Mobicom 2009

[23] Maria Gorlatova, Zainab Noorbhaiwala, Abraham Skolnik, John Sarik, Michael Zapas, Marcin Szczodrak, Jiasi Chen, Luca Carloni, Peter Kinget, Ioannis Kymissis, Dan Rubenstein, Gil Zussman, "Prototyping Energy Harvesting Active Networked Tags (EnHANTs) with MICA2 Motes ," in SECON 2010

[24] R. Zollinger. (2012, May) inDOOR Energy Harvester. [Online]. http://itp.nyu.edu/sigs/sustainable/indoor-energy-harvesting

[25] Dexin Li, Pai H. Chou, “Maximizing efficiency of solar-powered systems by load matching”, ISLPED 2004

[26] Jaein Jeong, “A Practical Theory of Micro-Solar Power Sensor Networks”, PhD thesis dissertation, UC Berkeley, 2009

[27] A. Krause, C. Guestrin, and J. K. A. Gupta, "Near-optimal Sensor Placements: Maximizing Information while Minimizing Communication Cost," in IPSN 2006

[28] D. Ganesan, R. Cristescu, and B. Beferull-Lozano, "Power-efficient sensor placement and transmission structure for data gathering under distortion constraints," ACM Transaction on Sensor Networks, vol. 2, no. 2, pp. 155--181, May 2006

[29] P. Cheng, C.-N. Chuah, and X. Liu, "Energy-aware node placement in wireless sensor networks," in IEEE Global Telecommunications Conference, GLOBECOM 2004

[30] M J Handy, M Haase, D Timmermann, Low Energy Adaptive Clustering Hierarchy withDeterministic Cluster-Head Selection”, 4th International Workshop on Mobile and Wireless Communications Network (2002)

[31] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan “Energy-Efficient Communication Protocol for Wireless Microsensor Networks”, in the Proceedings of the Hawaii International Conference on System Sciences, January 4-7, 2000

[32] Samuel R. Madden, Michael J. Franklin, and Joseph M. Hellerstein, Wei Hong, “TinyDB: An Acquisitional Query Processing System for Sensor Networks”, ACM Trans. Database System, 2005, volume 30

[33] K. Langendoen, “Medium access control in wireless sensor networks”, in H. Wu and Y. Pan, editors, Medium Access Control in Wireless Networks. Nova Science Publishers, Inc., 2008

[34] Christophe J. Merlin and Wendi B. Heinzelman, “Duty Cycle Control for Low-Power-Listening MAC Protocols”, IEEE Transactions on Mobile Computing, 2010

[35] Q. Han, S. Mehrotra, N. Venkatasubramanian, "Energy Efficient Data Collection in Distributed Sensor Environments," in ICDCS, 2003.

[36] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan, “ Adaptive duty cycling for energy harvesting systems”, ISLPED 2006

[37] Recas, J., C. Bergonzini, D. Atienza, and T. Simunic, “Prediction and Management in Energy Harvested Wireless Sensor Nodes”, Wireless VITAE 2009

[38] D. K. Noh, L. Wang, Y. Yang, H. K. Le, and T. Abdelzaher, "Minimum Variance Energy Allocation for a Solar-Powered Sensor System," in Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems, Marina del Rey, CA, USA, 2009, pp. 44—57

[39] Sharma, N.; Gummeson, J.; Irwin, D.; Shenoy, P., "Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems", SECON 2007

[40] C. Renner and V. Turau, "Adaptive Energy Harvest Profiling to Enhance Depletion-Safe Operation and Efficient Task Scheduling", in Journal of Sutainable Computing: Informatics and Systems, Volume 2, Issue 1, March 2012

[41] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan, "Harvesting aware power management for sensor networks", in DAC 2006

[42] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy ," in 18th Euromicro Conference on Real-Time Systems, 2006. , 2006, pp. 10-270

[43] C. Moser , D. Brunelli , L. Thiele , L. Benini, "Lazy Scheduling for Energy Harvesting Sensor Nodes", in DIPES 2006

[44] Shaobo Liu, Qing Wu, and Qinru Qiu, "An adaptive scheduling and voltage/ frequency selection algorithm for real-time energy harvesting systems", in DAC 2009

[45] Shaobo Liu, Qinru Qiu, and Qing Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting", in DATE 2008

[46] J. B. Steck and T. S. Rosing, "Adapting task utility in externally triggered energy harvesting wireless sensing systems ," in Sixth International Conference on Networked Sensing Systems (INSS), 2009, Pittsburgh, PA , 2009, pp. 1-8

[47] A. Ravinagarajan, D. Dondi, and T. S. Rosing, "DVFS based task scheduling in a harvesting WSN for structural health monitoring," in DATE 2010

[48] C. Moser, J.-J. Chen, and L. Thiele, "Reward Maximization for Embedded Systems with Renewable Energies ," in RTCSA 2008

[49] C. Moser, J.-J. Chen, and L. Thiele, "Power management in energy harvesting embedded systems with discrete service levels," in ISLPED 2009

[50] C. Moser, J.-J. Chen, and L. Thiele, "Optimal service level allocation in environmentally powered embedded systems," in Proceedings of the 2009 ACM symposium on Applied Computing, 2009

[51] T. Voigt, A. Dunkels, J. Alonso, H. Ritter, and J. Schiller, "Solar-aware clustering in wireless sensor networks," in ISCC, 2004.

[52] J. Islam, M. Islam, and N. Islam, "A-sLEACH: An advanced Solar Aware Leach Protocol for Energy Efficient Routing in Wireless Sensor Networks," in ICN 2007

[53] E. Lattanzi, E. Regini, A. Acquaviva, and A. Bogliolo, "Energetic sustainability of routing algorithms for energy-harvesting wireless sensor networks," Computing Communication, vol. 30, no. 14-15, pp. 2976--2986, Oct. 2007.

[54] L. Lin, N. B. Shroff, and R. Srikant, "Asymptotically Optimal Energy-Aware Routing for Multihop Wireless Networks With Renewable Energy Sources," IEEE/ACM Transactions on Networking, vol. 15, no. 5, pp. 1021-1034, Oct. 2007

[55] L. Lin, N. B. Shroff, and R. Srikant, "Energy-aware routing in sensor networks: A large system approach," Ad Hoc Network, vol. 5, no. 6, pp. 818--831, Aug. 2007

[56] K. Zeng, K. Ren, W. Lou, and P. J. Moran, "Energy-aware geographic routing in lossy wireless sensor networks with environmental energy supply," in QShine '06, Waterloo, Ontario, Canada, 2006

[57] D. Hasenfratz, A. Meier, C. Moser, J.-J. Chen, and L. Thiele, "Analysis, Comparison, and Optimization of Routing Protocols for Energy Harvesting Wireless Sensor Networks ," in IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010

[58] M. K. Jakobsen, J. Madsen, and M. R. Hansen, "DEHAR: A distributed energy harvesting aware routing algorithm for ad-hoc multi-hop wireless sensor networks," in WoWMoM 2010

[59] A. Kansal and M. B. Srivastava, "An Environmental Energy Harvesting Framework for Sensor Networks," in ISLPED '03, Seoul, Korea, 2003, pp. 481--486

[60] K.-W. Fan, Z. Zheng, and P. Sinha, "Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks," in SenSys 2008

[61] Bo Zhang, Robert Simon, and Hakan Aydin, " Maximum utility rate allocation for energy harvesting wireless sensor networks" in Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems (MSWiM '11).

[62] L. Wang, et al., "AdaptSens: An Adaptive Data Collection and Storage Service for Solar-Powered Sensor networks," in 30th IEEE Real-Time Systems Symposium, 2009, RTSS 2009., Washington, DC, 2009, pp. 303-312

[63] James Reserver Data Management Systems. [Online]. http://dms.jamesreserve.edu/

[64] Liu, Yongpan, et al. "Thermal vs energy optimization for dvfs-enabled processors in embedded systems." In 8th International Symposium on Quality Electronic Design, 2007 (ISQED'07)

Chapter 3 Quality-aware Energy Management Framework for Harvesting Wireless Sensor Networks

In energy harvesting systems, energy is derived from environmental sources such as sunlight, wind and heat. Renewable energy sources allow us to continuously harvest energy from the environment, providing a constant and perpetual source of energy to the systems they drive. Despite its low efficiency, renewable energy technology is a viable and promising solution for low power wireless sensor network systems. Scavenging energy could enable smart sensors to be functional indefinitely and as a result, eliminating the cost for battery, enabling sustainable and manageable infrastructures.

Nevertheless, renewable energy depends heavily on environmental conditions (e.g., harvested solar energy on a sunny day is much higher than it is on a cloudy or rainy day). The time-varying characteristics of renewable energy sources create a shift in research focus from energy-efficient to energy-neutral approaches, i.e., from optimizing energy consumption to adapting systems to deal with unstable energy sources while meeting application quality constraints. Renewable energy sources such as solar energy show predictable patterns that are exploited in several coarse-grain (or slot-based) energy harvesting prediction methodologies [4, 9, 15]. Energy harvesting prediction is utilized for planning activities of nodes in a sensor network to keep them functional, yet still powered on.

Wireless sensor networks (WSN) are deployed in infrastructures, such as buildings or bridges and enable various data collection applications (e.g., structural monitoring). Sensors collect information about their surrounding environment, update a base station and respond to frequent or sporadic monitoring requests [14]. The base station stores such information in a cache and the bounded difference between the cached values and the actual instantaneous data values at the sensors is called the error margin. The energy cost of data collection applications relates heavily to the frequency of data requests and updates between sensors and the base station, which in turn affects accuracy of the collected data or the error margin. In systems with energy harvesting capabilities, I envision that sensors only communicate when there is sufficient harvested energy. There is therefore, a tight coupling between the ability of the system to harvest energy and the consequent data accuracy – intuitively better harvesting leads to better data quality whereas poor harvesting conditions imply loss of accuracy.

I propose an energy harvesting management framework called QuARES for data collection applications in wireless sensor networks. To the best of our knowledge, this work is the first attempt to jointly use both application data quality (expressed as error margins) and harvesting ability to manage the energy budget of such systems. Our framework includes two stages: an offline slot-based energy budget allocation algorithm and an online adaptation strategy. The offline stage exploits the slot-based harvested energy prediction and the relation between energy cost and data accuracy to allocate energy budget for each time slot in a given harvesting period (e.g., one day, one week). In the online stage, an online adaptation policy is proposed to guarantee timely responses to queries in spite of the time-varying characteristics of harvested energy.

Our contribution includes: (1) exploiting application tolerance to quality degradation to adapt the sensor data collection process under unstable energy harvesting conditions; (2) design of the QuARES framework, an energy harvesting management framework with 2 stages (online and offline) that utilizes energy harvesting prediction and knowledge of application tolerance–energy cost to maintain system sustainability and optimize data quality; (3) performance evaluation of the QuARES management framework as compared to other offline/online strategies (fixed-error-margin, minimum variance[9]) under different application and energy harvesting scenarios using the QualNet simulator (here, the battery model was modified to simulate energy harvesting) considering different sensor inputs, application constraints, weather conditions and battery capacities; (4) implementation, deployment and measurement in a real-world campus testbed, Responsphere at UCI [11]. Our simulator is also a valuable tool for designers to tune system parameters, to check feasibility of application constraints under various energy harvesting conditions and to study system performance. Experimental Results show that our framework can tolerate lower error margins (i.e. higher data accuracy) of *30-70%*, ensure a response to all queries; additionally, sensors do not have to shut down to replenish. Results also support the fact that while offline stage is needed to plan the energy budget to tolerate lower error margins, the online adaption is required for responsiveness to queries.

3.1 Related Work

Recent research has enabled wireless sensor motes to harvest energy from the surrounding environments [1-3]. Energy-neutral approaches have been proposed to cope with the time-varying characteristics of energy harvesting profile. Most of these approaches are cross-layer, considering energy status at battery layer and adapting system at other layers. Hsu et al. [4] adapt duty cycling of systems to the changes in renewable sources. Hasenfratz et al. [10] modify routing protocol at MAC layer to exploit both temporal and spatial variations of renewable energy and maximize data delivery rate for sensors. Voigt et al. [7] adapt LEACH, a cluster-based routing protocol for sensor networks to take advantage of energy harvesting. At operating system layer, Liu et al. [5] and Moser et al. [6] propose task scheduling techniques for energy harvesting systems. At the application layer, Noh et al. [8] use an adaptive technique to turn on and off storage services based on different energy thresholds. Ravinagarajan et al. [12] adapt task utility of structural health monitoring applications to maximize accuracy of tasks.

Some approaches exploit patterns in renewable energy profile to predict future harvested energy and to plan energy budget accordingly. Wang et al. [9] propose a minimum variance slot-based energy budget allocation for systems which prefer steady level of operation. This solution is not suitable for systems whose level of operation is dictated by application and user constraints that vary. Moser et al. [13] allocate energy budget for time slots to maximize quality of service for general systems. However, given the fluctuation of harvested energy within each time slot, an offline budget allocation methodology cannot always guarantee the desired quality of service.

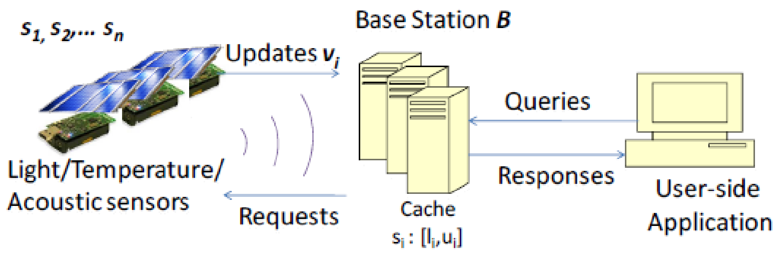


Figure 3.1 Data Collection in Wireless Sensor Networks

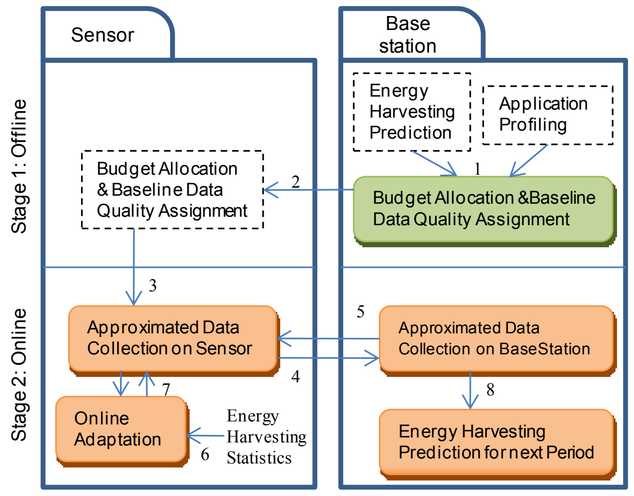
None of these works consider data collection applications with data accuracy tolerance, i.e. those that admit flexibility of error. [21] and [22] attempt to optimize data rate for data collection in wireless sensor networks without considering application quality needs as well as application's tolerance to data quality degradation. In [14], Han et al. propose an adaptive data collection protocol which is aware of these data accuracy requirements and exploits error margin of many applications to minimize energy consumption and prolong battery life-time. This approach is designed for and suited to battery powered sensor systems. Our work, on the other hand, exploits error tolerance in both offline and online stages to adapt the system to the fluctuations of renewable energy sources.

3.2. Data Quality in Energy Harvesting Systems

Our system consists of a wireless sensor network (WSN) deployed in an infrastructure for monitoring purposes. The components of our systems are a set of n sensor nodes {s_1, s_2,...,s_n} and a base station(s) *B* as shown in Figure 3.1. Each sensor s_i collects information about its surrounding environment by reading value vi from its embedded sensor and periodically sends an update to the base station(s). v_i is a property of the environment, e.g., temperature, humidity or sound, that the application needs to monitor through our WSN. In this study, I assume that all sensor nodes are equipped with a harvesting circuitry. Harvested energy is accumulated in an energy buffer that supplies power for sensor nodes' operation.

A base station B resides at a node with unlimited resources, e.g., power, storage, computation. It collects data from sensors and stores them in a cache. The cache contains an approximated value u_i for each sensor s_i's true value v_i. Base station B is connected to a monitoring application on the user side. The application periodically polls sensor nodes through the base station(s) for the monitored information. When necessary, the application can ask for sporadic information. In particular, the application sends a query Q_j to the base station each time it needs data from a sensor node or a set of sensor nodes. The query Q_j contains data accuracy constraints specified as error margin. If the approximated value u_i for a sensor s_i satisfies these constraints, the base station B returns u_i to the monitoring application. Otherwise, B sends an update request to the sensor s_i, to retrieve the latest value v_i, and replies to the query Q_j with the updated approximated value.

Data accuracy can be expressed as an error margin of the actual value v_i, e.g., $v_i \pm 10$ or $v_i \pm 10\%$. Such error tolerance can be exploited while tuning the system. Error margins can be increased or decreased to meet both data accuracy constraints and system constraints, such as varying energy supply. Our energy harvesting management framework exploits this error tolerance to adapt the system to the availability of renewable energy sources. However, there are several challenges in managing such dynamic systems as outlined in the next section.



Sensor
Base station
Stage 1: Offline
Stage 2: Online
Energy Harvesting Prediction
Application Profiling
1
Budget Allocation &Baseline Data Quality Assignment
2
Budget Allocation & Baseline Data Quality Assignment
3
Approximated Data Collection on Sensor
4
5
Approximated Data Collection on BaseStation
7
8
Online Adaptation
6
Energy Harvesting Statistics
Energy Harvesting Prediction for next Period

Figure 3.2 QuARES Framework

3.2.1 Energy Harvesting Systems

Exploiting renewable energy patterns, such as daily and seasonal patterns, several prediction algorithms (see [4], [9], [15], [16]) have been proposed to predict future harvested energy. Prediction information is composed of the predicted total energy harvested in each time slot in a harvesting period rather than a series of instantaneous values. Typically, systems keep track of the average rate of energy harvesting, they do not maintain specific statistics on variations in energy harvesting as a function of time.

Furthermore, in energy harvesting systems, dynamicity is not only from the fluctuation in energy supply but also from the changes in input data and application quality constraints. To make the application demand match the energy supply is a critical task for system sustainability, realizing additional criteria such as the optimization of application quality, specifically data accuracy under these constraints, is even more challenging.

The first challenge is to utilize the prediction information about future harvested energy to sustain the system and maximize overall data accuracy. If high data accuracy is assigned to an interval with predicted low energy, the energy supply will not meet the energy demand and the system might run out of battery and shut down, and as a result suspend monitoring activities. If low data accuracy is assigned to an interval with predicted high energy, the harvested energy is not utilized and might be wasted due to energy overflow. The second challenge is to adapt the assigned data accuracy to the actual rate of harvested energy that the system perceives at run-time. If the energy in the buffer is low and the harvested energy rate is lower than the predicted average rate, the system will not have enough energy to sustain itself, let alone maintain the assigned data accuracy.

A high level definition of the problem is as follows: Given predictions of energy harvesting and knowledge of error tolerance and energy cost, the goal is to guarantee maximum system lifetime while optimizing data accuracy.

3.2.2 Proposed Framework - QuARES

I propose a framework called QuARES – Quality Aware Renewable Energy System to address this problem. QuARES is a cross-layer energy management framework consisting of *2* stages, one offline stage and one online stage. Each stage addresses a challenge in section 3.2.1 respectively. Figure 3.2 depicts our framework and its stages.

Stage 1 is executed offline in the current harvesting period. In this stage, the base station runs a prediction algorithm to extrapolate information about energy harvested in the next harvesting period. The predicted information and the knowledge about data accuracy and energy cost are inputs to an optimizing algorithm (step 1, Fig. 3.2). This algorithm allocates energy budget for each time slot in the next period and optimize overall data accuracy. The results are then sent to sensor nodes (step 2) and each sensor node stores the energy budget in its memory.

Stage 2 is executed online in the subsequent harvesting period. At the beginning of each time slot, the corresponding energy budget is retrieved from memory and a corresponding baseline error margin is looked up (step 3). Sensor nodes and the base station exchange messages according to their protocol to maintain the error margin (steps 4 & 5). Online adaptive policies monitor the energy buffer and the actual harvested energy (step 6) to adjust the baseline error margin (step 7). Adaptation allows the system to cope with variation in renewable energy source and maintain system sustainability. In addition, energy harvesting statistics are sent to the base station (together with messages in step 4) to update energy harvesting prediction of the next harvesting period (step 8).

3.3. PROBLEM FORMULATION

In this section, I define system parameters, energy harvesting and data accuracy models. Next, I describe the algorithms used in each stage of our framework. Each sensor has a battery, with capacity C, to store harvested energy. Energy accumulated beyond capacity C will be discarded (energy overflow). Let $E_{initial}^{0}$ be the available energy at the beginning of the harvesting period. Let E_{min} be the minimum energy to be reserved at the end of the harvesting period.

Energy Harvesting Model - I denote the length of a harvesting period as T. A harvesting period is then divided into equal-length intervals or time slots. Let N be the number of slots in a harvesting period T. The values of T and N are defined by the harvested energy prediction algorithm on the basis of renewable energy source, the nature of sensor input and query model. Our framework, however, is independent of these parameters. For each time slot i, the prediction algorithm provides $E_{harvest}^{i}$ which is the amount of harvested energy in that slot. I assume an ideal slot based harvested energy prediction algorithm.

Data Accuracy Model - I model data accuracy in terms of error margin. Error margin δ_i is the bounded difference between the sensed value v_i at the sensor node s_i and the approximated value u_i at the base station B, such that $|u_i - v_i| \leq \delta_i$. The entry in the base station's cache is not a single value but an approximation range $[l_i, u_i]$ where l_i is the lower bound and u_i is the upper bound for sensed value v_i and $u_i - l_i = 2\delta_i$. This error margin is the constraint of the application and is not the measurement error or sensitivity of physical sensors. I provide the following definitions:

Definition 1: Baseline Average Error margin denoted as $\overline{\delta_{\iota,base}}$ is the average of baseline error margin predicted in the offline stage, representing the predicted average data accuracy of data collected from sensor s_i.

Definition 2: Actual Average Error Margin denoted as $\overline{\delta_\iota}$ is the average error margin maintained by sensor s_i in a harvesting period T, representing the actual average data accuracy of data collected from sensor s_i.

Our framework, QuARES, optimizes $\overline{\delta_{\iota,base}}$ in the offline stage and uses online adaptation to maintain $\overline{\delta_\iota}$ close to $\overline{\delta_{\iota,base}}$ during online stage.

Definition 3: Consistent state refers to the state of cached entry on the base station B. If the approximation range *[l_i, u_i]* satisfies $u_i - l_i = 2\delta_i$ and sensed value v_i satisfies $l_i \leq v_i \leq u_i$, the cache entry is in a consistent state.

Definition 4: Inconsistent state refers to the state of cached entry on the base station B when v_i falls outside the approximation range *[l_i, u_i]* or when $u_i - l_i \neq 2\delta_i$.

When sensor node s_i reads a new value v_i, it checks if the cache is still consistent. If the cache is inconsistent, a new approximation range $[v_i - \delta_i, v_i + \delta_i]$ is sent to the base station to update the cache. This process is called source update. If the sensor node does not update the base station, e.g., running out of battery, the cache entry is in inconsistent state. The number of source updates essentially reflects the physical characteristics of monitored phenomenon. I assume the sampling rate of sensors is sufficient to detect changes in the monitored environment and time to send update message is negligible.

When the base station B receives a new query Q_j, it first checks if the current cache entry satisfies the data accuracy constraint A_j, i.e., $\delta_i \leq A_J$. If it is satisfied, the base station immediately responses to the query with value $\frac{(u_i+l_i)}{2}$. Otherwise, the base station sends a request to s_i for current sensed value v_i. The sensor node replies with the updated approximation range $[v_i - \delta_i, v_i + \delta_i]$, the base station updates its cache entry and sends v_i to the application. This process is called consumer update. The number of consumer updates reflects the nature of query model both in term of query frequency and associated accuracy constraints.

The smaller the error margin, the more source updates and less consumer updates. On the other hand, the larger the error margin, the less source updates and more consumer updates. The two extremes are $\delta = 0$ (only source updates) and $\delta = \infty$ (only consumer updates). However, considering both types of updates, Olston et al. [17] find an error margin δ_{max} where the total number of updates and energy consumption is minimum. Hence, we only need to consider error margin in the range $[0, \delta_{max}]$ as beyond this, both the error margin and energy consumption is higher.

I assume to have a vector $(\delta_1 \ldots \delta_K)$ and a corresponding vector $(E_{cost}^1 \ldots \delta_{cost}^K)$ where E_{cost}^j is the energy required to maintain error margin δ_j in a time slot. K is the number of error margin levels. This function from error margin to energy consumption is an abstraction of the relation between application quality and the nature of monitored physical phenomenon and query behavior.

Problem Formulation

Given input:

Harvesting period T, N time slots

Battery capacity $C \leq 0$ and Initial battery $E_{initial}^0$

Minimum battery remained after T: E_{min}

Energy harvesting prediction of each time slot $E_{harvest}^i$ $\quad 0 \leq i < N$

Error margin and energy cost vectors: $(\delta_1 \ldots \delta_K)$ and $(E_{cost}^1 \ldots \delta_{cost}^K)$

Objective: Minimize $\bar{\delta}$, the average actual error margin, in a harvesting period T

Figure 3.3 Problem Formulation

A more formal characterization/formulation of the problem is given in Fig. 3.3 where our goal is to minimize the actual average error margin. In the next two sections, I describe our algorithms in each stage to achieve this.

3.3.1 Stage 1: Budget Allocation and Baseline Data Quality Assignment

In stage 1, I solve a linear optimization problem (see Algorithm 3-1) to allocate energy budget and assign a corresponding baseline error margin for each time slot. The linear optimization problem for each sensor node is solved by a linear solver at the base station. System parameters, harvested energy prediction and data accuracy – energy cost in Fig. 3.3 are input to this optimization problem.

Let $E^{i}_{initial}$denote the energy in the battery at the beginning of time slot *i*. Since the battery cannot store more than its capacity C, $E^{i}_{initial}$ must be constrained by this upper bound (constraint 1, Algorithm 3-1). Any excess energy is discarded (energy overflow). Let E^{i}_{budget}denote the energy budget for slot *i* which could only be drawn from the available energy in the battery at the beginning of the slot and the energy harvested during this slot (constraint 2). Vice versa, the energy in the battery at the beginning of a slot is limited by the amount of energy available in the previous slot subtracted by its energy consumption (constraint 3).

The base line error margin for each slot is assigned based on the allocated energy budget. Let *q[i,j]= 1* if error margin δ_j is assigned to slot *i*, *q[i,j] = 0* otherwise. Each slot *i* could only be assigned one base line error margin (constraint 4) and this error margin is computed in constraint

5. In addition, its energy budget must be sufficient to maintain this error margin, i.e., at least E_{cost}^{j} (constraint 6). Let δ_{max}^{i} be the maximum tolerated error margin of the application in slot i and also the upper bound for the assigned base line error margin of slot i (constraint 7). Many applications could have time-based quality constraints such as monitoring closely during day-time than night-time or vice versa. Under this constraint, the QuARES framework makes sure the data accuracy is never degraded beyond application needs.

Algorithm 3-1: Offline planning

```
Stage 1 at Base Station B
1. Run a linear solver to solve this linear programming for each sensor s
Objective        Minimize $\overline{\delta_{base}} = \frac{\sum_{i=0}^{N-1} \delta_{base}^{i}}{N}$
Subject to constraints:
   Input and Constraints in Figure 3.3
   $E_{initial}^{i} \le C$                                                  $\forall i: 0 \le i < N$        (1)
   $E_{budget}^{i} \le E_{initial}^{i} + E_{harvest}^{i}$                              $\forall i: 0 \le i < N$        (2)
   $E_{initial}^{i+1} \le E_{initial}^{i} + E_{harvest}^{i} - E_{budget}^{i}$          $\forall i: 0 \le i < N$        (3)
   $\sum_{j=0}^{K-1} q[i,j] = 1$                                  $\forall i: 0 \le i < N$        (4)
   $\sum_{j=0}^{K-1} q[i,j] \times \delta_j = \delta_{base}^{i}$                   $\forall i: 0 \le i < N$        (5)
   $\sum_{j=0}^{K-1} q[i,j] \times E_{cost}^{j} \le E_{budget}^{i}$                $\forall i: 0 \le i < N$        (6)
   $\delta_{base}^{i} \le \delta_{max}^{i}$                                     $\forall i: 0 \le i < N$        (7)
2. Send energy budget allocation and baseline data quality assignment to sensor $s$
Stage 1 at Sensor node s
1. Receive energy budget allocation and baseline data quality from base station
2. Save in memory for run-time use in the next harvesting period
```

The objective of this optimization problem is to maximize the average base line error margin, δ_{base}. Next, I describe stage 2 of our framework, online dynamic adaptation whose task is to maintain the assigned baseline error margin and guarantee continuous system operation.

3.3.2 Stage 2: Online Dynamic Adaptation

During stage 2, data collection protocol runs on both sensor and base station to keep the cache in a consistent state and respond to monitoring queries. However, the actual harvested energy rate could be lower than the predicted average rate. The system thus needs online adaptation policies to tackle energy supply fluctuations, maintain system operation and data accuracy constraints.

Our online adaption running on sensor node is a heuristic which keeps track of current harvesting rate and battery status to adjust the error margin, guarantee system operation and consistency of cached entry at the base station. I develop *2* dynamic adaptation policies: inter-frame adaptation and intra-frame adaption (see Algorithm 3-2).

1) Inter-frame adaption is triggered at the beginning of each time slot. The harvested energy often does not come at a constant rate; when harvested energy is abundant and the energy buffer is almost full, energy overflow happens. The harvested energy thus could be less than what expected and the system needs to adapt its energy budget plan and base line error margin. The inter-frame policy keeps track of this energy discrepancy and distributes the energy offset among current and future slots' energy budget. Our inter-frame adaptation algorithm is summarized in Policy 1, Algorithm 3-2.

```
Algorithm 3-2: Online Adaptation
Policy 1: Inter-frame online adaptation (slot i)
1. buffer   = current energy in the buffer
2. offset   = $E^{i}_{initial}$ - buffer
2. if (offset > epsilon ) then     #adjust budget of future slots
3.        $E^{i}_{budget} = E^{i}_{budget} - offset$
4.        j = find_quality_level($E^{i}_{budget}$)
5.        $\delta = \delta_j$
6.        update_server($s_i, v - \delta, v + \delta$)
Policy 2: Intra-frame online adaptation (slot i)
1. h  = current_harvesting_rate;
2. if (|h - $h_{old}$| < epsilon) then return
3. else $h_{old}$  = h
4. buffer   = current energy in the buffer
5. reserve = buffer energy reserved for future sub-slots
6. l = length of a sub_slot
7. supply =  h*l + (buffer-reserve)     #energy supply for this sub-slot
8. if (supply < $E^{i}_{budget}$ /number_of_subslot  )
9.     then     $\delta$  = find_quality_level (supply*number of subslot)
10.   else     $\delta = \delta^{i}_{base}$
11. if $\delta$ changes then
12.        update_server($s_i, v - \delta, v + \delta$)
```

2) Intra-frame adaptation on the other hand is triggered more often every sub-slot within a time slot to quickly adapt to fluctuations of the renewable energy source. The length of a sub-slot depends on the energy source, which is the interval of time that the harvested energy rate remains fairly stable, e.g. *1-5* minutes. Every sub-slot, intra-frame policy will check if the current harvesting

rate is significantly less than the predicted average harvesting rate and adjust error margin in the current sub-slot accordingly. In the next sub-slots, if the harvesting rate increases above the expected average rate, the policy restores the baseline error margin. Our inter-frame adaptation algorithm is summarized in Policy 2, Algorithm 3-2.

3.4. Evaluation

In this section, I first explain the experimental setup to evaluate the effectiveness of our proposed framework QuARES and then I compare the results with other existing policies in terms of error margin (data accuracy), responsiveness to queries, and energy consumption.

3.4.1 Experimental Setup

I initially implemented the approximated data collection application and QuARES framework in QualNet network simulator [18]. The simulator is configured to simulate a sensor network of Mica motes with ZigBee standard specification. Power consumption of sensor node is set accordingly to Mica-2 [19]. I am also developing a prototype test bed with harvesting capability (see section 3.6).

Sensor data are generated randomly from the range *[-150, 150]*. The sampling rate is *100Hz*, each sampling either increases or decreases the previous value by an amount randomly chosen in the range *[0.5, 1.5]*. Figure 3.4b gives an example of randomly generated sensor data for simulated time of *6* hours. Periodic queries arrive every *100 ms*. Sporadic queries are modeled by Poisson distribution with mean interval = *100 ms*. Each query is associated with an error tolerance of mean = *20* and deviation = *1*.

Energy harvesting profile is retrieved from National Renewable Energy Lab website [20]. Solar profiling for a day is shown in Figure 3.4a. The data is average solar irradiance (mW/m^2) at a specific location every *5* minutes. The irradiance is converted to harvested energy by linear conversion considering solar panel size *9.6cmx6.4cm*, solar cell efficiency *10%* and harvesting efficiency *80%*. I modify QualNet battery model to charge battery every *1* minute. I assume a perfect solar energy prediction algorithm which gives accurate slot-based prediction to our offline stage. I choose $T = 1$ day and $N = 48$ slots.

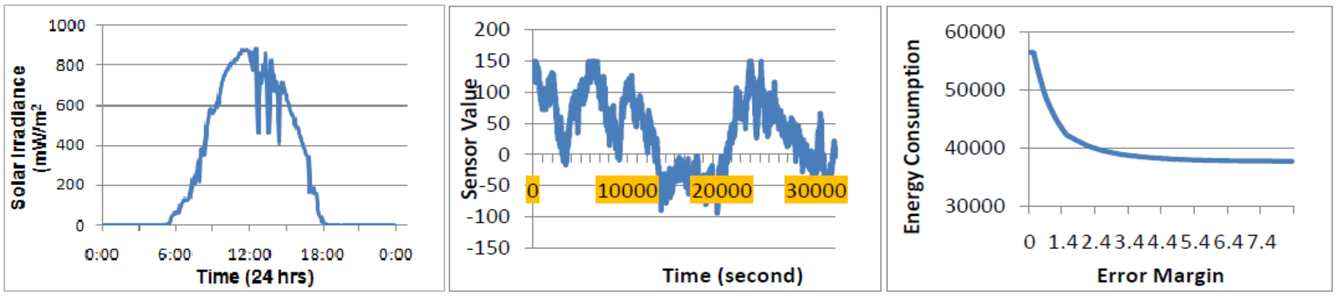


Figure 3.4 Solar Data, Sensor data, and Data Quality

I profile error margin vs. energy cost by simulating application for different error margins in the range $[0, \delta_{max}]$ (see Figure 3.4c). Since it is impossible to profile every value in this range, I choose to profile values at a constant interval step, $r = 0.1$. For each error margin $\delta_{profile}$, I fix the baseline error margin for all slots (δ_{base}=$\delta_{profile}$) and set energy buffer always full. I run the simulation for a simulated time of one day and divide the total energy consumption by the number of time slots

N to obtain average energy cost per slot at error margin $\delta_{profile}$. For the given query model and sensor input, I find that $\delta_{max} \approx 8.0$.

3.4.2 Experimental results

To evaluate the effectiveness of our proposed QuARES framework, I have implemented several offline or online policies for energy management during data collection. I evaluate the policies in terms of their data accuracy (error margin), system sustainability, and energy consumption. These implemented policies are as follows:

- FIX_ERROR ($\delta = 8.0$) and FIX_ERROR ($\delta = 0.5$): FIX_ERROR is an offline policy and has no online adaptation. A fixed baseline error margin is assigned to all time slots.
- GREEDY_ADAPT: Greedy adaptation is a completely online protocol without offline energy budget assignment. It sets an error margin at the beginning of each time slot according to the amount of available energy in the buffer.
- MIN_VAR: Adopted from [9], it allocates energy budget for slots in the offline stage with minimum variance. Its goal is to maintain steady operation for the system. It does not have online adaptation.
- QuARES: As presented in this chapter, our quality-aware framework minimizes error margin both in offline and online stages.

Table 3.1 shows results of QuARES in comparison with other approaches. FIX_ERROR ($\delta = 8.0$) has a very high error margin and consumes minimum energy. Due to high error margin, source updates is very low compared to consumer updates; this scheme is thus very close to push-based data collection. The system responds to all the queries at the trade-off of a higher error margin. This is the extreme case where energy saving is a dominant requirement compared to data accuracy and is suitable for traditional battery powered systems [16]. The second column FIX_ERROR ($\delta = 0.5$) is another fixed error rate policy which attempts to maintain a lower error margin $\delta = 0.5$ by exploiting energy harvesting. Due to very low error margin, the source updates are very high compared to the consumer updates. This scheme thus is very close to pull-based data collection. However, it fails since the sensor node cannot maintain this low error margin when the harvested energy is low. The battery is exhausted and system needs to shut down for *45* minutes to replenish energy. This leads to a very high number of failed responses to queries. FIX_ERROR approach in general cannot work in dynamic energy harvesting systems.

Table 3.1 Comparison Results

	FIX_ERROR $(\delta = 8.0)$	**FIX_ERROR** $(\delta = 0.5)$	**GREEDY_ ADAPT**	**MIN_VAR**	**QuARES offline**	**QuARES offline + online**
Average Error Margin	8.00	0.50	0.348	0.388	0.156	0.159
Total Energy Consumption (J)	1813	2686	2656	2641	2641	2641
Shut down time for harvesting (minute)	0	45	21	7	4	0
Failed responses to queries	0	570	420	196	64	0

The third and the fourth column show results of GREEDY_ADAPT and MIN_VAR [9]. Both have comparable average error margin and energy consumption. However, neither of them consider fluctuation of energy harvesting in its greedy online adaptation or offline budget allocation. Therefore, in both cases, the sensor node runs out of battery and shuts down (only energy harvesting continues), leading to failed responses to queries. It is thus necessary to have online adaptation to handle fluctuation of renewable energy. In columns 5 and 6 of Table 3.1, I study the significance of each stage in the QuARES framework: offline budget allocation and online adaptation.

I compare our QuARES offline and the whole QuARES with both online and offline stages. As seen from Table 3.1, without online adaptation, QuARES (offline) must shut down the sensor node for *4* minutes and thus failed to response to *64* queries. The system often shuts down during the sunrise when both battery reserves and harvested energy rate is low.

Among all approaches, QuARES with both online and offline stages keeps the system alive for the whole harvesting period, responds to all queries, maintains low error margin from *30-70%,* and enables a uniform energy usage across nodes. The results show that while our proposed offline stage in QuARES maximize data accuracy, the online adaption stage is required for successful query responsiveness.

Varying Application Constraints: I evaluated our framework for different application constraint profiles. Application Constraint profile 1 (AC1) maintains low error margin during day time ($\delta < 1.5$), from *5am* to *7pm*.

Table 3.2 Comparison results

	App. Constr. 1		App. Constr. 2	
	MIN_VAR	**QuARES**	**MIN_VAR**	**QuARES**
Average Error Margin	3.09	0.891	1.067	0.783
Energy Cons. (J)	2481	2509	2481	2501
Shut down time (min.)	32	0	9	0
Failed responses to queries	521	0	227	0

Application Constraint profile 2 (AC2) maintains low error margin during night time, $\delta < 1.5$ from *7pm* to *5am*. QuARES (in Table 3.2) satisfies all the application constraints at the trade-off of suboptimal energy budget allocation compared to no constraint case, higher error margin and less energy utilization. Interestingly, the QuARES offline stage also gives immediate feedback to system designers if the given data accuracy constraints are infeasible in the next harvesting period and needed to be adjusted. MIN_VAR, on the other hand distributes energy budget among slot with minimum variance regardless of application constraints, thus do not give any feedback to designers when application constraints are infeasible. Furthermore, the budget is not in accordance with energy demand in different slots, MIN_VAR spends more energy in slots where it should save energy for higher-demand slots in the future and soon runs out of battery and fails to response to queries.

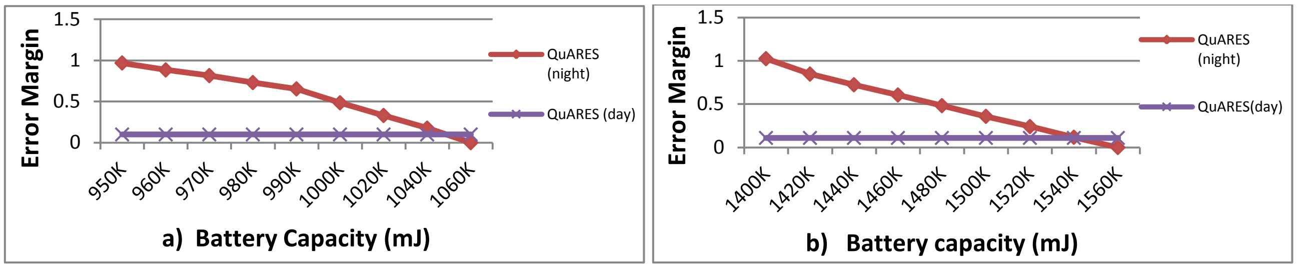


Figure 3.5 Impact of battery capacity on application data

Impact of Battery Capacity: I simulate the application with different battery capacities. Figure 3.5a shows average error margin during day time and night time under different battery capacities on a summer day. As seen from Figure 3.5 a, on one hand, the battery capacity has a negligible effect on the error margin during the day as the energy supply is abundant. On the other hand, the battery capacity has significant impact on the error margin during the night. There is no energy harvested during this period and the capacity of battery limits energy saving to maintain data accuracy at night. In addition, I compare results on summer days and winter days (Figure 3.5b). Results show that to obtain a comparable error margin, the battery capacity required on winter days is larger than the battery capacity on summer days as winter night time is longer than summer night time. For example, to achieve average error margin of *1.0* at night requires battery capacity *C = 950000 mJ* on a summer day but requires *C = 1400000 mJ* on a winter day.

3.5 Case Study – Indoor Energy Harvesting Systems in Sensorized Infrastructures

In this section, I carry out simulation with synthetic input from our study of energy harvesting system in a sensorized infrastructure. I first describe our study and then show our simulation result applying QuARES in this context.

In this study, I first measure energy harvesting availability in a building with a focus on light energy sources. I identify two sources of energy: light bulbs inside offices and hall ways and solar light from windows around the building. I experiment with different types of light bulbs and Table 3.3 shows the results of our measurement.

Table 3.3 Indoor Light Energy Harvesting

Type of bulb	**Energy Used (W)**	**Light Output**	**Harvesting power (mW)**
Daylight Complact Fluorescent	14	800	28.32
Bright White Compact Fluorescent	14	800	29.75
Round back light	Max. 55	N.A	31.09
Soft White Compact Fluorescent	14	900	35.15
Halogen light	55	825	66.48
Soft White Incandescent Bulb	57	780	148.97

The table shows various types of bulb used for lighting at offices and home. Our solar panel is placed in front of the light source at the distance of *10cm*. I record the harvesting power from the solar panel (Solar-made, *9.6cmx6.4cm*) in this table. I observe that the higher the energy a light bulb consumes, the higher the harvesting power it provides as the irradiance increase. Temperature also plays an important role as soft while incandescent bulb with same energy consumption but has higher temperature but also significant higher harvesting power. Among these types of bulbs, soft white compact fluorescent light bulb has a nice balance between its energy consumption and energy harvesting power supply.

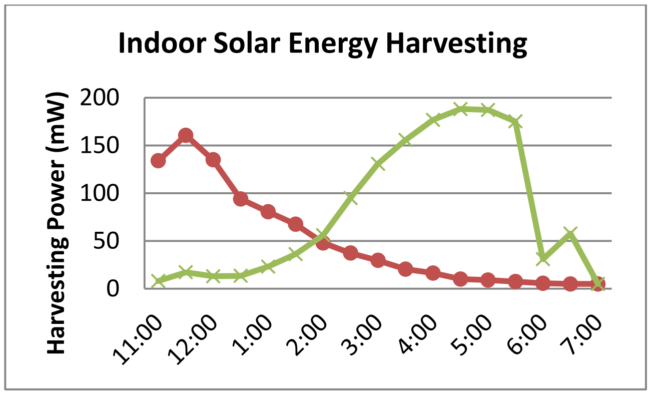


Figure 3.6 Indoor Solar Energy Harvesting Profile

In addition to indoor lighting bulb, another interesting energy harvesting source we find in a building is solar light filtered by glass windows. I place our solar panel on the inside of such windows and measure the output of the solar panel. Figure 3.6 shows the profile of energy harvesting at *2* windows at different locations of the building. The two energy harvesting profiles have a similar shape and peak point but one is shifted toward the right while the other to the left in the time domain. One peak point is at *11am* while the other is at *5pm*. The reason is the energy harvesting reaches its peak point when the angle of the sun is directly orthogonal to the solar panel which is in the morning for the window at the east side and in the afternoon for the window at the west side. It is well known that energy harvesting depends on solar irradiance, temperature as well as angle of the sun.

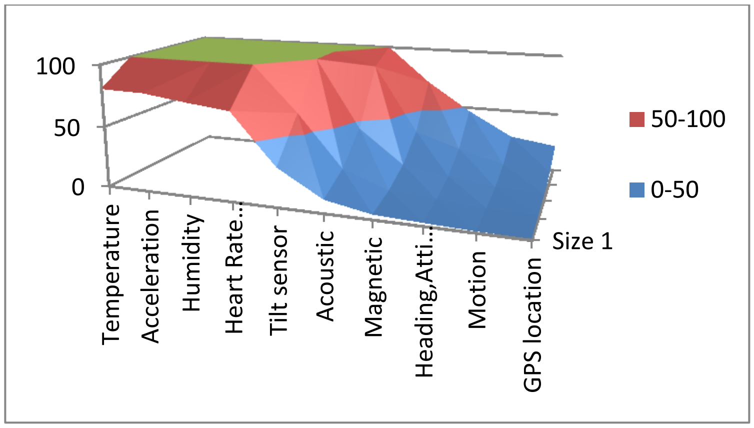


Figure 3.7 System Operation Time Estimation

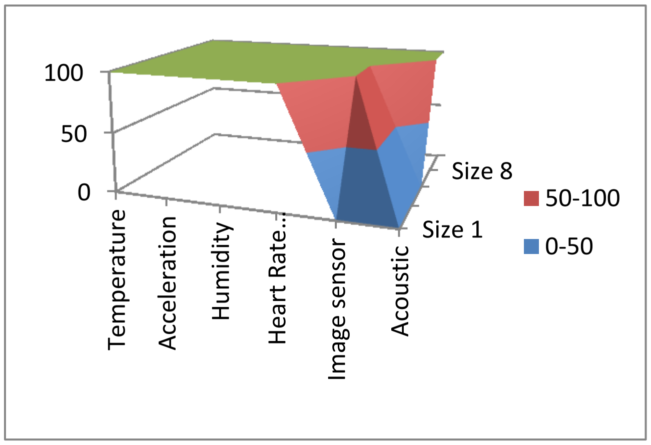
If we do an estimation of system operation time and apply this for different type of sensors on a Micaz2 sensor board and combined energy harvesting sources from one window and one soft white compact fluorescent bulb, we will have Figure 3.7 which estimates the duty cycle during which the system is active.

$$\text{system operation time} = \frac{\text{maximum energy consumption}}{\text{average energy harvesting}}$$

From this raw estimation, we may think that energy harvesting supply with *1* solar panel is not enough to sustain system operation even for low power sensor such as temperature and humidity. And it might need *16* of such solar panel to provide *80-100%* system runtime.

In fact, I run QuARES with synthetic input of combined energy harvesting sources from one window and one light bulb, the result is shown in Figure 3.8. With only *1* solar panel, the energy harvesting system is able to sustain *100% time*. The low-power image sensor needs *2* solar panels (as opposed to *4-8* solar panels in the raw estimation above). The acoustic sensor board needs *8* solar panels which is half of the worst case estimated size. In addition, QuARES also provide information about the battery capacity and data quality for the designer to explore the system design space. The summary of data quality for different type of sensor is given in Figure 3.9.

In this case study, we see that using QuARES we can design an energy harvesting system with smaller solar panel size and with a guarantee of data quality. System designers can also choose different battery size according to application need or requirement of data quality.



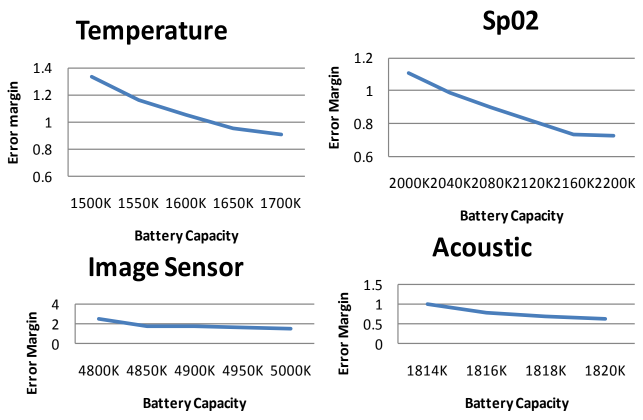


Figure 3.8 System Operation Time by QuARES

Figure 3.9 Data Quality for Different Types of Sensors

3.6 Prototype System and Test Bed

I am deploying a test bed of energy harvesting wireless sensor network in Responsphere [11]. Figure 3.10 shows our Responsphere infrastructure and prototype of our energy harvesting platform, including a Crossbow sensor temperature, light and acoustic sound), two solar panels and *2 AA* batteries. Table 3.4 shows our measurement at noon at *4* different locations inside and outside a building in Responsphere infrastructure (Floor Plan, Figure 3.10). Locations 1-2 are indoor while locations 3-4 are outdoor. Under artificial light condition (location 2), solar panels provide low but stable power to sensors. Under natural light (loc. 1 next to a window and loc. 3-4) solar panels provide higher but fluctuating power as it subjects to several conditions such as time of the day, surrounding objects, cloud or wind which may create or shift shadow or direction of windows for indoor case. I plan to use our measurement of solar profile and real sensor data to input to our network simulator. The simulator of data collection application and QuARES framework plays an important role in designing and tuning real system parameters, studying the feasibility of application constraints and energy harvesting as well as system performance.

Table 3.4 Measurement of Solar Panel Output at Various Locations in UCI

	Indoor (1: next to a window)	**Indoor (2: inside a room)**	**Outdoor (3: under shadow)**	**Outdoor (4: sunny location)**
Voltage (V)	16.24	7.17	15.2 - 15.7	17.2 - 17.7
Current (mA)	20.7	2.6	9 – 11	59.4 - 65.2
Availability	12 hours	8 hours	12 hours	12 hours

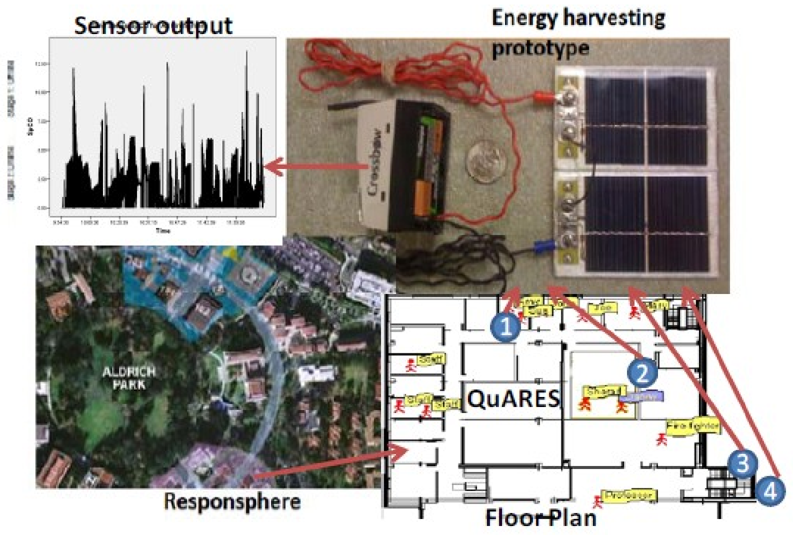


Figure 3.10 Responsphere Infrastructure and Solar Testbed

3.7. Conclusion

In conclusion, this chapter proposes a complete autonomous energy management framework in energy harvesting WSN whose goal is to optimize data accuracy for approximated data collection applications and sustain system operation. The offline stage explores energy harvesting prediction information to allocate energy budget among time slots in a harvesting period and maximize overall data accuracy. The online adaptation stage maintains the predicted data accuracy while coping with harvested energy fluctuation. Our framework is evaluated extensively in comparison with other approaches and considering different weather conditions, battery capacities and application constraints.

References

[1] K. Lin, J. Yu, J. Hsu, S. Zahedi, D. Lee, J. Friedman, A. Kansal, V. Raghunathan, and M. Srivastava, "Heliomote: Enabling Long-Lived Sensor Networks Through Solar Energy Harvesting," in SenSys, 2005.

[2] X. Jiang, J. Polastre, and D. Culler , "Perpetual Environmentally Powered Sensor Networks," in IPSN, 2005.

[3] F. Simjee and P. H. Chou, "Everlast: Long life, Supercapacitor operated Wireless Sensor Node," in ISLPED, 2006.

[4] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, V. Raghunathan, "Adaptive Duty Cycling for Energy Harvesting Systems," in ISLPED, 2006.

[5] S. Liu, Q. Wu, Q. Qiu, "An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems," in DAC, 2009.

[6] C. Moser, D. Brunelli, L. Thiele, L. Benini, "Lazy Scheduling for Energy Harvesting Sensor Nodes," in DIPES, 2006.

[7] T. Voigt, A. Dunkels, J. Alonso, H. Ritter, J. Schiller, "Solar-aware clustering in wireless sensor networks," in ISCC, 2004.

[8] D. K. Noh, L. Wang, Y. Yang, H. K. Le, and T. Abdelzaher, "Minimum Variance Energy Allocation for a Solar-Powered Sensor System," in DCOSS, 2009.

[9] L. Wang, D. K. Noh, Y. Yang, H. K. Le, T. F. Abdelzaher and M. Ward, "AdaptSens: An Adaptive Data Collection and Storage Service for Solar-Powered Sensor networks," in RTSS, 2009.

[10] D. Hasenfratz, A. Meier, C. Moser, J. J. Chen, and L. Thiele , "Analysis, Comparison, and Optimization of Routing Protocols for Energy Harvesting Wireless Sensor Networks," in SUTC, 2010.

[11] Responsphere infrastructure test bed, 17 March 2011, http://www.responsphere.org/index.php

[12] A. Ravinagarajan, D. Dondi and T. S. Rosing, "DVFS Based Task Scheduling in a Harvesting WSN for Structural Health Monitoring," in DATE, 2010.

[13] C. Moser, J. J. Chen, L. Thiele, "Power Management in Energy Harvesting Embedded Systems with Discrete Service Levels," in ISLPED, 2009.

[14] Q. Han, S. Mehrotra, N. Venkatasubramanian, "Energy Efficient Data Collection in Distributed Sensor Environments," in ICDCS, 2003.

[15] C. Bergonzini, B. Lee, J. R. Piorno, T. S. Rosing, "Management of Solar Harvested Energy in Actuation-based and Event-triggered Systems," in Energy Harvesting Workshop, 2009.

[16] M. Ali, B. Al-Hashimi, J. Recas and D. Atienza, "Evaluation and Design Exploration of Solar Harvested-Energy Prediction Algorithm," in DATE, 2010.

[17] C. Olston, B. T. Loo, and J. Widom, "Adaptive precision setting for cached approximate values," in ACM SIGMOD, 2001.

[18] QualNet network simulator. [Online]. "http://www.scalable-networks.com/products/QualNet/"

[19] Crossbow Technology INC. Mica-2 Data Sheet. [Online] Crossbow, http://www.xbow.com.

[20] National Renewable Energy Lab. [Online]. http://www.nrel.gov

[21] B. Zhang, R. Simon and H. Aydin,"Energy Management for Time-Critical Energy HarvestingWireless Sensor Networks" in SSS, 2010.

[22] K Fan, Z. Zheng and P. Sinha ,"Steady and Fair Rate Allocation for Rechargeable Sensors in Perpetual Sensor Networks" in SenSys, 2008.

Chapter 4 Quality-aware Energy Management for Real-time Systems with (m,k)-firm constraints

To cope with fluctuations in the renewable energy sources, energy harvesting embedded systems need to deploy various schemes to adapt their energy consumption. In this chapter, we focus on computation-intensive system, for example real-time systems. Real-time systems have powerful processor to execute multiple tasks simultaneously. The energy consumption on task processing is dominant, in the range of *60-90%* of the system energy consumption which may include other tasks such as sensing and communication. Tuning energy consumption on task execution therefore makes a large impact on controlling system energy consumption and coping with variations of energy harvesting supply.

For real-time systems, I look at two knobs for energy consumption tuning which are task selection and Dynamic Voltage/Frequency Scalding (DVFS). I target solar-powered firm real-time multi-task systems. Each task executes under one of multiple possible (m,k)-firm constraints (abbreviated as (m,k) constraint), where at least *m* out of every continuous *k* instances of a task are required to complete their execution. In this work, a holistic middleware framework for energy management which orchestrates DVFS and application QoS in solar-powered real-time systems under (m,k) constraints is proposed.

The proposed solution is composed of three components: (a) an offline phase to assign a tentative QoS level for tasks in each time slot in the next harvesting period. This is done based on energy harvesting prediction, multi-level (m,k)-constraint QoS model and estimated energy consumption for each level using nominal speed, (b) a runtime QoS update phase, which adapts per-slot QoS using a more accurate runtime energy harvesting prediction to accommodate anomalous deviations from the offline profile, and (c) a real-time EDF-based task scheduling and DVFS. I prove that our heuristic DVFS scheme when adapting QoS from one level to another level guarantees schedulability. Extensive experiments are carried out to show the effects and the robustness of our technique for both synthetic benchmarks and a case study with smart camera systems. The proposed framework shows an improvement of *19%-50%* in terms of total QoS compared to a DVFS framework for (m,k)-constraint real-time systems that is not harvesting-aware and does not provide adaptive (m,k)-constraint model.

4.1 Introduction

Renewable energy technology has enabled many embedded systems such as environmental/habitat monitoring and structural health monitoring of critical infrastructures and buildings [1] to harvest energy from the surrounding environments, providing perpetual energy for systems to operate autonomously. Renewable energy sources often exhibit both temporal and spatial variations, which cause dynamic fluctuations in harvested energy. There is an increasingly growing research to characterize the solar energy's spatial and temporal variability, mostly exploiting its diurnal and seasonal patterns to extrapolate its future availability [2]. However, unexpected fluctuations caused by the stochastic variability of atmospheric conditions require short-term or runtime prediction algorithms to capture [3]. While prediction can be exploited to some degree, there is still a need for system solutions to cope with energy harvesting uncertainties and fluctuations effectively.

Harvesting-aware schemes at device level to tackle renewable energy fluctuations such as duty cycling [2], DVFS [4], and DPM [5] enable energy tuning. At application level, various models for adapting application quality in energy harvesting systems such as adapting data rate [6], adapting error margin [7], and adapting reward-based QoS [8] can be deployed for adapting energy in systems and matching with harvested energy. In real-time systems, scheduling to meet task deadlines directly affects performance. While application-level adaptive schemes in those systems such as dropping jobs are a knob to provide larger and coarser energy adaptation, device-level adaptive schemes such as DVFS provide further energy savings. This work is an attempt to enable both application and device adaptations to tackle the aforementioned challenges in real-time harvesting systems.

This work targets sustainable firm real-time multi-task systems – i.e., systems with multiple tasks running simultaneously under firm real-time constraints. Firm real time systems allow some tolerance in scheduling, i.e., by dropping jobs selectively. In particular, (m,k) constraint was presented in [15], specifying that at least m out of k consecutive task invocation deadlines need to be met. (m,k) constraint model can adequately enforce both quantity and distribution of job dropping while other metrics such as average percentage of missed deadlines base solely on quantity of job dropping. Furthermore, it has been shown that (m,k) constraints can effectively model QoS for a wide range of applications including multimedia and control applications [10]. For example, in a smart camera system such as CITRICS [13], video images are captured from a camera sensor in a number of frames per second (maximum 30fps). If image or video processing algorithms need at least 10 frames to process each second, the QoS can be modeled as a (10,30) constraint.

In this chapter, I propose a QoS-adaptive and Harvesting-aware middleware framework, named as (m,k)_HAM. (m,k)_HAM orchestrates both application-level and device-level adaptive schemes to benefit from coarse and fine-grained energy tuning in order to cope with dynamic fluctuations in renewable energy sources. I focus on firm real-time applications with reward-based QoS model (multi-level (m,k) constraints for each task) and DVFS as the device-level knob. Our target system has a supercapacitor-based energy storage system. While supercapacitors provide high power density without aging effect, they incur additional challenges in system management due to self-leakage. The proposed framework then takes a further step to consider the non-linear characteristic of the self-leakage in energy storage.

In summary, the proposed framework has three main components: 1) Nominal Speed Computation and QoS Adaptation: Nominal speed is computed for each QoS level to estimate corresponding energy budget. An offline dynamic programming algorithm adapts QoS for the next harvesting period given the estimated energy consumption at each level and the long-term energy harvesting prediction. Considering self-leakage in supercapacitor, QoS assignment by the proposed algorithm remains optimal; 2) Runtime QoS Update: a light-weight greedy algorithm to adapt the QoS given energy storage status and the recent energy harvesting trend. It is run at the beginning of each time slot during a harvesting period, and 3) Real time task scheduling and DVFS: In this step, a real-time preemptive EDF-based task scheduling is followed by DVFS to reduce energy consumption. A heuristic DVFS is developed for transition jobs when QoS changes from one level to another. The distinct technical contributions in design of algorithms and theoretical discussions in each step of the framework are highlighted as follow:

- The original model of (m,k)-firm in which the system has only a fixed (m,k) constraint is extended to be a multi-level (m,k)-constraint QoS model in this work. QoS level can change from time to time to meet energy budget. Such QoS adaptation introduces challenges in scheduling and DVFS for jobs in transition to meet (m,k) constraints. I formally prove that jobs in transition between two QoS levels still meet (m,k) constraints under evenly-distributed (m,k) patterns and CPU utilization ≤ 1. A heuristic DVFS scheme for such jobs to guarantee schedulability is provided.
- Design of optimal offline algorithm for QoS adaptation under non-linear energy storage leakage model (supercapacitor-based).
- Design of a runtime QoS adaptation algorithm and DVFS to tackle variations in energy harvesting profile. The proposed adaptation algorithm is a greedy look-ahead scheme with negligible energy overhead.
- Applying the proposed framework to real-time applications such as image and video processing algorithms in smart camera systems.

- The experimental results demonstrate significant outperformance of our proposed framework compared to existing state-of-the-art works in terms of total QoS and average shutdown time (i.e., during which systems must shut down to replenish energy). The results support our claim that it is important to orchestrate application QoS, DVFS, and energy harvesting and it is necessary to have both offline and runtime QoS adaptation working together to achieve both mentioned goals. A case study with smart camera systems is carried out extensively to show the applicability and impact of (m,k)_HAM.

In summary, our proposed Harvesting-aware energy Management for real-time systems under (m,k) constraints ((m,k)_HAM) is a generalized energy management framework targeting super-capacitor-based solar embedded systems with DVFS capabilities and real-time applications with multiple reward-based QoS levels. However, in this chapter, I advance the theory for applying the framework for (m,k)-constraint real-time applications.

The rest of the chapter is organized as followed. Section 4.2 introduces the related work. Section 4.3 presents our system overview, formulates the problem and introduces the (m,k)_HAM framework. Section 4.4 explains the multi-level (m,k)-constraint QoS model. The proposed framework and algorithms are detailed in section 4.5. The experimental results for a synthetic benchmark and a case study of smart camera systems are shown in section 4.6 and 4.7 respectively. Section 4.8 concludes this chapter.

4.2 Related Work

While in hard real-time systems, a failure to execute a job by its deadline can lead to catastrophes, soft real-time systems and firm real-time systems offer some tolerance to delayed execution or dropping jobs. The metric to measure performance in these systems is usually the number of missed deadlines or the average percentage of missed deadlines. However these metrics are not adequate. For example "*10%* of deadlines can be missed" may mean one missed deadline in every *10* task invocations or *100* missed deadlines followed by *900* met deadlines, each may have a significantly different impact on overall performance. To overcome such shortcoming of the QoS metrics based solely on quantity of missed deadlines, a variety of window-based QoS constraints have been proposed for firm real-time systems to constrain not only the quantity but also the distribution of missed deadlines. For example, [15] introduced the concept of (m,k) constraint for the first time. Later, [10] generalized the concept of (m,k) and introduced the notion of weakly-hard real-time systems to cover other types of dropout patterns. A (m,k) constraint specifies that tasks are desired to meet m deadlines in any k consecutive task invocations. The model has been applied successfully to a broad range of applications including inertial navigation systems, computerized numerical control, webphone, and multimedia applications [25, 13].

There are several related work in the area of energy-efficient real-time scheduling for weakly hard real time systems. [10] proposed a fixed priority scheduling and provided a theoretical analysis on schedulability and bound of response time. [15] proposed a dynamic priority-based scheduling which assigns higher priority to a task that is closer to miss its (m,k) constraint in order to improve its chance of meeting a deadline. [11] monitors and quantifies how close a task to miss a deadline by criticality functions for not only (m,k) constraints but also other window-based constraints. In addition to dynamic priority scheduling, other pattern-based scheduling algorithms have been proposed such as Evenly Distributed pattern [20], or Red Only pattern [17]. These patterns statically identify task invocations to execute to meet (m,k) constraint and may drop other task invocations to reduce energy consumption. [25], [9], and [13] further apply DVFS and DPM techniques to reduce dynamic and leakage energy consumption for real-time systems with (m,k) constraints. These works are not designed for energy harvesting systems, they do not have the adaptation capability needed to address harvesting fluctuations.

In the context of harvesting-aware real-time systems, EDF-based scheduling algorithms were proposed in [12, 14, 23, 27]. Aforementioned EDF-based scheduling is maybe adopted to enhance the performance of our EDF-based scheduler. However, it is not in the scope of this work. Scheduling algorithms exploiting DVFS function of CPU have been studied [4, 21, 19]. These works use number of missed deadlines as QoS metrics. Hence, they are not able to guarantee important QoS property such as distribution of missed deadlines for harvesting real time systems. [4, 21] target non-periodic task sets, therefore when applying for periodic task sets as in our case, the deadlines of tasks might easily be missed. [19] employs a novel harvesting-aware control algorithm for DVFS considering power loss of converters. However, the timing and energy overhead of such control algorithm limit its applicability in real-time systems. Furthermore, [26] proposes a framework for energy harvesting management in multi-core real-time systems.

Recently, [16] addresses (m,k) constraint guarantee in harvesting real time systems. This work proposes a two-step approach for energy management, an offline stage exploiting energy harvesting prediction to select (m,k) constraints for tasks in the next harvesting period and an online adaptation to tackle harvesting variations. This work does not fully support adaptive QoS, it has only one fixed (m,k) constraint for each task for the whole harvesting period. In term of fully adaptive quality-aware framework for energy harvesting systems, QuARES [7] is the closest work. However, their QoS model does not target real-time systems with (m,k) constraints. In addition, they assume ideal energy storage and do not consider DVFS or any device-level adaptation scheme along with data quality adaptation. Our framework is fully QoS adaptive in both offline planning and runtime adaptation. To show the effectiveness and impact of our framework, I investigate a case study of smart camera systems in section 4.7.

4.3 System Overview and Proposed Framework

This section first presents the system overview from the component perspective. The problem of energy management in real-time harvesting systems under (m,k) constraint is then formulated followed by overview of our proposed framework. Figure 4.1 shows the target embedded system, including an energy harvesting sub-system and a central processing unit.

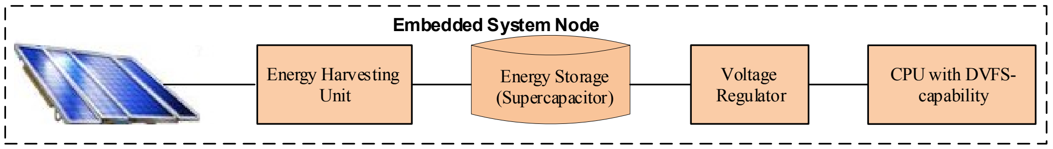


Figure 4.1 Target System

Energy harvesting subsystem generates energy to supply the whole system. This work focuses on solar harvesting because of its high harvesting potential and accessibility. The solar panel is controlled by an energy harvesting unit with Maximum Power Point Tracking (MPPT) to optimize the generated energy. Harvested energy is stored in a supercapacitor(s). I choose supercapacitor because they have high power density, large charge/discharge cycles without aging effect. Their non-negligible leakage is captured and considered in the proposed framework. The initial energy in the supercapacitor at the beginning of a harvesting period is denoted as ES_0. At the end of each harvesting period, the supercapacitor should maintain at least the energy threshold (E_{min}) to ensure sustainability for the system. Supercapacitor supplies power to the CPU through a voltage regulator, I assume a constant loss for voltage regulating. The CPU runs real-time periodic tasks, whose quality can be modeled using the proposed multi-level (m,k) constraint QoS model. The CPU is capable of doing dynamic voltage/frequency scaling (DVFS). I assume the transition time from one speed to another is negligible and the CPU can preempt a task instance, changing its speed or switching to another task at any time.

The proposed framework provides an energy management scheme during each harvesting period. A harvesting period, denoted as T is then divided into N equal-length intervals, called slots. For solar power, T is equal to 1 day and the length of each slot is usually 30 minutes, i.e. $N = 48$. The total amount of harvested energy in each slot i is EH_i ($0 \leq i < N$). Long-term prediction [2] predicts EH_i for all slots i in the next harvesting period based on history of previous k_{long} days while short-term prediction [3] predicts EH_i for the next slot i in the current harvesting period based on history of previous k_{short} slots. The system records history of harvested energy which is used for energy management purposes such as prediction and adaptation.

The energy management framework adapts the energy consumption of the system to match with energy harvesting in order to stay sustainable. Such adaptation can rely on energy harvesting status (prediction and actual values) to assign QoS_i for each slot i, i.e. assigning appropriate (m,k) constraints for each task in order to maximize the overall QoS and ensure fairness among slots.

Problem Formulation: Given a periodic task set, multi-level (m,k)-constraint QoS model, the set of available CPU speeds for DVFS and corresponding power consumption, and the predicted harvested energy $EH_0, EH_1, \dots, EH_{N-1}$ for all the slots in the next harvesting period, the objective is to schedule the task set to lexicographically maximize the pair (QoS_{min}, QoS_{sum}), where $QoS_{min} = min\{QoS_i : i = 0 \dots N-1\}$, and $QoS_{sum} = sum\{QoS_i : i = 0 \dots N-1\}$.

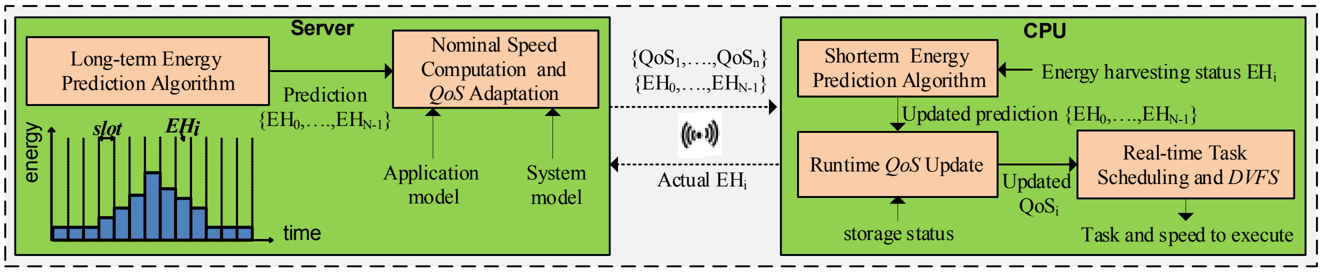

Server
Long-term Energy Prediction Algorithm
Prediction {EH₀,….,EH_{N-1}}
Nominal Speed Computation and QoS Adaptation
Application model
System model
energy
slot
EHi
time
{QoS₁,….,QoSₙ}
{EH₀,….,EH_{N-1}}
Actual EHᵢ
CPU
Shortterm Energy Prediction Algorithm
Energy harvesting status EHᵢ
Updated prediction {EH₀,….,EH_{N-1}}
Runtime QoS Update
Updated QoSᵢ
Real-time Task Scheduling and DVFS
storage status
Task and speed to execute

Figure 4.2 (m,k)_HAM Framework

To address this problem, our proposed framework utilizes three main components at the middleware layer as shown in Figure 4.2.

1) Nominal Speed Computation and QoS Adaptation: Given the periodic task set, multi-level (m,k)-constraint QoS model, and CPU speed set, nominal speed is first computed for each QoS level such that all deadline constraints are met. Energy consumption for each QoS level is estimated based on this nominal speed and corresponding power consumption. The multi-level QoS model with

estimated energy consumption and energy harvesting prediction for the next harvesting period are inputs to an offline dynamic programming algorithm. The algorithm assigns QoS for slots in the next harvesting period to maximize the pair (QoS_{min}, QoS_{sum}). The proposed algorithm is proven to be optimal while considering leakage in the supercapacitor. Given the complexity of this step, it is off-loaded to a cloud or server where power and resource are unlimited.

2) Runtime QoS Update: A light-weight greedy algorithm runs periodically at the beginning of each slot during the harvesting period to adjust QoS in response to fluctuations in actual harvesting EH_i and supercapacitor status. The algorithm relies on recent harvesting history to estimate the trend of harvesting in the near future and its impact on QoS. From this analysis, the algorithm adapts the QoS accordingly.

3) Real-time task scheduling and DVFS: EDF-based scheduling and DVFS for (m,k)-constraint real-time applications is adapted from [9]. A heuristic DVFS is developed for transition jobs when QoS changes from one level to another. The saving in energy consumption from DVFS is accumulated, which can be used to increase QoS.

In the next section, I explain our multi-level (m,k)-constraint QoS model before going into detail of the algorithms for each component in section 4.5.

4.4 Adaptive Multi-level (m,k) Constraint QoS Model

This section presents the proposed multi-level (m,k) constraint QoS model for real time tasks. The set of given periodic tasks is $\tau = \{\tau_i\}$. Each periodic task τ_i is characterized by a tuple (p_i, d_i, e_i)

where p_i is the period, d_i is the relative deadline, and e_i is the execution time. It is assumed that the deadline is the same as the period and the system is not over-loaded, i.e. $U = \sum_{\tau} \frac{e_i}{p_i} < 1.0$. (m,k) constraint is used to model real-time application performance. Figure 4.3 shows an example of a task under *(2,5)* constraint. In all sliding windows of *5* continuous task instances, there are at least *2* instances executed. Although techniques such as EDF can guarantee schedulability for non-harvesting systems with *U≤1.0*, the schedulability in harvesting systems is still challenged by uncertainties and fluctuations in the energy availability.

To meet (m,k) constraint while preserving energy, a number of jobs can be selected to run while others can be dropped. The selected jobs from a pattern are called mandatory jobs while others are called optional jobs. There are well known patterns for selecting mandatory jobs such as E-pattern and Red Only pattern used in the literature [25, 9, 5]. E-pattern for (m,k) constraints [20] categorizes jobs according to Equation 4-1, where $\pi_j = 1$ means mandatory jobs and $\pi_j = 0$ means optional jobs. This pattern distributes mandatory jobs as equally as possible. At the beginning of each slot, the E-pattern is restarted. Figure 4.3 shows the E-pattern for constraint *(2,5)*.

$$\pi_j = \begin{cases} 1, & \text{if } j = \left\lfloor \left\lceil \frac{j \times m}{k} \right\rceil \times \frac{k}{m} \right\rfloor \\ 0, & \text{otherwise} \end{cases} \tag{4-1}$$

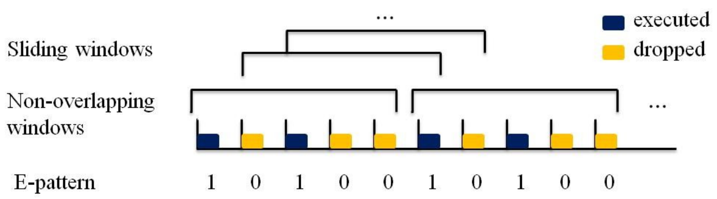


Figure 4.3 Example of a Task under (2,5) Constraint and E-pattern

On the other hand, the Red Only pattern selects the first m of every k jobs as mandatory jobs. Compared to the Red Only pattern, the equal distribution property of E-pattern has better impact on application performance in several ways. [25] proves that if it is feasible to schedule any other pattern, it is also feasible to schedule E-pattern, asserting E-pattern the most schedulable pattern among all possible ones. Furthermore, from the perspective of applications such as a smart camera, it is more desirable to capture and process images frequently to detect events rather than processing multiple images in a short time interval and then staying idle for a long interval (Red Only pattern) when unexpected events might happen.

The multi-level QoS model consists of Q_max levels $(q_1, q_2, \dots, q_{Q\_max})$. Each level q specifies a $(m_i^q,\ k_i^q)$ constraint for task τ_i, where $m_i^q \leq m_i^{q'}$ if $q \leq q'$, assuming $k_i^q = k_i^{q'}$. Because of multi-level QoS model and dynamic adaptation in energy harvesting systems, it is possible that QoS level and tasks' (m,k) constraints change across time slots.

Definition 1: Transition windows are sliding windows that contain jobs from two adjacent slots. A job belongs to a slot if it arrives in that slot.

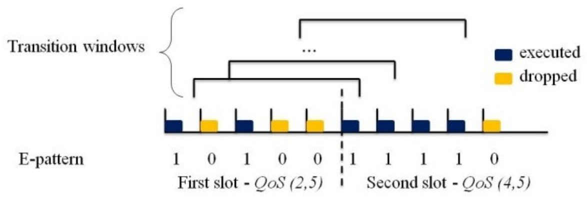
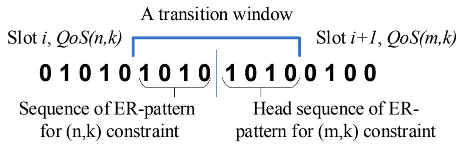


Figure 4.4 Transition Window Example



A transition window
Slot i, QoS(n,k)
Slot i+1, QoS(m,k)
0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0
Sequence of ER-pattern for (n,k) constraint
Head sequence of ER-pattern for (m,k) constraint

Figure 4.5 E-Pattern Changes as QoS Changes

Figure 4.4 shows an example of a task with QoS *(2,5)* in the first slot and QoS *(4,5)* in the second slot. There are *4* transition windows. Among them, one has QoS *(2, 5)*, two have QoS *(3,5)* and one has QoS *(4,5)*. Not all transition windows have the same QoS, yet I want to put a theoretical lower bound on QoS of these transition windows. I prove that all transition windows would meet (m,k) constraint of at least one slot.

Let *l(z)* be a sequence of a *1* followed by all *0's*, whose length is *z*. In each non-overlapping window of *k* continuous jobs, E-pattern for constraint (m,k) is composed of x_m sequences $l(\left\lfloor \frac{m}{k} \right\rfloor)$ followed by y_m sequences $l(\left\lceil \frac{m}{k} \right\rceil)$ where

$$x_m = m \times (t_m + 1) - k \text{ and } y_m = k - m \times t_m \text{ where } t_m = \left\lfloor \frac{k}{m} \right\rfloor \qquad \textbf{(4-2)}$$

Let $g_m(h,l)$ be the number of 1's in a sequence of the E-pattern for constraint (m,k), whose length is *l* and which starts from job *h*. Note that $g_m(\sigma k,k) = m$. In addition, we have:

$$g_m(0,l) = \begin{cases} 1 + \left\lfloor \dfrac{l-1}{t_m} \right\rfloor & \text{if } l \le x_m t_m \\ 1 + \left\lfloor \dfrac{l + x_m - 1}{t_m + 1} \right\rfloor & \text{otherwise} \end{cases} \quad \text{where } 0 \le l \le k \qquad \textbf{(4-3)}$$

The first lemma and corollary below compare $g_m(h,l)$, the number of *1's* in sequences of the same length *l* and under the same (m,k) constraint, the starting points *h* however can be different. Lemma 2 and 3 compare the number of *1's* in sequences of the same length *l* but under different (m,k) constraints.

Lemma 1: In the E-pattern for (m,k) constraint, the number of *1's* in a sequence of length *l* starting from position σk (head sequence) is greater or equal to the number of *1's* in any sequence of length *l*, i.e.

$$\mathrm{g_m}(\sigma\mathrm{k},\mathrm{l}) \geq \mathrm{g_m}(\mathrm{h},\mathrm{l}) \text{ where } 0\leq \mathrm{l},\mathrm{h},\sigma \qquad \textbf{(4-4)}$$

This lemma is a restatement of lemma 4 from [20], hence it is not necessary to include a proof. Using Lemma 1, I prove the following corollary.

Corollary 1: In the E-pattern for (m,k) constraint, the number of *1's* in a sequence of length *l* starting from position *σk-l* (tail sequence) is smaller or equal to the number of *1's* in any sequence of length *l*, i.e.

$$\mathrm{g_m}(\sigma\mathrm{k}-\mathrm{l},\mathrm{l}) \leq \mathrm{g_m}(\mathrm{h},\mathrm{l}) \text{ where } 0\leq\sigma,\mathrm{h} \text{ and } 0\leq \mathrm{l}\leq\sigma\mathrm{k} \qquad \textbf{(4-5)}$$

Proof: From Lemma 1, $g(\sigma k - l, l + h) \leq g(0, l + h)$. Both these sequences share identical subsequences $g(\sigma k, h) = g(0, h)$. Removing these identical subsequences, Equation (4-5) is derived and hence it is proved. □

Now let us compare the number of *1's* in sequences of the same length *l* but under different (m,k) constrains.

Lemma 2: The number of *1's* in a head sequence of length *l* in the E-pattern for constraint (m,k) is greater than or equal to that in a head sequence of the same length *l* in the E-pattern for constraint (n,k) where *m≥n*, i.e.

$$g_m(\sigma k, l) \geq g_n(\omega k, l) \text{ where } 0 \leq \sigma, \omega, l, \text{ and } m \geq n \qquad \textbf{(4-6)}$$

Lemma 3: The number of *1's* in a tail sequence of length *l* in the E-pattern for constraint (m,k) is greater than or equal to that in a tail sequence of the same length *l* in the E-pattern for constraint (n,k) where *m≥n*, i.e.

$$g_m(\sigma k - l, l) \geq g_n(\omega k - l, l) \text{ where } 0 \leq \sigma, \omega,\ 0 \leq l \leq \min(\sigma k, \omega k), \text{ and } m \geq n \qquad \textbf{(4-7)}$$

The proofs for Lemma 2 and Lemma 3 are shown in the Appendix. With these lemmas and corollary proven, we are now ready to prove the lower bound of QoS for all transition windows.

Theorem 1: Let the QoS of the two adjacent slots be (n,k) and (m,k). For any transition window starting from job instance *h*, $g(h,k) \geq min(m,n)$ where *s-k<h<s* and *s* is the first job of the second slot.

Proof: There are two cases below:

Case 1: *n≤m*. At the beginning of the second slot, starting from job *s*, the new E-pattern for constraint (m,k) starts. In all transition windows (see Figure 4.5), a head sequence of (m,k) of length *l* replaces a sequence of (n,k) with the same length. According to lemma 2, a head sequence of (m,k) has more or same *1's* as a head sequence of (n,k) of the same length *l* because *n≤m*, whereas according to lemma 1, a head sequence of (n,k) has more or same *1's* as any sequence of the same length under the same (n,k) constraint. Therefore, in each transition window, the new sequence of E-pattern for (m,k) constraint creates more or same *1's* as the E-pattern with (n,k) constraint. As a result, the number of *1's* in each transition window is greater than or equal to *n*.

Case 2: *n>m*. Each transition window is composed of a sequence of length *k-l* of E-pattern for (n,k) constraint followed by a head sequence of E-pattern for (m,k) constraint of length *l*. According to corollary 1, a sequence of length *k-l* for (n,k) constraint has more or same *1's* as a tail sequence of the same length and under the same (n,k) constraint. According to Lemma 3, a tail sequence of length *k-l* under (n,k) constraint has more or same *1's* as a tail sequence of the same length but with (m,k) constraint where *n>m*. As a result, the number of *1's* in each transition window is greater than or equal to *m*.

In both cases, the number of *1's* in any transition window is greater than or equal to min *(m,n)*. Therefore, we have Theorem 1 proved. □

Corollary 2: QoS of all transition windows is always greater than or equal to the minimum QoS of the two slots.

The above corollary is an immediate result following Theorem 1 so I do not include a proof here. This theoretical result shows that multi-level (m,k) constraint QoS model enables smooth QoS adaptation in energy harvesting systems.

4.5 Algorithms for QoS-Adaptive Energy Management

Our framework consists of three main components. The first component is an offline dynamic programming based algorithm which adapts the QoS in different slots based on energy available in the system provided by a long-term prediction algorithm. The second one is a light-weight greedy algorithm to adapt the QoS at runtime given the recent energy harvesting trend and energy storage status. Finally, the framework has an EDF-based real time scheduler for mandatory jobs followed by dynamic voltage/frequency scheduling to reduce energy consumption. In this section, I present our algorithms for each components mentioned above.

4.5.1 Speed Nominal Computation and Offline QoS Adaptation

This offline planning leverages the given energy harvesting prediction for the next harvesting period to assign optimal QoS for each slot. Assume the CPU has M available frequency and voltage pairs, (f_j,V_j). The system is said to run at speed $s_j = f_j/f_{max}$ when it runs at frequency f_j and voltage V_j. Hence, the set of available CPU speeds is $\{s_1,s_2,..,s_M\}$ and the corresponding power consumption is $\{power_{s_1}, power_{s_2}, \dots, power_{s_M}\}$.

Energy consumption for each QoS level is estimated using a nominal speed. The method in [18] is utilized to find a minimum nominal continuous speed for mandatory jobs of all tasks at each QoS level, by finding the highest intensity interval in the hyper-period. Each QoS level has a different (m,k) constraint for each task, the number and distribution of mandatory jobs at each level are hence different and so is the corresponding nominal continuous speed. This continuous speed is approximated by the closest (larger or equal) discrete speed s available in the system. The energy consumption EC_q in a slot for each QoS level q is estimated based on the number of mandatory jobs, the execution time $\frac{e_j}{s}$ for each job j at the selected speed s and power consumption $power_s$, as follow

$$\mathrm{EC_q} = \sum_j \frac{\mathrm{T}}{\mathrm{Np_j}} \times \frac{\mathrm{m_j^q}}{\mathrm{k_j^q}} \times \frac{\mathrm{e_j}}{\mathrm{s}} \times \mathrm{power_s} \tag{4-8}$$

Given the energy harvesting prediction *EH_i* for each slot *i* and energy consumption estimation *EC_q* for each QoS level *q* above, the offline QoS adaptation assigns QoS for each slot in order to maximize *{QoS_{min},QoS_{sum}}*. QoS adaptation inherently changes the nominal speed (i.e., voltage and frequency) across time slots. Note that energy consumption is estimated at nominal speed for each slot because static optimal DVFS is computational expensive and it is not accurate as task arrival and execution time change at runtime. The real time scheduler and runtime DVFS will be presented in section 4.5.3.

The energy at the beginning of slot *i+1* is computed recursively as followed:

$$\mathrm{ES_{i+1}} = \mathrm{ES_i} + \mathrm{EH_i} - \mathrm{EC_q} - \mathrm{EL_i} \tag{4-9}$$

where ES_i is energy at the beginning of slot *i* and *EL_i* is energy leakage in slot *i*. In this work, a lightweight approximation model for supercapacitor leakage [22] is used.

$$\mathrm{P_{Leakage}(V_{sup})} = \propto \mathrm{e^{\beta V_{sup}}} \tag{4-10}$$

$$\mathrm{EL_i} = \frac{\mathrm{T}}{\mathrm{N}} \times \mathrm{P_{leakage}}(\overline{\mathrm{V_{sup,\imath}}}) \tag{4-11}$$

where V_{sup} is the voltage of the supercapacitor, α and β are empirical parameters of the supercapacitor. $\overline{V_{sup,\imath}}$ is the average voltage in the supercapacitor corresponding to the average energy $E_{avg,i}$ during slot *i*:

$$E_{avg,i} = ES_i + \frac{1}{2}(EH_i - EC_q) = \frac{1}{2}C\overline{V_{sup,\imath}}^2 \qquad \textbf{(4-12)}$$

Our extensive experiments show that this estimation of leakage for short time period such as a slot of *30* minutes has negligible error. It is especially true when the rates of harvesting and consumption are close and energy stored in the supercapacitor changes gradually. Substitute Equation (4-10 to 4-12) into (4-9), we have:

$$ES_{i+1} = \frac{1}{2}C\overline{V_{sup,\imath}}^2 + \frac{1}{2}\left(EH_i - EC_q\right) + \frac{T}{N}\alpha e^{\beta\overline{V_{sup,\imath}}} \qquad \textbf{(4-13)}$$

For a given EHi and ECq, ESi+1 is monotonically increasing with $\overline{V_{sup,\imath}}$ if the first order derivative of (13) with respect to V_{sup} is non-negative, i.e.

$$\frac{dES_{i+1}}{dV} = CV_{sup} - \frac{T}{N}\alpha\beta e^{\beta V_{sup}} \geq 0 \qquad \textbf{(4-14)}$$

I observe that for supercapacitor parameters reported in [22], the first order derivation above is always nonnegative. Therefore this property, i.e., *ES_{i+1}* is monotonically increasing with $\overline{V_{sup,\iota}}$, applies for a large class of supercapacitors.

Definition 1: A QoS assignment $A = \{QoS_i, i: 0 \ldots N-1\}$ is feasible if the energy in the supercapacitor at the beginning of all slots is positive and remaining energy in the supercapacitor at the end of the harvesting period is greater or equal to threshold E_{min}.

Algorithm 4-1 finds the optimal QoS_{min}. It iterates through all QoS levels from Q_{max} downto 1. For each QoS level, it performs a test of feasibility, considering energy harvesting, energy consumption and non-linear energy leakage in the supercapacitor. Because energy leakage is non-linear, it is not straightforward that the QoS found by algorithm 1 is optimal QoS_{min}. I formally prove this in Lemma 4 below.

Lemma 4: The QoS_{min} found by Algorithm 4-1 is optimal if the supercapacitor satisfies the inequality (4-14).

ALGORITHM 4-1. Finding Optimal QoS_{min}

```
Input: $ES_0$, $E_{min}$, $EH_i$, $Q\_max$ levels $(q_1, q_2, \ldots, q_{Q\_max})$ and corresponding $EC_q$ for each level $q$
Output: $QoS_{min}$
1.  for each QoS level $q = Q_{max}$ to 1 do
2.      for each $i = 0$ to $N-1$ do
3.          compute $ES_{i+1}$ using Equation (9);
4.          if ($ES_{i+1} < 0$) then
5.                  Check the next QoS level;
6.          end
7.      end
8.      if ($ES_N < E_{min}$) then
9.          Check the next QoS level;
10.     else
11.         $QoS_{min}=q$;
12.     end
13. end
```

Proof: I prove this lemma by contradiction. Assume that *QoS_{min}* found by Algorithm 4-1 is not optimal *QoS_{min}*. Therefore, there must exist a feasible QoS assignment A in which $QoS'_{min} = min\{QoS_i(A), i: 0 \ldots N-1\}$ and $QoS'_{min} >$ *QoS_{min}*. Let $A' = \{QoS_i = QoS'_{min}, i: 0 \ldots N-1\}$. I prove that if A is a feasible QoS assignment, A' is also a feasible QoS assignment.

By definition, A' is feasible if $ES'_i \geq 0$ for all slots *i:0..N-1* and $ES'_N \geq E_{min}$. I prove this by induction. At the beginning of a harvesting period, energy storage in supercapacitor is the same, i.e., $0 \leq ES_0 = ES'_0$. Given $ES_i \leq ES'_i$, remaining energy at the beginning of the next slot for assignment A and A' is

$$\mathrm{ES_{i+1}} = \frac{1}{2}\mathrm{CV^2} + \frac{1}{2}\left(\mathrm{EH_i} - \mathrm{EC_{QoS_i}}\right) + \frac{\mathrm{T}}{\mathrm{N}}\alpha \mathrm{e}^{\beta \mathrm{V}} \qquad \textbf{(4-15)}$$

$$\mathrm{ES'_{i+1}} = \frac{1}{2}\mathrm{CV'^2} + \frac{1}{2}\left(\mathrm{EH_i} - \mathrm{EC_{QoS'_{min}}}\right) + \frac{\mathrm{T}}{\mathrm{N}}\alpha \mathrm{e}^{\beta \mathrm{V'}} \qquad \textbf{(4-16)}$$

Since the supercapacitor satisfies the inequality (4-14), ES_{i+1} is monotonically increasing function of voltage *V*. Since $ES_i \leq ES'_i$, *V≤V'*. In addition, $EC_{QoS_i} \geq EC_{QoS'_{min}}$ because $QoS_i \geq QoS'_{min}$. As a result, $0 \leq ES_{i+1} \leq ES'_{i+1}$. At the end of the harvesting period, $E_{min} \leq ES_n \leq ES'_n$.

Therefore A' is a feasible assignment in which $QoS_i = QoS'_{min} > q, \forall i: 0..N-1$. This is a contradiction to Algorithm 4-1 because it must have checked all QoS levels greater than optimal *QoS_{min}* and all of them (including QoS'_{min}) must be infeasible. □

Given the optimal QoS_{min} determined by Algorithm 4-1, the dynamic programming solution in [8] is adapted to find the optimal QoS_{sum}. I extend the algorithm to take into consideration the non-linear supercapacitor leakage and prove that the dynamic programming is still correct.

Let *D[i,Q]* be the maximum energy remained in the supercapacitor at the beginning of slot *i (i:0..N)* if the sum of the QoS in the first *i* slots is *Q*. $D[0,0] = ES_0$. To update *D[i,Q]* where $QoS_{min} \times i \leq Q \leq Q_{max} \times i$, the following equation is used

$$D[i, Q] = \max_{QoS_{min} \leq q \leq Q\_max} \{D[i-1, Q-q] + EH_{i-1} - EC_q - EL_{i-1}\} \tag{4-17}$$

D[i,Q] is computed as the maximum energy remaining in the storage considering all possible QoS assignment ($\geq QoS_{min}$) for the previous slot *i-1* given QoS sum of the first *i* slots is *Q*. At the end, maximum QoS_{sum} is determined as

$$QoS_{sum} = \max\{Q: D[N, Q] \geq E_{min}\} \tag{4-18}$$

ALGORITHM 4-2. Finding Optimal QoS_{sum}

Input: ES_0, E_{min}, EH_i, *Q_max* levels $(q_1, q_2, \ldots, q_{Q\_max})$ and corresponding EC_q for each level q

Output: QoS_{sum}

```
1. for slot i= 1..N do
2.     for $Q = QoS_{min} \times i..Q\_max \times i$ do
3.         Compute D[i,Q] according to Equation (4-17);
4.     end
5. end
6. Compute $QoS_{sum}$ according to Equation (4-18);
```

The pseudo-code for the dynamic programming to find optimal QoSsum is given in Algorithm 4-2. Despite supercapacitor leakage is not a linear function, I prove that the dynamic programming is still correct.

Lemma 5: The output QoS_{sum} of the dynamic programming above is optimal if supercapacitor satisfies the inequality (4-14).

Proof: In order to prove this, I first prove that the function *f* below is monotonically increasing with *D[i-1,Q-q]* for a given *q*

$$f = D[i-1, Q-q] + EH_{i-1} - EC_q - EL_{i-1} \tag{4-19}$$

Substitute Equation (4-10 to 4-12) into (4-19), we have

$$f = \frac{1}{2}CV^2 + \frac{1}{2}\left(EH_{i-1} - EC_q\right) + \frac{T}{N}\alpha e^{\beta V} \tag{4-20}$$

Where $\frac{1}{2}CV^2 = D[i-1, Q-q] + \frac{1}{2}(EH_{i-1} - EC_q)$

Since the supercapacitor satisfies the inequality (4-14), the function *f* is monotonically increasing with voltage *V*. Since *D[i-1,Q-q]* is a quadratic function of *V*, *f* is also monotonically increasing with *D[i-1,Q-q]*. Therefore, *D[i,Q]* can rely on the maximum value of *D[i-1,Q-q]* which was computed by dynamic programming algorithm. Hence the dynamic programming result, QoS_{sum} is optimal. □

Complexity: To find the nominal speed for tasks in the hyper-period takes $O(nM^2)$ where N is the number of tasks, and M is the number of jobs in the hyper-period. This repeats for each QoS levels, hence the complexity to find nominal speed for all QoS levels is $O(nM^2QoS_{max})$. The runtime complexity of the Algorithm 4-1 to find maximum QoS_{min} is $O(QoS_{max}N)$ while the complexity of the dynamic programming in Algorithm 4-2 to find maximum QoS_{sum} is $O(N^2QoS_{max}^2)$.

4.5.2 Run-time QoS Adaptation

The predicted energy harvesting is usually different from the actual harvesting at runtime due to changes in the weather. Long-term energy harvesting prediction algorithm such as [2] has inaccuracy (*10%-40%* in our experiments).

Hence, at the beginning of each time slot, I propose an efficient algorithm to adjust the QoS; by accessing the trend of energy harvesting at runtime. The trend factor $\mu(i)$ is defined as the weighted ratio of predicted energy harvesting $EH_j^{predicted}$ and actual energy harvesting EH_j^{actual} in the last k_{short} slots.

$$\mu(i) = \frac{\sum_{j=1}^{k_{short}} \omega(j) \times \frac{EH_{i-j}^{actual}}{EH_{i-j}^{predicted}}}{\sum_{j=1}^{k_{short}} \omega(j)} \qquad (4\text{-}21)$$

where the weight for each of previous k_{short} slots is defined as $\omega(j) = \frac{1}{j}$. The trend factor is applied to future energy harvesting and to estimate the fluctuation's impact factor $\theta(i)$ on QoS.

$$\theta(i) = \frac{ES_i + \mu(i)\sum_{j:i..N-1} EH_j^{predicted} - E_{min}}{\sum_{j:i..N-1} EC_{QoS_j}} \qquad (4\text{-}22)$$

Equation (4-22) is the ratio of expected available energy and energy required for current QoS assignment in the remaining slots, *i..N-1*. It estimates the impact of harvesting fluctuations on QoS. If harvesting amount in remaining slots and harvesting fluctuations play an important role in supplying energy for QoS of the remaining slots, its impact will reflect in $\theta(i)$. If energy harvesting in the remaining slots is insignificant compared to accumulated energy in the supercapacitor ESi, the fluctuations will not have significant impact on $\theta(i)$. This ratio is used in the Algorithm 4-3 below to adjust QoS of the current slot *i*. Once a new energy budget is computed, the updated QoS is found using a binary search among sorted QoS levels.

ALGORITHM 4-3. Run-time *QoS* adaptation

```
Input: $QoS_i$, $EH_j^{actual}$($j = i-k_{short} .. i-1$), $EH_{j'}^{predicted}$($j' = i..N-1$)
Output: Updated $QoS_i$
1. compute $\mu(i)$ according to Equation (21);
2. compute $\theta(i)$ according to Equation (22);
3. new_budget = $\mu(i) \times EC_{QoS_i}$;
4. $QoS_i$ = BinarySearch$QoS$ (new_budget);
```

4.5.3 Real-time Scheduler and DVFS

At runtime, jobs may finish earlier than its worst case execution time. In addition, since the nominal discrete speed is greater than (or equal to) the nominal continuous speed, jobs may finish earlier than its scheduled finishing time. These two factors create slack time which, when accumulated, can be used to significantly reduce frequency/voltage of following jobs and save energy consumption. The extra saving energy from runtime DVFS can compensate for harvesting abatement or improve QoS in the future.

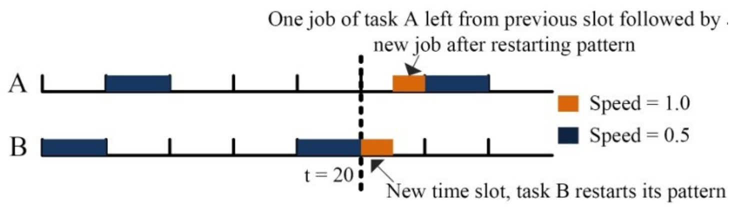

One job of task A left from previous slot followed by new job after restarting pattern
A
B
Speed = 1.0
Speed = 0.5
t = 20
New time slot, task B restarts its pattern

Figure 4.6 Scheduling Tasks in Transition

The runtime DVFS is adapted from previous work [9] whose effectiveness has been shown for continuous speed. In this work, I adapt the technique by choosing the closest discrete speed to the continuous speed. Our distinct contribution is in frequency/voltage scaling during transition windows which was not considered in previous DVFS techniques for systems under (m,k) constraints.

I show an example in Figure 4.6 that scheduling jobs in transition windows at the maximum speed of the two slots will not guarantee schedulability of all mandatory jobs. For instance, assume there are two tasks:

Task A: execution time = 2, period = 8 with QoS (1,2)

Task B: execution time = 2, period = 4 with QoS (1,4)

Using nominal speed computation described in section 4.5.1, the system can run at speed $s = 0.5$, meeting all mandatory job deadlines and satisfying both task A and B's QoS constraints. Assume at $t = 20$, a new slot starts and QoS of both tasks remains the same, the nominal speed would be the same $s = 0.5$. However, the E-pattern for each task restarts. Figure 4.6 shows that at $t=20$, there is one job of task A arriving at $t = 16$ in the previous slot is still in the ready queue. At the same time, one new job of task B arriving at $t = 20$ becomes mandatory after its E-pattern restarts. The only option to meet both these jobs' deadlines is to execute them at full speed $s = 1.0$. This example shows our observation that neither speed of the previous slot nor the current slot can guarantee meeting all mandatory job deadlines. After completing jobs in transition windows, the system can resume to run at the nominal speed.

Lemma 6: If the system runs jobs in the ready queue at $s = min\,(1.0, s_1 + s_2)$ while there are still transition jobs, where s_1 and s_2 are nominal speeds for the previous and current slots respectively, schedulability is guaranteed for all jobs.

Proof: Since utilization of the whole task set $U_{tot} \leq 1.0$, it is possible to schedule all mandatory jobs at full speed $s_{max} = 1.0$.

Now let t_1 denote the time when the current slot starts. J_1 is the set of mandatory jobs in the previous slot arriving before t_1, and J_2 is the set of mandatory jobs of the current slot arriving at or after t_1. The utilization of J_1 is called U_1 and of J_2 is called U_2. We have $U_1 \leq s_1, U_2 \leq s_2$, and $U_1 + U_2 \leq s_1 + s_2$. The system should maintain speed $s = s_1 + s_2$ until the latest deadline (t_2) of jobs which are ready at t_1.

Let t_0 be the end of the last idle time before t_1; so there is no idle time in (t_0, t_1). Assume that a deadline is missed at $t_1 \leq d \leq t_2$. Then there cannot be an idle time in (t_0, d). Because jobs in J_1 executes at speed s_1 in (t_0, t_1), the remaining execution time of jobs in J_1 after t_1 in the worst case is given by:

$$s_1\left(\frac{1}{s_1}\sum_{J_1}\left|\frac{d-t_0}{p_i}\right|e_i-(t_1-t_0)\right) \tag{4-23}$$

Because there is no idle time in (t_1, d), the demand from J_2 in (t_1, d) is given by:

$$\sum_{J_2}\left|\frac{d-t_1}{p_i}\right|e_i \tag{4-24}$$

Therefore, to miss a deadline at *d*, we have

$$(t_1-t_0)+\frac{s_1\left(\frac{1}{s_1}\sum_{J_1}\left|\frac{d-t_0}{p_i}\right|e_i-(t_1-t_0)\right)+\sum_{J_2}\left|\frac{d-t_1}{p_i}\right|e_i}{s_1+s_2}>d-t_0 \tag{4-25}$$

which gives the following:

$$s_1\left(\frac{1}{s_1}\sum_{J_1}\left|\frac{d-t_0}{p_i}\right|e_i-(t_1-t_0)\right)+\sum_{J_2}\left|\frac{d-t_1}{p_i}\right|e_i>(d-t_1)(s_1+s_2) \tag{4-26}$$

And we have

$$\sum_{J_1}\left|\frac{d-t_0}{p_i}\right|e_i-s_1(t_1-t_0)+\sum_{J_2}\left|\frac{d-t_1}{p_i}\right|e_i>(d-t_1)(s_1+s_2) \tag{4-27}$$

which is followed by

$$\begin{aligned}(d-t_1)(s_1+s_2)&<U_1(d-t_0)-s_1(t_1-t_0)+U_2(d-t_1)\\ &\leq U_1(d-t_0)-U_1(t_1-t_0)+U_2(d-t_1)\end{aligned} \tag{4-28}$$

which eventually gives

$$(\mathrm{U}_1 + \mathrm{U}_2)(\mathrm{d} - \mathrm{t}_1) > (d - \mathrm{t}_1)(\mathrm{s}_1 + \mathrm{s}_2) \qquad \textbf{(4-29)}$$

The inequality (29) above contradicts the fact that $U_1 + U_2 \leq s_1 + s_2$. Therefore, it is possible to schedule all mandatory jobs at speed *s= min(1.0,* $s_1 + s_2$*)*.□

Algorithm 4-4 shows our real-time scheduler and DVFS. When a new job arrives, it is put in to the ready queue if it is a mandatory job according to its QoS constraint in the current slot and the E-pattern, otherwise it is discarded (line 2-6). To compute slack time, the system maintains an auxiliary data structure called alpha queue which has remaining time of jobs if they are executed at nominal continuous speed and with worst case execution time. Jobs in both ready queue and alpha queue are sorted according to its EDF order (or arrival time if they have the same deadline).

The system calls the scheduler when it is ready to execute a new job. This happens when a job finishes, or it is dropped due to lack of energy or when a new job arrives. The scheduler chooses the job at the head of the ready queue (line 11). If there is still any transition job in the ready queue, this job at the head of the ready queue is executed at the speed selected by Lemma 5 (line 12-13). Otherwise, the job is scheduled at a speed according to the nominal speed and extra slack time it has, taking into consideration the discrete speeds available on the processor (line 15-16). Scanning through the alpha queue of worst case schedule, it is possible to compute available slack time for run time DVFS.

When a job is finished, dropped, or pre-empted, the alpha queue is synchronized to reflect the remaining worst case execution time of jobs (line 10). Detail of the alpha queue implementation can be found in [9]. When a job is finished or dropped, its pattern and QoS violation count are also updated (line 9).

ALGORITHM 4-4. Real-time scheduler and *DVFS*

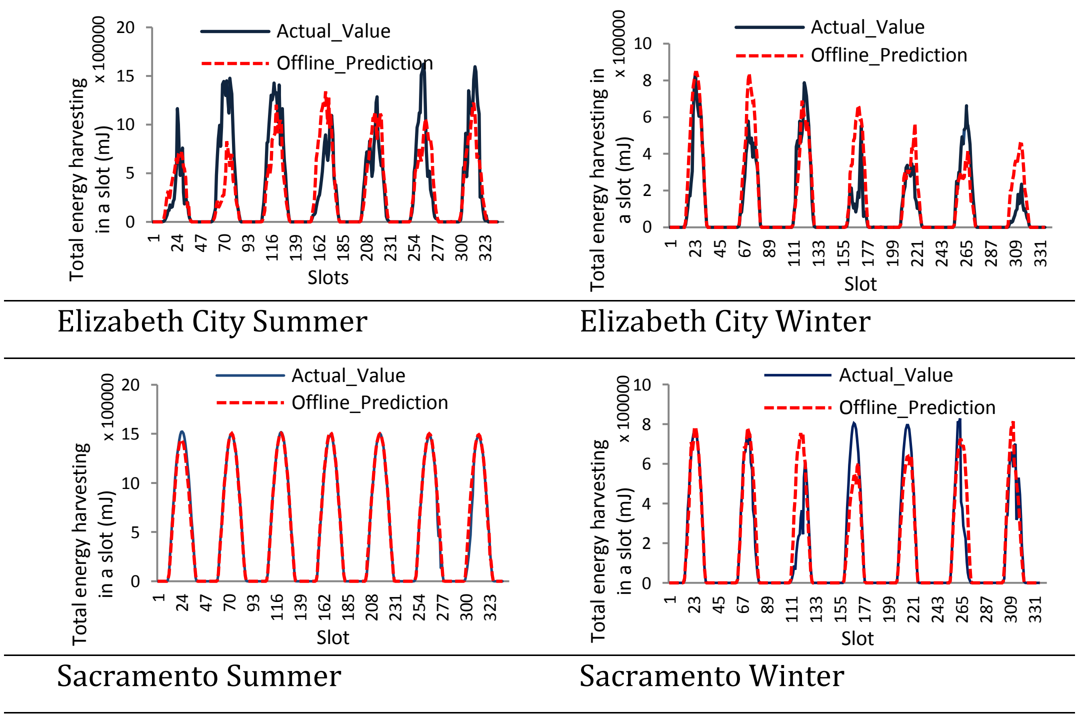
Input: Ready queue and alpha queue

Output: next job to run and its speed *s*

```
1. Task_arriving(job)
2.     if job is mandatory according to E-pattern & QoS in the current slot then
3.         ready_queue.add(job);
4.         alpha_queue.add(job);
5.     end
6.     schedule();
7. Schedule() //when a job is finished, dropped, or preempted
8.     update task pattern and check if there is QoS violation;
9.     update alpha_queue;
10.    job = read_queue.next();
11.    if there is transition job in the ready queue then
12.        $s$ = min(1.0, $s_1+s_2$);
13.    else
14.        compute slack time by consulting the alpha queue;
15.        set speed $s$ according to nominal speed and slack time;
16.    end
```

4.6 Experiments

The experimental setup is first explained in section 4.6.1. The proposed adaptive QoS framework, (m,k)_HAM for energy harvesting real-time systems under (m,k) constraints is evaluated using a synthetic benchmark, comparing with state-of-the-art work. Experimental results for a case study of smart camera systems are shown in section 4.7.



Elizabeth City Summer

Elizabeth City Winter

Sacramento Summer

Sacramento Winter

Figure 4.7 Solar Harvesting Profile and Prediction for a Week in Elizabeth City and Sacramento, Summer and Winter, 2013

4.6.1 Experimental Setup

I set up a simulation environment that integrates the harvesting system, the energy storage, and the DVFS-capable processor with our proposed framework, (m,k)_HAM. Harvesting period T is equal to one day, number of slots N is *48* and slot duration is *30* minutes. The offline QoS adaptation are run before each new harvesting period on a server and the run-time QoS adaptation is called at the beginning of each slot.

The experiments use harvesting data from [24] in Sacramento, California and Elizabeth City, North Carolina. The solar irradiance is converted to harvested energy by linear conversion considering solar panel size of *9.6 cm × 7.6 cm*, solar cell efficiency *10%* and harvesting efficiency *80%*. Figure 4.7 shows the total energy harvesting in each slot for a week in summer and a week in winter in both locations. Harvesting profile in Elizabeth City shows a large variation among slots within a day and between days in a week. It also has a significant gap between prediction and actual harvesting. On the other hand, harvesting profile in Sacramento, CA consistently shows a higher harvesting potential than Elizabeth City and its pattern has less variation, especially in the summer. In the long-term solar energy prediction algorithm, energy harvesting data from the last *3* days is used to predict energy harvesting of the next harvesting period.

The target embedded processor is PXA270 processor which operates at seven different voltages and frequencies as shown in Table 4.1 below. Energy storage is a supercapacitor of 2000F, the leakage parameters are extracted from [22].

Table 4.1 Processor PXA270 DVFS Configuration

Frequency (MHz)	13	104	208	312	416	520	624
Power (mW)	44	116	279	390	570	747	925

In our synthetic benchmark, task sets are randomly generated with number of tasks between *[6...10]*. Task periods are uniformly distributed in the range *[5...30]* seconds. Jobs arrive with jitter uniformly distributed between *0-10%* of the period. Worst case execution times are generated and scaled based on utilization. *100* random task sets are generated for each utilization {*0.4, 0.8*} to represent low and high utilization scenarios and idle time is shorter than break-even time for sleep modes. The average ratio of actual execution time over worst case execution time is *0.8*. For all tasks, I chose *k = 5* and there are five QoS levels where *m = [1...5]*. The solar panel size and the supercapacitor size are doubled for utilization of *0.8*.

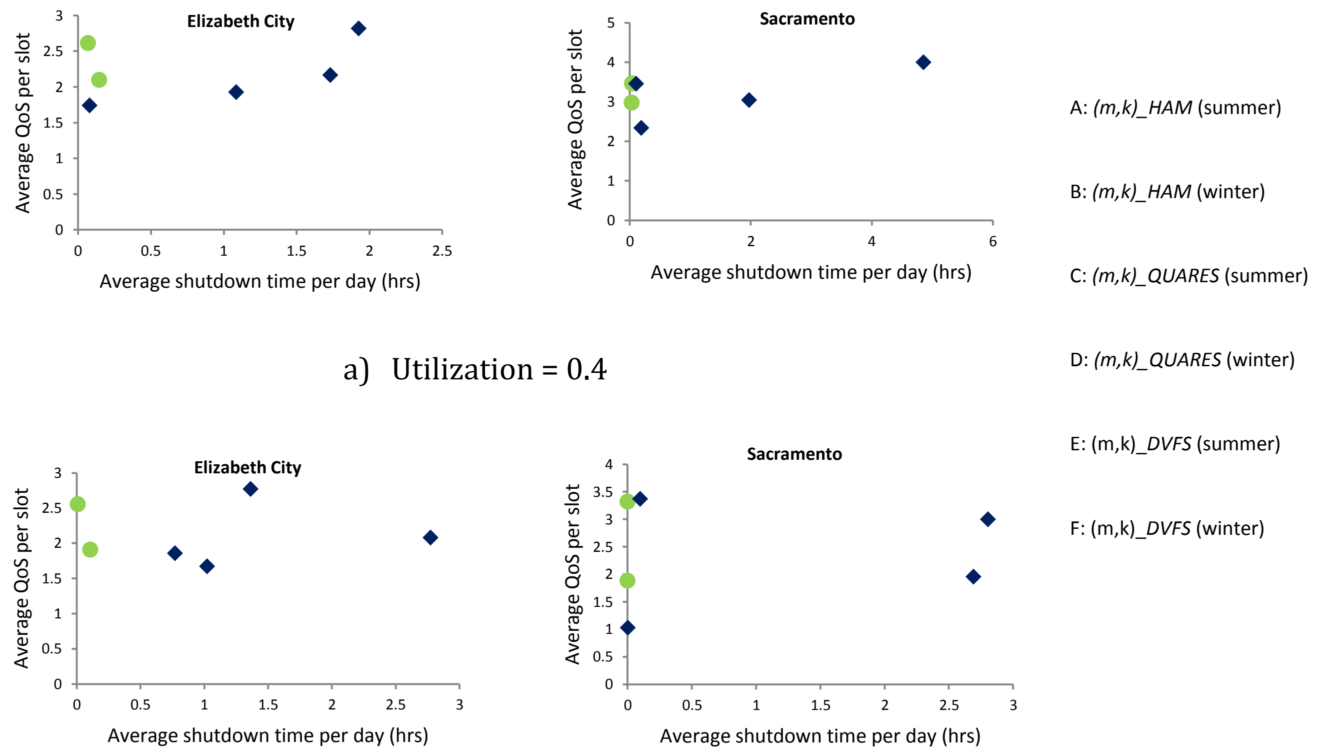
4.6.2 Experimental Results

To test the effectiveness of our proposed framework, several experiments compare this work with related work are set up.

Comparison with existing work: Since this is the first work to propose an adaptive framework for real-time energy harvesting systems under (m,k) constraints, I compare our approach, (m,k)_HAM, with two adapted state-of-the-art work. Our multi-level (m,k) constraint QoS model and corresponding energy consumption for each level (after applying nominal speed computation in section 4.5.1) are input to each of the adapted related work below.

- (m,k)_DVFS: is adapted from [9]. The original DVFS work is based on single (m,k) constraint QoS model. I extend it to allow selection from multi-level (m,k) QoS model each day based on predicted average energy harvesting per slot. This QoS level is applied to all slots and there is no adjustment at run time. This approach does not have QoS adaptation in both offline and at run time.
- (m,k)_QUARES: is adapted from [7] which is an adaptive QoS framework for energy harvesting systems. It is adapted to take in multi-level (m,k) constraint QoS model as input instead of the original data quality model. QUARES is however not aware of supercapacitor leakage. In addition, QUARES assumes ideal energy harvesting prediction, and its run time adaptation does local adaptation per slot which does not consider the trend of harvesting with fluctuations.

Figure 4.8a shows our comparison results with existing work for utilization of *0.4*. In each graph, the x-axis shows the average shutdown time per day while the y-axis shows the average QoS per slot. When the system runs out of energy, it must shut down to replenish energy from harvesting sources. The average QoS per slot is essentially QoS_{sum} each day divided by number of slots, N. A good approach should show both system sustainability and high performance, i.e. low shutdown time and high QoS. In all graphs, our approach (m,k)_HAM is consistently in the higher left corner, achieving negligible shut down time per day and high QoS per slot simultaneously. Both (m,k)_QUARES and (m,k)_DVFS, either show very high shutdown time or lower QoS. (m,k)_QUARES has similar average QoS per slot to (m,k)_HAM but it has very high shutdown time (in average, *1.5-2* hrs each day). (m,k)_QUARES is an adaptive QoS framework; it is able to exploit energy harvesting prediction to optimize QoS in the offline phase. However, its offline phase is not aware of leakage in the supercapacitor, making its offline plan inaccurate. Furthermore, (m,k)_QUARES's runtime QoS adaptation is local to each slot, it does not look ahead in its adaptation. Because of these two shortcomings, the system with (m,k)_QUARES often runs out of energy and has to shut down to replenish energy. These are obviously not desirable properties of a sustainable system.



a) Utilization = 0.4

b) Utilization = 0.8

Figure 4.8 Comparison of (m,k)_HAM with Adapted Existing Work, (m,k)_QUARES and (m,k)_DVFS

(m,k)_DVFS is an energy-efficient technique developed for a single (m,k) QoS model systems, it has neither offline QoS nor runtime QoS adaptation. As a result, the system has lower QoS than other adaptive QoS approaches, i.e. (m,k)_HAM and (m,k)_QUARES (*19%-27%*). In addition, in some scenarios (Elizabeth City summer and Sacramento summer), without a runtime QoS adaptation mechanism to handle changes in energy harvesting condition, it has high average shut-down time per day.

The result is similar for *U=0.8* as shown in Figure 4.8b. (m,k)_HAM has negligible shut down time and high QoS in all cases. (m,k)_QUARES has high average shut down time per day (almost 3 hours) while (m,k)_DVFS has lower QoS (*24%-50%*). The only exception is in the case of Sacramento summer where both (m,k)_DVFS and (m,k)_HAM have no shutdown time and similar average QoS.

(m,k)_HAM demonstrates strength in both dimensions, sustainability and performance consistently in all experiments. It has better QoS and much less shutdown time compared to existing work.

Offline QoS adaptation vs. Runtime QoS adaptation: To understand the importance of each of these QoS adaptation components and how they work in concert with each other, several variants are set up for comparison with (m,k)_HAM:

- (m,k)_HAM_A: In this setting, there is no QoS adaptation. The offline phase only determines optimal QoS_{min} based on energy harvesting prediction, leakage estimation and nominal speed computation. At runtime, tasks are scheduled by the EDF-based scheduler followed by runtime DVFS.

- (m,k)_HAM_B: This setting has no offline QoS adaptation; the offline phase determines optimal *QoS_{min}* for all slots. The runtime QoS adaptation is enabled to work with the EDF-based scheduler and runtime DVFS.
- (m,k)_HAM_C: The offline phase is fully adaptive, finding optimal QoS assignment for all slots. The runtime QoS adaptation, however, is disabled. Tasks are scheduled by the EDF-based scheduler followed by runtime DVFS.

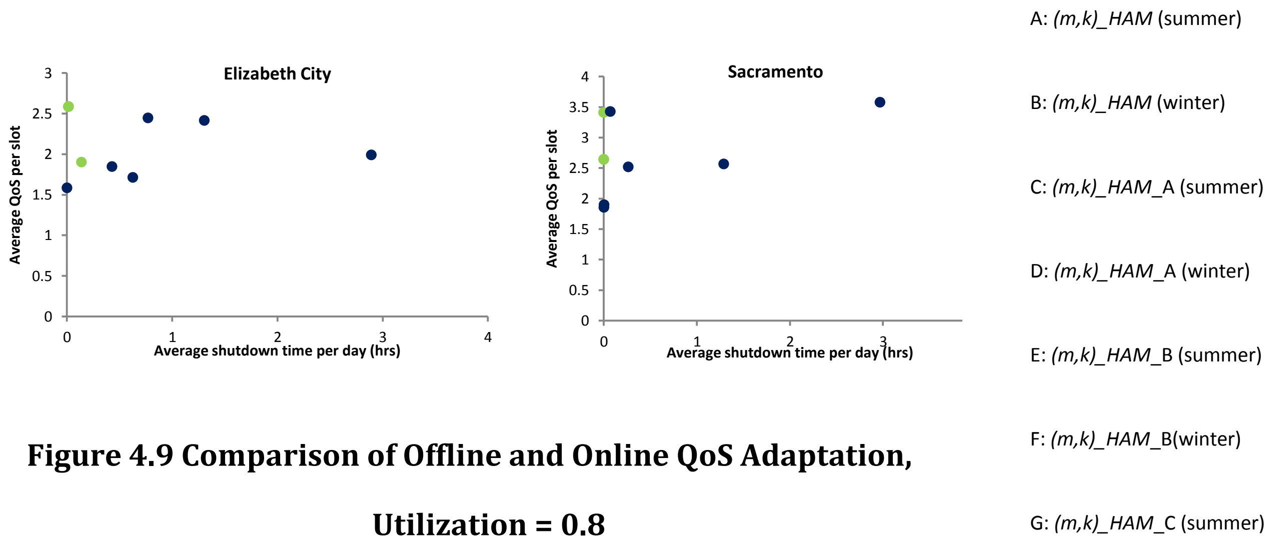


Figure 4.9 Comparison of Offline and Online QoS Adaptation, Utilization = 0.8

As shown in Figure 4.9, without offline and runtime QoS adaptation, (m,k)_HAM_A has lower QoS than other variants. Compared to (m,k)_HAM, this variant has *52%* lower QoS on average. Since it always runs at low fixed *QoS_{min}*, it consumes less energy and has low shutdown time. (m,k)_HAM_B, the variant without offline QoS adaptation but with run-time QoS adaptation, has better QoS than (m,k)_HAM_A. However, its shut down time is higher (average 0.2-0.7hrs shutdown time each day). (m,k)_HAM_C, the variant with offline QoS adaptation but no run-time adaptation has similar QoS to (m,k)_HAM. Without runtime adaptation, this approach is not able to react to changes in the weather conditions (resulting in average 1.5-3hrs shutdown each day). This can be observed clearly when actual harvesting is significantly different from the offline harvesting prediction (in all cases except for Sacramento summer). This experiment shows the importance of full adaptation with both offline QoS and runtime QoS adaptation in achieving high QoS for sustainable energy harvesting systems.

4.7 Case Study - Smart Camera Systems

This section presents a case study – harvesting smart camera systems. Traditional sensor networks with temperature, humidity or acoustic sensors usually have a low cost and low power micro-processor on each node with small memory for simple processing tasks. Applications such as camera surveillance, traffic monitoring, object detection, however, require more powerful processor and larger memory to capture, process, and store large-size images.

Smart camera networks [13] recently emerge as a new class of wireless sensor networks. Each node in a smart camera network is an integrated platform consisting of a camera, a microphone, and a frequency-scalable CPU, replacing traditional low power micro-processor (see Figure 4.10). Because of limited network bandwidth, real-time transmission of high quality and high frame rate is impossible. Smart camera networks push more computation and storage to the edge of the network where powerful processors are capable of doing pre-processing tasks for a wide range of applications. The preprocessed data can be stored on the nodes which have sufficient memory. The nodes only communicate with each other and with a central server when needed (e.g., event detections) via the standard sensor network protocols. [13] measure power consumption of the integrated platform and the results show that the CPU consumes about *50-60%* of total system power consumption. Therefore, being able to adapt the CPU energy consumption as in our framework, (m,k)_HAM is important. I propose to power each node with solar panel and supercapacitor as shown in Figure 4.10. These are called harvesting smart camera networks and systems. Our framework, (m,k)_HAM focuses on application QoS adaptation and DVFS at each node in the network.

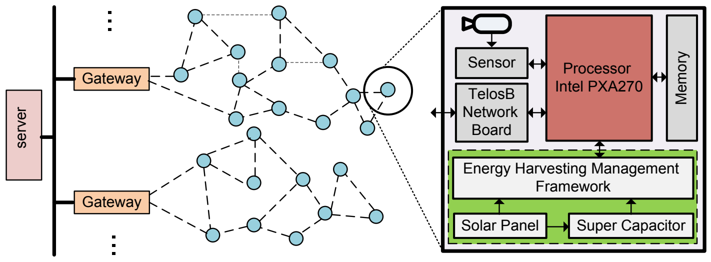


Figure 4.10 Harvesting Smart Camera Networks and Systems

Image processing tasks have been shown to be good candidate for (m,k) constraint model [10]. In this study, the focus is on two applications, feature extraction (section 4.7.1) and object detection (section 7.2). Object detection experiments are run with solar panels = *9.6 cm × 7.6 cm*, 1 supercapacitors of *2000F*, E_{min}=*40%* full supercapacitor. Feature extraction experiments are run with double size of the solar panel and the supercapacitor. The harvesting data input is the same as in the synthetic benchmark experiments.

4.7.1 Feature Extraction Application

Feature extraction is a comprehensive application which does not only detect object but also obtain the contour of the occluding object. The application combines of background subtraction and median filter tasks followed by Canny edge detector and array add [13]. Table 4.2 shows the execution time for each of these image processing tasks and for different image resolution (*64 x 64* *pixels* and *128 x 128 pixels*). For each image size, I choose a feasible period that allows room for DVFS and attains high maximum frames per second. At the bottom, the table shows the total utilization of the task set and its multi-level QoS range. For image size *64 x 64*, the period is set to *33ms*, the maximum frames per second (QoS) is *30fps*. The minimum QoS is *5fps* to meet application requirement. For image size *128 x 128*, I set a period of *50ms*, allowing the maximum QoS be *20fps*. Another possibility for setting period for image size *128 x 128* is *100ms*, allowing more room for DVFS but lower QoS range of *5-10fps*.

Table 4.2 Feature Extraction Application and Individual Tasks' Execution Time (ms) for Different Image Resolutions

Task (Description)	**Image size 128x128**			**Image size 64x64**	
	Exec. Time	Period-1	Period-2	Exec. Time	Period
T1 (IPP Canny Edge Detector)	22 ms	50 ms	100 ms	5.2 ms	33 ms
T2 (IPP Median Filter)	2.1 ms	50 ms	100 ms	0.48 ms	33 ms
T3 (IPP Background Subtraction)	1.4 ms	50 ms	100 ms	0.15 ms	33 ms
T4 (IPP Add)	0.48 ms	50 ms	100 ms	0.049ms	33 ms
Total Utilization	0.52		0.26	0.18	
QoS Range	5-20 fps		5-10 fps	5-30 fps	

Table 4.3 shows the result of our case study for feature extraction application. For image size *64 x 64*, the utilization is low. There is room for aggressive DVFS to reduce energy consumption and the overall energy consumption is low. The average QoS per slot is almost at the maximum level,

30fps. There is no shutdown time. For image size, *128 x 128*, the results show that the setting with period of *50ms* has higher QoS than the setting with period of *100ms*. However, during winter in Elizabeth City, there is a significant average shutdown time for setting with period of *50ms*. The second setting (period of *100ms*) does not have shutdown time because its period is larger, the system does more aggressive offline speed reduction and runtime DVFS to reduce energy consumption and to cope with harvesting fluctuations.

Table 4.3 Feature Detection Experimental Results

	Image size 64x64		**Image size 128x128 (period=50ms)**		**Image size 128x128 (period=100ms)**	
	Average QoS (fps)	Shutdown time	Average QoS	Shutdown time	Average QoS (fps)	Shut-down time
Sacramento – Winter	29.9	0	12.0	0	9.8	0
Sacramento – Summer	30	0	19.9	0	10	0
Elizabeth City – Winter	28.6	0	8.64	0.6hrs	8.2	0
Elizabeth City - Summer	29.7	0	18.0	0	9.8	0

This result suggests that to achieve high QoS and low shutdown time, the system should configure itself with image size *128 x 128* and *period=50ms* in most seasons except for Elizabeth City in winter. During such harsh time, it should have a lower QoS setting, such as image size *128 x 128* and *period=100ms* or image size *64 x 64*. These are other dimensions rather than (m,k) constraint that can be adapted to improve system sustainability. It is beyond the scope of this work but it is an interesting observation for future work.

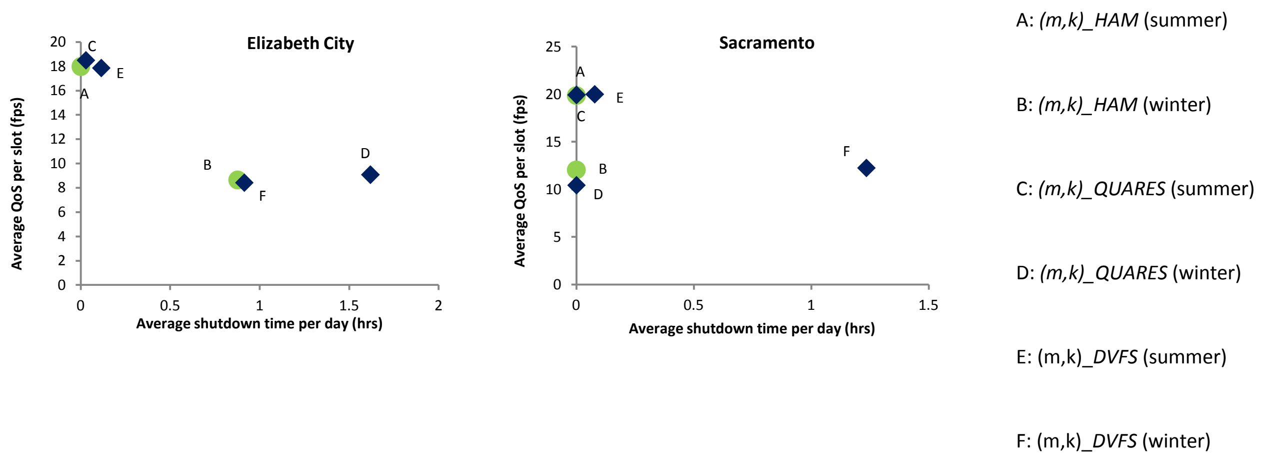


Figure 4.11 Comparison with Existing work, Case Study – Feature Extraction Application

Figure 4.11 shows our comparison with adapted existing work for this case study of feature extraction applications on energy harvesting smart camera systems. Except for Elizabeth City winter (where harvesting is low), (m,k)_HAM consistently enables continuous operation with high average QoS. While (m,k)_QUARES and (m,k)_DVFS perform well in Elizabeth City and Sacramento summers, they show inferior performance in Sacramento winter. (m,k)_QUARES has high shutdown time (average *1 hr* per day) while (m,k)_DVFS has *16%* lower average QoS per slot compared to (m,k)_HAM.

4.7.2 Object Detection Application

Object detection application is an important and extremely useful application for smart camera systems. It includes a background subtraction task followed by a median filter task in order to detect a new object that is not in the default background [13]. Table 4.4 shows the tasks for an object identification application, their execution time and period for three image resolutions *64 x 64* pixels, *128 x 128* pixels and *256 x 256* pixels. For both image size *64 x 64* and *128 x 128*, the period is set to be *33ms*, providing the maximum QoS of *30fps*. For image size *256 x 256*, I have two settings: one with period of *50ms* and one with period of *100ms*.

Table 4.4 Object Detection Application and Individual Tasks' Execution Time (ms) for Different Image Resolutions

Task (Description)	Image size 256x256		Image size 128x128		Image size 64x64	
	Exec. Time	Period	Exec. time	Period	Exec. Time	Period
T2 (Median Filter)	35 ms	50/100 ms	8.3	33 ms	2.0	33 ms
T3 (Background Subtr.)	3.7 ms	50/100 ms	0.42	33 ms	0.093	33 ms
T4 (Add)	3 ms	50/100 ms	0.29	33 ms	0.061	33 ms
Total Utilization	0.8/0.4		0.27		0.06	
QoS Range	5-20 fps/5-10fps		5-30 fps		5-30 fps	

The results for object identification are shown in Table 4.5 and Table 4.6. For image size *64 x 64*, there is no shutdown time and in all cases, the system achieves the maximum QoS of *30fps*. For image size *128 x 128*, there is no shutdown time. The average QoS is, however, lower than the QoS for image size *64 x 64* since at higher image resolution, tasks are longer and are more computational intensive. For image size *256 x 256*, the system has reasonable QoS without shutdown time during the summers but there is significant shutdown time during the winters. In this case, it requires a larger solar panel to harvest sufficient energy and to supply energy demand. With a larger solar panel, our experiment shows that it is feasible to run object detection application with image size *256 x 256* without any shutdown.

Table 4.5 Object Identification Result for Image Size 64x64 and 128x128

	Image size 64x64		Image size 128x128	
	Average QoS	Shut-down time	Average QoS	Shut-down time
Sacramento – Winter	30 fps	0	25.4 fps	0
Sacramento – Summer	30 fps	0	30 fps	0
Elizabeth City – Winter	30 fps	0	17.7	0
Elizabeth City - Summer	30 fps	0	29 fps	0

Table 4.6 Object Identification Result for Image Size 256x256

	Image size 256x256 (p=50ms)		Image size 256x256 (p=100ms)	
	Average QoS	Shut-down time	Average QoS	Shut-down time
Sacramento – Winter	6.9 fps	0.8hrs	8.19 fps	0
Sacramento – Summer	15.8 fps	0	10 fps	0
Elizabeth City – Winter	5.8 fps	3.1hrs	6.3 fps	2.0hrs
Elizabeth City - Summer	12.5 fps	0	9.69 fps	0

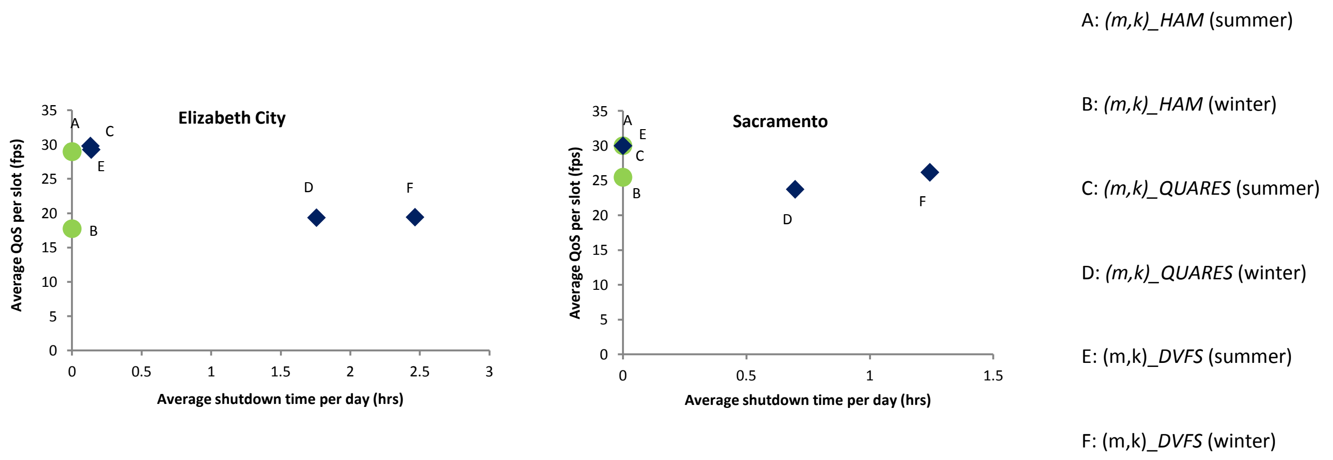


Figure 4.12 Comparison with Existing Work, Case Study – Object Identification Application

Figure 4.12 shows result for object identification application with resolution *128 x 128*. Results are similar to case study of feature extraction applications where in the summers, all the approaches perform well. However, in the winter, both (m,k)_QUARES and (m,k)_DVFS have very high shutdown time (on average, *0.7-2.5* hrs per day). Our approach, (m,k)_HAM provides high average QoS without shut-down time in both locations and in all seasons.

4.8 Conclusion

In this chapter, I propose a unified framework for adapting application QoS and exploiting DVFS in energy harvesting real-time systems under (m,k) constraints. I prove QoS adaptation will not violate QoS constraints during transition windows and I propose a heuristic DVFS policy for jobs in transition such that schedulability is guaranteed. Our experiments prove that offline planning and runtime QoS adaptation along with DVFS, tightly coupled with knowledge of energy storage and harvesting prediction is important to mask fluctuations in ambient energy harvesting sources, achieving sustainable systems with high QoS performance.

APPENDIX

Lemma 2: The number of *1's* in a head sequence of length *l* in the E-pattern for constraint (m,k) is greater than or equal to that in a head sequence of same length *l* in the E-pattern for constraint (n,k) where *m>=n*, i.e.

$$g_m(\sigma k, l) \geq g_n(\omega k, l) \text{ where } 0 \leq \sigma, \omega, l, \text{ and } m \geq n \qquad \textbf{(4-30)}$$

Proof: Let *l' = l % k*. Then Inequality (4-30) is equivalent to

$$\mathrm{g_m}(0,\mathrm{l}') \geq \mathrm{g_n}(0,\mathrm{l}'), \qquad \forall \mathrm{l}': 0 \leq \mathrm{l}' < k, m \geq n \tag{4-31}$$

Let $t_m = \left\lfloor \frac{k}{m} \right\rfloor$ and $t_n = \left\lfloor \frac{k}{n} \right\rfloor$, $t_m \leq t_n$ because *m≥n*. Consider these cases:

Case 1*:* $l' \leq x_m t_m$ and $l' \leq x_n t_n$. According to Equation (4-3), Inequality (4-31) is equivalent to $\left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_m}} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_n}} \right\rfloor$, which is obvious because $t_m \leq t_n$.

Case 2: If $l' > x_m t_m$ and $l' \leq x_n t_n$. According to Equation (4-3), Inequality (4-31) leads to

$\left\lfloor \frac{\mathrm{l}'+\mathrm{x_m}-1}{\mathrm{t_m}+1} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_n}} \right\rfloor$. We consider two further cases below:

If $t_m < t_n$ then $t_m + 1 \leq t_n$, we have $\left\lfloor \frac{\mathrm{l}'+\mathrm{x_m}-1}{\mathrm{t_m}+1} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_n}} \right\rfloor$.

Consider the case when $t_m = t_n$. According to Equation (2), we have $\mathrm{l}' \leq \mathrm{x_n} \times \mathrm{t_n} = (\mathrm{n} \times (\mathrm{t_n}+1) -$ k×tn≤m×tm+1−k×tm=xm×tm<l' which is a contradiction.

Case 3: $\mathrm{l}' \leq x_m t_n$ and $l' > x_n t_n$. According to Equation (4-3), Inequality (4-31) is

$\left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_m}} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'+\mathrm{x_n}-1}{\mathrm{t_n}+1} \right\rfloor$. Because $x_n t_n < l'$, we have $(\mathrm{l}'+\mathrm{x_n}-1)\mathrm{t_n} \leq (\mathrm{l}'-1)(\mathrm{t_n}+1)$, thus $\left\lfloor \frac{\mathrm{l}'+\mathrm{x_n}-1}{\mathrm{t_n}+1} \right\rfloor < \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_n}} \right\rfloor \leq \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_m}} \right\rfloor$.

Case 4: $l' > x_m t_n$ and $l' > x_n t_n$. According to Equation (4-3), Inequality (4-31) is

$\left\lfloor \frac{\mathrm{l}'+\mathrm{x_m}-1}{\mathrm{t_m}+1} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'+\mathrm{x_n}-1}{\mathrm{t_n}+1} \right\rfloor$. We consider two further cases below:

If $t_m < t_n$ then $t_m + 1 \leq t_n$, which gives $\left\lfloor \frac{\mathrm{l}'+\mathrm{x_m}-1}{\mathrm{t_m}+1} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'-1}{\mathrm{t_n}} \right\rfloor \geq \left\lfloor \frac{\mathrm{l}'+\mathrm{x_n}-1}{\mathrm{t_{n+1}}} \right\rfloor$.

Consider the case $t_m = t_n$. Applying Equation (4-2), we have:

$$x_m = m(t_m + 1) - k \geq n(t_n + 1) - k = x_n. \text{ Therefore, } \left\lfloor \frac{l' + x_m - 1}{t_m + 1} \right\rfloor \geq \left\lfloor \frac{l' + x_n - 1}{t_n + 1} \right\rfloor. \square$$

Lemma 3: The number of *1's* in a tail sequence of length *l* in the E-pattern for constraint (m,k) is greater than or equal to that in a tail sequence of same length *l* in the E-pattern for constraint (n,k) where *m>=n*, i.e.

$$g_m(\sigma k - l, l) \geq g_n(\omega k - l, l) \text{ where } 0 \leq \sigma, \omega,\ 0 \leq l \leq \min(\sigma k, \omega k), \text{ and } m \geq n \qquad \textbf{(4-32)}$$

Proof: Let *l' = l % k*. Then Inequality (4-32) is equivalent to

$$g_m(k - l', l') \geq g_n(k - l', l'), \qquad \forall l' : 0 \leq l' < k \text{ and } m \geq n \qquad \textbf{(4-33)}$$

Let $t_m = \left\lfloor \frac{k}{m} \right\rfloor$ and $t_n = \left\lfloor \frac{k}{n} \right\rfloor$, $t_m \leq t_n$ because *m≥n*. Similar to Equation (4-3), we have

$$g_m(k - l', l') = \begin{cases} \left\lfloor \dfrac{l'}{t_m + 1} \right\rfloor & \text{if } l' \leq y_m(t_m + 1) \\ \left\lfloor \dfrac{l' - y_m}{t_m} \right\rfloor & \text{otherwise} \end{cases} \qquad \textbf{(4-34)}$$

Case 1: $l' \leq y_m(t_m + 1)$ and $l' \leq y_n(t_n + 1)$. According to Equation (4-34), Inequality (4-33) becomes $\left\lfloor \frac{l'}{t_m + 1} \right\rfloor \geq \left\lfloor \frac{l'}{t_n + 1} \right\rfloor$ which is obvious because $t_m \leq t_n$.

Case 2: $l' \leq y_m(t_m + 1)$ and $l' > y_n(t_n + 1)$. According to Equation (4-34), Inequality (4-33) is $\left\lfloor \frac{l'}{t_m + 1} \right\rfloor \geq \left\lfloor \frac{l' - y_n}{t_n} \right\rfloor$. There are two further cases below:

If $t_m < t_n$ then $t_m + 1 \le t_n$, which gives $\left\lfloor\frac{l'}{t_m+1}\right\rfloor \ge \left\lfloor\frac{l'-y_n}{t_n}\right\rfloor$

If $t_m = t_n$. Applying Equation (4-2) and because $m \ge n$, we have

$l' \le y_m(t_m + 1) = (k - mt_m)(t_m + 1) \le (k - nt_n)(t_n + 1) = y_n(t_n + 1) < l'$ which is a contradiction

Case 3: $l' > y_m(t_m + 1)$ but $t' \le y_n(t_n + 1)$. Applying Equation (4-34), Inequality (4-33) is equivalent to $\left\lfloor\frac{l'-y_m}{t_m}\right\rfloor \ge \left\lfloor\frac{l'}{t_n+1}\right\rfloor$.

We have $(l' - y_m)(t_m + 1) > l' \times t_m$ from $l' > y_m(t_m + 1)$, which leads to

$$\left\lfloor\frac{l'-y_m}{t_m}\right\rfloor > \left\lfloor\frac{l'}{t_m+1}\right\rfloor \ge \left\lfloor\frac{l'}{t_n+1}\right\rfloor.$$

Case 4: $l' > y_m(t_m + 1)$ and $l' > y_n(t_n + 1)$. Applying Equation (4-34) to Inequality (4-33), we have $\left\lfloor\frac{l'-y_m}{t_m}\right\rfloor \ge \left\lfloor\frac{l'-y_n}{t_n}\right\rfloor$ to be proven. Because $m \ge n$ and $t_n \ge t_m$, trivially $(l' - k + mt_m)t_n \ge (l' - k + nt_n)t_m$.

Using $y_m = k - mt_m$ and $y_n = k - nt_n$ by definition in Equation (4-2), we have

$(l' - y_m)t_n \ge (l' - y_n)t_m$ which is followed by $\left\lfloor\frac{l'-y_m}{t_m}\right\rfloor \ge \left\lfloor\frac{l'-y_n}{t_n}\right\rfloor$. □

References

[1] W.K.G. Seah, Zhi Ang Eu, H. Tan. "Wireless sensor networks powered by ambient energy harvesting (WSN-HEAP) - Survey and challenges." In VITAE2009.

[2] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. "Adaptive duty cycling for energy harvesting systems." In ISLPED’06, pp. 180-185. ACM, 2006.

[3] J. Recas Piorno, Carlo Bergonzini, David Atienza, and T. Simunic Rosing. "Prediction and management in energy harvested wireless sensor nodes." In Wireless VITAE 2009.

[4] Shaobo Liu, Qinru Qiu, and Qing Wu. "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting." In DATE'08, pp. 236-241. IEEE, 2008.

[5] Linwei Niu, and Gang Quan. "Leakage–aware scheduling for embedded real–time systems with (m, k)–constraints." International Journal of Embedded Systems 5, no. 4 (2013): 189-207.

[6] Kai-Wei Fan, Zizhan Zheng, and Prasun Sinha. "Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks." In Proceedings of the 6th ACM conference on Embedded network sensor systems, pp. 239-252. ACM, 2008.

[7] Nga Dang, Elaheh Bozorgzadeh, and Nalini Venkatasubramanian. "QuARES: Quality-aware data collection in energy harvesting sensor networks." In Green Computing Conference and Workshops (IGCC), 2011 International, pp. 1-9. IEEE, 2011.

[8] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. "Power management in energy harvesting embedded systems with discrete service levels." In Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design, pp. 413-418. ACM, 2009.

[9] Tarek A. AlEnawy, and Hakan Aydin. "Energy-constrained scheduling for weakly-hard real-time systems." In Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International, pp. 10-pp. IEEE, 2005.

[10] Guillem Bernat, Alan Burns, and Albert Liamosi. "Weakly hard real-time systems." Computers, IEEE Transactions on 50, no. 4 (2001): 308-321.

[11] Guillem Bernat and Ricardo Cayssials “Guaranteed On-Line Weakly-Hard Real-Time Systems”, IEEE Real-Time Systems Symposium, 2001

[12] Maryline Chetto. "Optimal scheduling for real-time jobs in energy harvesting computing systems." IEEE Transactions on Emerging Topics in Computing (2014).

[13] Phoebus Chen, Kirak Hong, Nikhil Naikal, S. Shankar Sastry, Doug Tygar, Posu Yan, Allen Y. Yang et al. "A low-bandwidth camera sensor platform with applications in smart camera networks." ACM Transactions on Sensor Networks (TOSN) 9, no. 2 (2013): 21.

[14] Hussein El Ghor, Maryline Chetto, and Rafic Hage Chehade. "A real-time scheduling framework for embedded systems with environmental energy harvesting." Computers & Electrical Engineering 37, no. 4 (2011): 498-510.

[15] Moncef Hamdaou and Parameswaran Ramanathan, “A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines”, IEEE Transactions on Computers, Vol 44, No. 12, December 1995

[16] Hessam Kooti, Nga Dang, Deepak Mishra, and Eli Bozorgzadeh. "Energy budget management for energy harvesting embedded systems." In Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on, pp. 320-329. IEEE, 2012.

[17] G. Koren and D. Shasha, “Skip-over: Algorithms and complexity for overloaded systems that allow skips,” in Proc. RTSS, 1995, p. 110.

[18] Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced cpu energy. In Proceedings of IEEE Symposium on Foundations of Computer Science. 374–382.

[19] Xue Lin , Yanzhi Wang , Siyu Yue , Naehyuck Chang and Massoud Pedram, “A Framework of Concurrent Task Scheduling and Dynamic Voltage and Frequency Scaling in Real-Time Embedded Systems with Energy Harvesting”, in ISLPED, 2013

[20] Parameswaran Ramanathan. "Overload management in real-time control applications using (m, k)-firm guarantee." Parallel and Distributed Systems, IEEE Transactions on 10, no. 6 (1999): 549-559.

[21] Jun Lu, Shaobo Liu, Qing Wu, and Qinru Qiu. "Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems." In Green Computing Conference, 2010 International, pp. 469-476. IEEE, 2010.

[22] Azalia Mirhoseini, and Farinaz Koushanfar. "HypoEnergy. hybrid supercapacitor-battery power-supply optimization for energy efficiency." In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, pp. 1-4. IEEE, 2011.

[23] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. "Dynamic power management in environmentally powered systems." In Proceedings of the 2010 Asia and South Pacific Design Automation Conference, pp. 81-88. IEEE Press, 2010.

[24] National Renewable Energy Lab. [Online]. http://www.nrel.gov

[25] Linwei Niu, and Gang Quan. "Energy minimization for real-time systems with (m, k)-guarantee." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 14, no. 7 (2006): 717-729.

[26] Yi Xiang and Sudeep Pasricha. "Harvesting-Aware Energy Management for Multicore Platforms with Hybrid Energy Storage”, in GLSVLSI’13

[27] Maryline Chetto and Audrey Queudet. "Clairvoyance and online scheduling in real-time energy harvesting systems." Real-Time Systems 50.2 (2014): 179-184.

Chapter 5 Orchestrated Application Quality and Energy Storage Management in Solar-Powered Embedded Systems

While energy harvesting technology is a promising solution toward achieving self-sustainable low power systems, the efficient energy storage for these energy harvesting systems is still a challenge because of high self-leakage (e.g., supercapacitors) and limited life cycles (e.g., batteries). In this work, I propose an adaptive quality-aware energy management middleware framework for energy harvesting embedded systems. Our hybrid energy storage model takes into consideration the battery life cycle, supercapacitor self leakage, and power loss in the harvesting circuit. The framework has an offline planning phase and a runtime adaptation phase. By incorporating abstract models for battery state of health (SoH) and supercapacitor self-leakage, the offline stage determines the budget for charging and discharging distribution of each storage component and accordingly adapts the application quality of service (QoS).

The runtime adaptation phase dynamically adjusts the charging and discharging distribution to the dynamic changes in energy harvesting profile. In comparison with related work, our proposed framework is able to capture the lifetime and characteristics of the energy storage components more accurately during adaptation and hence, resulting in a more sustainable system with realistic QoS.

5.1. Introduction

Renewable energy technology is a viable and promising solution for low power networked embedded systems. Environmental energy sources are ubiquitous in our surroundings and each system can be equipped with an energy harvesting circuitry to scavenge the energy from such sources. However, the spatial and temporal variations in scavenging energy from the environment lead to uncertainty in energy availability during operation.

Temporal variations refer to energy harvesting condition changes in time (e.g., day vs. night) while spatial variations refer to energy harvesting condition changes in space (e.g., locations under shadow vs. locations with direct sunlight). These variations continue to challenge the energy sustainability in embedded systems.

A solution to mask this uncertainty is using energy storage as a buffer to smooth down the variations in energy harvesting. The choice of energy storage is important because each type of energy storage has its own advantage and disadvantage. While batteries provide energy storage for a long period of time, they are constrained by limited battery lifetime (i.e., limited charging/discharging cycle counts) and power density (maximum charging/discharging current). On the other hand, lifetime of supercapacitors, another type of energy storage, is not a concern. Yet they have non-negligible leakage and lower energy density. As a result, using combination of supercapacitors and batteries, i.e., hybrid energy storage is currently the state-of-the-art practice in energy harvesting systems [5][2].

Our target solar-powered embedded system is a multisensory platform supporting simultaneous multiple sensing and/or processing applications (see Figure 5.1). The system transfers data from its sensors to a server via wireless connection. The energy scavenged from solar panels is stored in a hybrid energy storage subsystem. Our model of the hybrid energy storage has a switch-based architecture, composed of a rechargeable battery and a supercapacitor (similar to [4]).

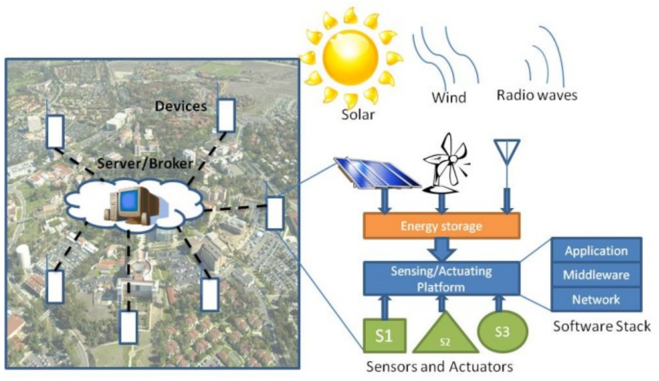


Figure 5.1 Networked Energy Harvesting

Continuous sensing/processing while handling energy harvesting variations and hybrid energy storage constraints mandate harvesting embedded systems to employ an energy storage management scheme. The management scheme not only needs to be aware of the amount of energy being harvested but also the load and energy demand from the applications. If the applications demand energy aggressively without being aware of the energy storage status, the energy storage may get exhausted and this can hurt the application quality. Continuous aggressive charge and discharge might also severely affect the health of storage components and hence shorten their lifetime. Without a proper energy management scheme, the system cannot deliver sustainable operation for a long period of time which contradicts with the objective of providing energy sustainability in systems. While adapting application quality to adjust energy consumption at middleware has been a promising solution for energy minimization and energy harvesting management, I focus on quality-aware energy harvesting management framework in concert with hybrid energy storage management. Many existing work at middleware consider the energy storage as a buffer with maximum capacity [8][9][6]. However, the power density constraint, the storage lifetime constraint, and energy efficiency in the hybrid energy storage architecture cannot be neglected. To the best of our knowledge, this is the first effort to bring hybrid-energy-storage-awareness into application quality adaptation at middleware. I propose a holistic middleware framework to orchestrate the application quality management and hybrid energy storage management for embedded systems such as multi-sensor platforms with solar energy harvesting capability.

In this chapter, I present: 1) an abstract model for hybrid energy storage with battery and supercapacitor to capture the characteristics of each storage type during charge and discharge, including battery lifetime and supercapacitor leakage, 2) an offline slot-based energy budget management to adjust application quality (QoS) according to energy availability and storage status, providing energy charge/discharge planning for each storage type with the objective of maximizing the application QoS under a given energy storage lifetime constraint, and 3) a semi-online adaptation phase captures the inaccuracy during offline approximation and prediction and dynamically adapts the charging and discharging distribution as well as the application QoS. Lastly, a controller directs charge and discharge at runtime.

In comparison with quality-aware frameworks [6][9] which do not consider the storage power density and storage lifetime, our experimental results show that such simple model can make the system fall short in realizing the expected application QoS for a long system lifetime. Our approach emphasizes that considering battery SoH (State-of-Health) and leakage of supercapacitor during offline planning along with an online scheme can help systems achieve high application QoS while reliably meeting battery lifetime constraint. In particular, this energy management framework effectively extends battery lifetime 1.6x-2x compared to [6][9].

5.2. Related work

There are several architectures proposed for energy harvesting systems with hybrid energy storage. In Chulsung and Chou [4] and Carli et. al [5], pure hardware implementation for hybrid energy storage management is proposed while in [1], various software energy management techniques are proposed on top of the hybrid energy storage. I first give an overview of hybrid energy storage architecture followed by their energy management strategy.

5.2.1 Hybrid energy storage architecture variants

Several variants of hybrid energy storage for energy harvesting embedded systems have been proposed in the literature. Chulsung and Chou[4] proposed Ambimax for multi-source energy harvesting, with each renewable energy source connected to a separate supercapacitors. The supercapacitor array bridges the gap between energy harvesting and energy consumption. A battery is used as backed up when supercapacitor array does not have sufficient energy to power the load. When supercapacitor array has extra energy, it can also charge the backed up battery. This architecture is shown in Figure 5.2.

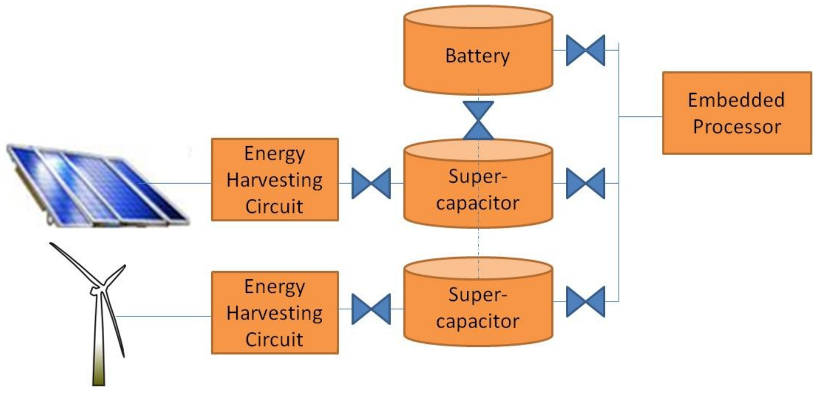


Figure 5.2 Hybrid Energy Storage Architecture Variant 1

Carli et. al[5] proposed another architecture for multi-source energy harvesting systems. Similar to [4], they have small supercapacitors connected to individual harvesting sources. Differently, they also have a large central supercapacitor called the main reservoir where all the charge from small supercapacitor flows to. The central supercapacitor is the sole source that powers the load. A back-up battery gets charged from the central supercapacitor when the supercapacitor has abundance of energy. Vice versa, the back-up batter charges the supercapacitor when harvesting is low. This architecture alternative is shown in Figure 5.3.

Lastly, [1] proposed a general hybrid electrical energy storage system architecture. This architecture uses charge transfer interconnect as a medium to connect all components, energy harvesting sources, battery arrays, supercapacitor arrays, and load. Different from the above two architectures, both battery and supercapacitor arrays can be charged by the energy harvesting sources. Similarly, both can be discharged to power the load. This architecture is shown in Figure 5.4.

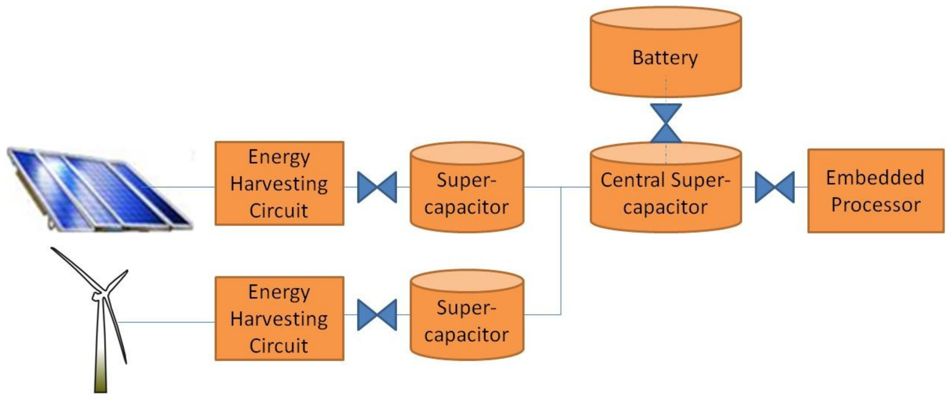


Figure 5.3 Hybrid Energy Storage Architecture Variant 2

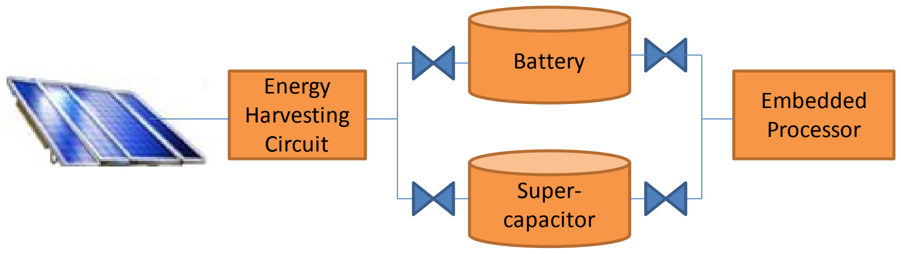


Figure 5.4 Hybrid Energy Storage Architecture Variant 3

I choose a similar architecture to the last one as it is for general purpose. The first two architectures do not use harvesting sources to charge battery directly in order to reduce battery aging effect. However, using supercapacitor to buffer energy may cause some significant energy loss due to converters between supercapacitor and batteries. Instead, with the last architecture, a software management technique can control the charging to the battery and supercapacitor while monitoring and keeping battery aging under constraint. For simplicity and efficiency, I have not considered a direct connection between battery and supercapacitor. However, it is feasible to integrate such option for energy transfer in this middleware framework for hybrid energy storage.

Note that this hybrid storage architecture is not limited to a single battery or supercapacitor. It can be a combination of multiple batteries and multiple supercapacitors. However, size constraint and efficiency must be considered carefully and the energy management will also get more complicated.

5.2.2. Energy Storage Management for Hybrid Energy Storage Architecture

In Chulsung and Chou [4] and Carli et. al [5], hybrid energy storage management is implemented in hardware using simple voltage thresholds (threshold to charge/discharge battery and supercapacitor) to avoid energy overflow and underflow. These methods are neither aware (or partially aware) of energy efficiency nor battery lifetime. Hybrid energy storage systems for automotive or mobile system have been proposed in [1][7]. In [1], their approach increases the battery lifetime by reducing battery charge average and standard deviation, and using supercapacitor as a buffer to smooth out charging and discharging current from the battery. This system architecture assumes a charge transfer interconnect between charging source (or load) and hybrid energy storage subsystem. It allows charging or discharging to/from both supercapacitor and battery at the same time However, their system avoids charging and discharging simultaneously which often happens in energy harvesting systems. [16] is an improved extension of [1]. In [7], the authors focus on the rated capacity effect (energy efficiency) of the battery. This work statically schedules discharging activities of a hybrid energy storage system and does not consider recharging activities or lifetime of batteries. Both [7] and [16] do not consider load adaptation. In my work, I dynamically adjust workload in orchestration with hybrid energy storage management to maximize application QoS while meeting storage lifetime and harvesting constraints.

Furthermore, [14] considers harvesting system with only rechargeable battery. They utilize Phase Change Memory to do load matching in energy harvesting WSNs, reducing number of charging/discharging cycles to battery. Their goal however is not to optimize application QoS. Previous works on energy budgeting for harvesting sensor networks [9][6] optimize application QoS or consumption of harvested energy. They are not aware of hybrid energy storage characteristics, neither power density constraint nor battery lifetime constraint. [6] assumes ideal storage and therefore overestimates energy availability and aggressively optimizes application QoS. As a result, it can cause significant battery lifetime degradation and shorten useful lifetime of the whole system.

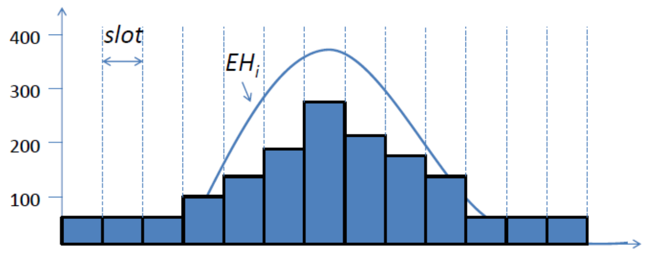


Figure 5.5 Slot-based Energy Harvesting and Budget Allocation Example

5.3. Our Proposed Framework

In this section, I present an overview of our proposed framework. The framework is for operation during a hyper-period of harvesting and application activities. For solar-powered embedded systems, I chose a day long hyper-period. Each period is then divided into n equal slots, each slot of duration ΔT. I leverage existing harvesting prediction algorithm [12] to gain knowledge about future energy harvesting in each slot. Figure 5.5 is an example of slot-based energy harvesting prediction and budget allocation. The optimization problem addressed in this multi-phase framework is formally stated as:

1) Given a set of QoS levels for each application, the hybrid energy storage characteristics, a target battery lifetime T_{life}, and predicted energy harvesting for each time slot,

2) Assign QoS level for each slot as well as charge/discharge distribution for energy storage components such that

3) The total QoS of the system is maximized.

Our framework structure is shown in Figure 5.6. Using prediction algorithm to estimate energy harvesting profile (i.e., solar) and prior knowledge on event and application activities, an offline phase is proposed to allocate charge/discharge budget to each energy storage component while maximizing the total QoS in the next hyper-period.

Our distinct contribution is that the proposed offline algorithm is not only a QoS-aware energy budget allocation but also takes into consideration the hybrid energy storage characteristics. It considers battery lifetime constraint, the leakage in supercapacitor, and efficiency of chargers and converters. Increasing the energy efficiency of the storage leads to maximizing energy availability and hence, maximizing QoS while the target system lifetime is guaranteed by considering battery status and lifetime constraint. Given the high complexity of this problem, the offline phase is driven by a server and the results are transferred to each node before the next hyper-period.

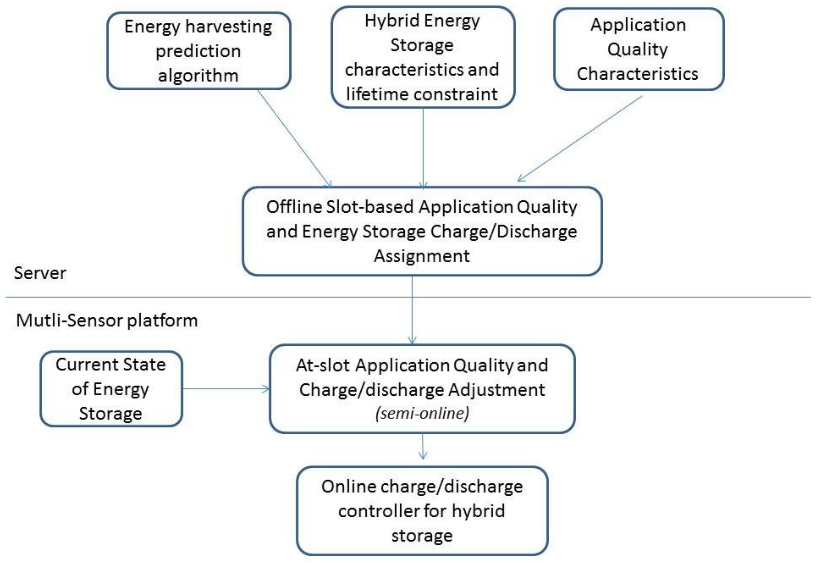

Energy harvesting prediction algorithm
Hybrid Energy Storage characteristics and lifetime constraint
Application Quality Characteristics
Offline Slot-based Application Quality and Energy Storage Charge/Discharge Assignment
Server
Mutli-Sensor platform
Current State of Energy Storage
At-slot Application Quality and Charge/discharge Adjustment (semi-online)
Online charge/discharge controller for hybrid storage

Figure 5.6 Our Proposed Middleware Framework for Energy-Storage-aware Application Quality Adaptation

These limits and bounds from the offline optimization are guidance to an online energy storage charge and discharge controller. In addition, given that energy harvesting profile is more accurately predicted with short-term prediction (i.e., extrapolating next slot energy harvesting from harvesting data of previous k slots [17]), a semi-online adaptation algorithm is deployed to correct error from offline prediction. The state of hybrid energy storage at the current time slot may not be the same as the expected value due to lack or surplus of energy in the past slots.

Therefore, the semi-online algorithm provides incremental updates on the expected QoS and charge/discharge distribution for the current slot. The proposed algorithm is lightweight and can be deployed at the middleware layer of the node.

5.4. Target System and Application Model

This section provides an overview of application quality model and hybrid energy storage model deployed in this work.

5.4.1 Application Quality Model

I assume each sensing application *j* has a multi-level quantized QoS model. Each level *k* is characterized by a configuration of application and/or system parameters represented as a tuple $<c_{jk1}, c_{jk2}, .., c_{jkm}>$ where *m* is the number of system and application parameters, and c_{jkx} is configuration value for parameter *x* at level *k* of application *j*. For example, data rate and error margin are configurable application parameters, while frame rate and resolution are configurable system parameters whose different values determine various QoS levels. The corresponding energy consumption EC_{jk} for each QoS level *k* of application *j* is determined through multiple measurement experiments, taking the average out of those measurements.

5.4.2 Hybrid Energy Storage Model

Figure 5.7 shows the hybrid energy storage architecture. I denote the output from the harvesting circuit as V_{src} and I_{src}. $\gamma_{battery}^{regulator}$ and $\gamma_{supercap}^{regulator}$ are the efficiency of the regulators controlling the charging process to the battery and supercapacitor, respectively. On the other side, the voltage and current required to power the load is denoted as V_{load} and I_{load}. The efficiency of the converters matching voltages of battery and supercapacitor with the required load voltage are $\gamma_{battery}^{converter}$ and $\gamma_{supercap}^{converter}$. State of Charge (SoC) is the percentage of charge remained in the battery, compared to the full rated capacity. SoC is updated based on charging and discharging current according to Columbus counting method.

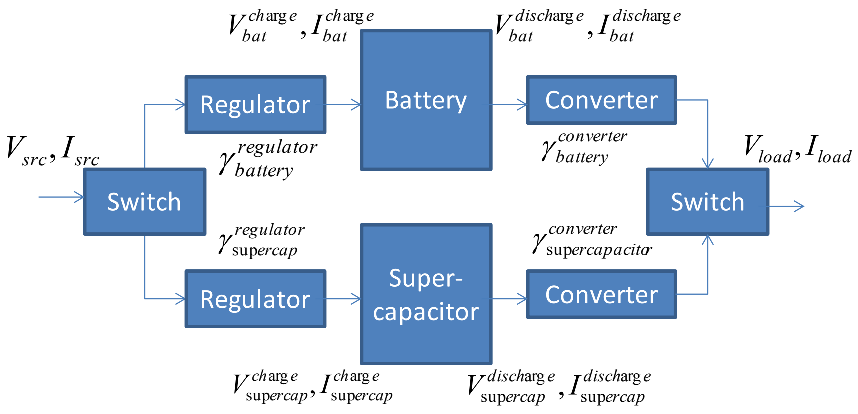

$V_{bat}^{charge}, I_{bat}^{charge}$
$V_{bat}^{discharge}, I_{bat}^{discharge}$
Regulator
Battery
Converter
V_{src}, I_{src}
$\gamma_{battery}^{regulator}$
$\gamma_{battery}^{converter}$
V_{load}, I_{load}
Switch
Switch
$\gamma_{supercap}^{regulator}$
$\gamma_{supercapacitor}^{converter}$
Regulator
Super-capacitor
Converter
$V_{supercap}^{charge}, I_{supercap}^{charge}$
$V_{supercap}^{discharge}, I_{supercap}^{discharge}$

Figure 5.7 Hybrid Energy Storage Subsystems

Battery lifetime model: Battery has a limited number of charging and discharging cycles, i.e. limited lifetime. To estimate battery lifetime, I adopt the model in [3][1]. This model allows us to use SoC average and standard deviation to estimate the State of Health (SoH) degradation. SoH degradation is a way to measure the ability of battery to store and deliver energy. This degradation gradually increases from *0* to *1* during battery lifetime, with *0* as a fresh battery and *1* as a battery without capacity.

Conventionally, a battery with a degradation of *0.2* is considered failed. SoH degradation of battery in a hyper-period *P* is computed as:

$$D(P) = (C_1 N e^{\alpha SoC_{dev}} + C_2) C_3 e^{\beta SoC_{avg}} (1 - D^s) \quad \textbf{(5-1)}$$

where C_1, C_2, C_3, α, and β are empirical constants specific to each battery. N is the number of effective charging/discharging cycles in a hyper-period P. D_s is the SoH degradation at the start of this hyper-period. The SoH degradation over multiple hyper-periods is the sum of degradation over each individual hyper-period:

$$D(kP) = \sum_{i:1\rightarrow k} D(P_i) \quad \textbf{(5-2)}$$

To meet the target lifetime T_{life}, assuming equal degradation in each period, equation below shows the constraints for SoH degradation in each hyper-period.

$$D(P) \leq \frac{0.2P}{T_{life}} \quad \textbf{(5-3)}$$

All three parameters, N, SoC_{dev} and SoC_{avg} in Equation 6-1 are dynamically adjusted in our proposed framework to meet the battery lifetime constraint in Equation 6-3.

Supercapacitor leakage model: For an accurate estimation of supercapacitor leakage energy, there are models using complex circuits with intensive calculations. I use the approximation model for supercapacitor leakage in [7] for low complexity and ease of integration.

$$P_{leakage} = \mu e^{\rho V_{sup}} \tag{5-4}$$

where V_{sup} is the voltage of the supercapacitor and μ and ρ are empirical parameters of the supercapacitor.

5.5. Algorithms for Application Quality and Energy Charge and Discharge Assignment

In this section, I first describe our offline optimal solution for slot-based QoS assignment and storage charge/discharge distribution. I then propose semi-online algorithm for QoS re-adjustment at the beginning of each slot and an online charge and discharge controller.

5.5.1 Offline Planning

I propose an optimal solution using ILP solvers. The objective is to assign each time slot i an expected application QoS and charge/discharge distribution. I summarize here the various set of constraints:

Capacity and Power Density Constraints: Let ES_i and EB_i denote the energy in the supercapacitor and the battery at the beginning of time slot *i*. To avoid overflow and underflow, ES_{max} and EB_{max} are the upper bounds; ES_{min} and EB_{min} are the lower bounds of ES_i and $\underline{EB_i}$, respectively.

$$ES_{min} \leq ES_i \leq ES_{max} \tag{5-5}$$

$$\mathrm{EB_{min}} \le \mathrm{EB_i} \le \mathrm{EB_{max}} \quad \textbf{(5-6)}$$

Similarly, the charging and discharging energy budget of the battery and supercapacitor in each slot EB

$$\mathrm{EB_{min}^{charge}} \le \mathrm{EB_i^{charge}} \le \mathrm{EB_{max}^{charge}} \quad \textbf{(5-7)}$$

$$\mathrm{EB_{min}^{discharge}} \le \mathrm{EB_i^{discharge}} \le \mathrm{EB_{max}^{discharge}} \quad \textbf{(5-8)}$$

$$\mathrm{ES_{min}^{charge}} \le \mathrm{ES_i^{charge}} \le \mathrm{ES_{max}^{charge}} \quad \textbf{(5-9)}$$

$$\mathrm{ES_{min}^{discharge}} \le \mathrm{ES_i^{discharge}} \le \mathrm{ES_{max}^{discharge}} \quad \textbf{(5-10)}$$

Remaining Charge Update: Energy in battery and supercapacitor at the beginning of slot *i+1* is updated based on its charge, discharge and leakage in slot *i*.

$$\mathrm{EB_{i+1}} = \mathrm{EB_i} + \gamma_{\mathrm{battery}}^{\mathrm{regulaor}} \mathrm{EB_i^{charge}} - \mathrm{EB_i^{discharge}} \quad \textbf{(5-11)}$$

$$\mathrm{ES_{i+1}} = \mathrm{ES_i} + \gamma_{\mathrm{supercap}}^{\mathrm{regulaor}} \mathrm{ES_i^{charge}} - \mathrm{ES_i^{discharge}} - \mathrm{ES_i^{leakage}} \quad \textbf{(5-12)}$$

Where $ES_i^{leakage}$ is the leakage energy from the supercapacitor. I use piecewise linear approximation to estimate leakage in Equation 6-4.

Energy Harvesting Constraints: The total energy charging to battery and supercapacitor in each slot, i.e., EB_i^{charge} and ES_i^{charge} is limited by the predicted energy harvesting EH_i.

$$\mathrm{EH_i} \ge \mathrm{EB_i^{charge}} + \mathrm{ES_i^{charge}} \quad \textbf{(5-13)}$$

Application Quality constraints: For each slot *i*, an application $QoS_{i,j}$ is assigned for application *j* and it must meet the minimum QoS requirement. The total energy consumption for all applications in each slot comes from the power supply, $EB_i^{discharge}$ and $ES_i^{discharge}$

$$\sum_{\forall j} \mathrm{EC}_{\mathrm{jQoS}_{\mathrm{i,j}}} = \gamma_{\mathrm{battery}}^{\mathrm{converter}} \mathrm{EB}_{\mathrm{i}}^{\mathrm{discharge}} + \gamma_{\mathrm{supercap}}^{\mathrm{converter}} \mathrm{ES}_{\mathrm{i}}^{\mathrm{discharge}} \qquad \textbf{(5-14)}$$

Battery Lifetime Constraints: Charging and discharging currents from battery, I_{bat}^{charge} and $I_{bat}^{discharge}$ are functions of battery *SoC* and the charging/discharging power.

$$\mathrm{I}_{\mathrm{bat}}^{\mathrm{charge}} = \mathrm{f}\left(\mathrm{SoC}, \mathrm{P}^{\mathrm{charge}}\right) = \frac{-\mathrm{VOC(SoC)} + \sqrt{\mathrm{VOC(SoC)}^2 + 4\mathrm{R} \times \mathrm{P}^{\mathrm{charge}}}}{2\mathrm{R}} \qquad \textbf{(5-15)}$$

where *VOC* is a function of SOC (see [3][1]) and *R* is a empirical constant of the battery. Similar function is for $I_{bat}^{discharge}$ and $P^{discharge}$. Since this is a non linear function, I use Taylor approximation to approximate charging and discharging currents from battery in each slot *i*, choosing $SoC_{pivot} = 0.5$ and $P_{pivot}^{charge} = EB_{max}^{charge}/(2\Delta T)$ as the pivot points.

$$\begin{aligned}\mathrm{I}_{\mathrm{bat,i}}^{\mathrm{charge}} = \mathrm{f}\left(\mathrm{SoC}_{\mathrm{pivot}}, \mathrm{P}_{\mathrm{pivot}}^{\mathrm{charge}}\right) + \left(\mathrm{SoC}_{\mathrm{i}} - \mathrm{SoC}_{\mathrm{pivot}}\right) \times \mathrm{f}_{\mathrm{SoC}}\left(\mathrm{SoC}_{\mathrm{pivot}}, \mathrm{P}_{\mathrm{pivot}}^{\mathrm{charge}}\right) \\ + (\mathrm{P}_{\mathrm{i}}^{\mathrm{charge}} - \mathrm{P}_{\mathrm{pivot}}^{\mathrm{charge}}) \times \mathrm{f}_{\mathrm{P}^{\mathrm{charge}}}(\mathrm{SoC}_{\mathrm{pivot}}, \mathrm{P}_{\mathrm{pivot}}^{\mathrm{charge}})\end{aligned} \qquad \textbf{(5-16)}$$

Where $P_i^{charge} = EB_i^{charge}/\Delta T$

The charging and discharging currents are used to compute effective cycles *N* and battery SoC_{dev} and SoC_{avg} as shown in the following equations.

$$\mathrm{N}_{\mathrm{i+1}} = \mathrm{N}_{\mathrm{i}} + \left(\mathrm{I}_{\mathrm{bat}}^{\mathrm{charge}} + \mathrm{I}_{\mathrm{bat}}^{\mathrm{discharge}}\right) \times \Delta\mathrm{T}/(2\mathrm{Q}_{\mathrm{bat}}) \qquad \textbf{(5-17)}$$

$$\mathrm{SoC_{i+1}} = \mathrm{SoC_i} + \left(\mathrm{I_{bat}^{charge}} - \mathrm{I_{bat}^{discharge}}\right) \times \frac{\Delta \mathrm{T}}{\mathrm{Q_{bat}}} n \tag{5-18}$$

$$\mathrm{SoC_{avg}} = \frac{1}{\mathrm{n}} \sum\nolimits_{\mathrm{i:0..n-1}} \mathrm{SoC_i} \tag{5-19}$$

$$\mathrm{SoC_{dev}} = \frac{2\sqrt{3}}{\mathrm{n}} \sum\nolimits_{\mathrm{i:0..n-1}} \left|\mathrm{SoC_i} - \mathrm{SoC_{avg}}\right| \tag{5-20}$$

N_i is the accumulated effective cycles in the first *i* slots while SoC_i is the battery SoC at the beginning of slot *i*. The battery lifetime degradation in each hyper-period is estimated from the values of *N*, SoC_{dev} and SoC_{avg} according to constraint (5-1). Substitute the Equation 5-1 into Equation 5-3 we have this battery lifetime constraint:

$$(\mathrm{C_1 N_n} e^{\alpha \mathrm{SoC_{dev}}} + \mathrm{C_2}) \times \mathrm{C_3} e^{\beta \mathrm{SoC_{avg}}} \times (1 - \mathrm{D^S}) \leq \frac{0.2\mathrm{P}}{\mathrm{T}} \tag{5-21}$$

This constraint is also nonlinear, I simplify it as followed. When $N_n \geq 1.0$, C_2 is much smaller than other components, hence it can be safely ignored. Equation 5-21 is equivalent to:

$$\log \mathrm{C_1} + \lg \mathrm{N_n} + \alpha \mathrm{SoC_{dev}} + \log \mathrm{C_3} + \beta \mathrm{SoC_{avg}} \leq \log \frac{0.2\mathrm{P}}{\mathrm{T_{life}}(1-\mathrm{D^S})} \tag{5-22}$$

When $0.1 \leq N_n \leq 1$, C_2 is estimated as $C_2 = \frac{1}{2} C_1 N_n e^{\alpha SoC_{dev}}$. Therefore, Equation 5-21 is equivalent to:

$$\log 1.5 + \log \mathrm{C_1} + \log \mathrm{N_n} + \alpha \mathrm{SoC_{dev}} + \log \mathrm{C_3} + \beta \mathrm{SoC_{avg}} \leq \log \frac{0.2\mathrm{P}}{\mathrm{T_{life}}(1-\mathrm{D^S})} \tag{5-23}$$

Objective Function: Given these energy harvesting constraints, hybrid energy storage capacity, battery lifetime constraints, and application quality constraints, the objective of the ILP is to maximize overall application QoS:

$$\text{Maximize} \sum\nolimits_{\mathrm{i,j}} \mathrm{Q_{i,j}}$$

5.5.2 Semi-online Adjustment

The predicted energy harvesting can be different from actual values due to uncertainty and change in the weather condition and workload. Battery SoC could also be different from offline prediction because of approximation error.

Hence, at the beginning of each time slot, I propose an efficient algorithm to adjust the QoS and charge/discharge distribution in energy storage. The idea is to keep (or to reduce) battery SoC as close as possible to the offline plan to meet the lifetime constraint. Extra harvesting energy if any is routed through the supercapacitor to improve QoS.

At run time, I keep track of variables related to battery lifetime including effective cycles *N* and SoC. The deviation from offline approximation is computed in each slot as follow:

$$\Delta \mathrm{N} = \mathrm{N}_{\mathrm{i}}^{\mathrm{online}} - \mathrm{N}_{\mathrm{i}}^{\mathrm{offline}} \tag{5-24}$$

$$\Delta \mathrm{SoC} = \int_{0}^{\mathrm{i}\Delta \mathrm{T}} \mathrm{SoC(t)} - \sum_{\mathrm{j:0..i-1}} \mathrm{SoC}_{\mathrm{j}}^{\mathrm{offline}} \times \Delta \mathrm{T} \tag{5-25}$$

This deviation can lead to battery lifetime constraint violation; hence it must be kept track and adjusted to stay within a safe bound. Charging and discharging activities of the battery directly affect battery's *N* and SoC. This impact is reflected in the observation below.

Increase in charge to the battery leads to increase in *N* and SoC and vice versa. Increase in discharge from the battery, however, leads to increase in *N* but decrease in SoC and vice versa. Therefore, to keep battery's *ΔN* and *ΔSoC* within an accepted bound, adjustment to charging and discharging from the battery is important. The short-term adaptation Algorithm 5-1 first detects

Algorithm 5-1: Short-term Adaptation: at beginning of slot *i*

```
Input: $\Delta\boldsymbol{N}, \Delta\boldsymbol{SoC}$
Output: Updated $QoS_i$
1. if ($\Delta N > \varepsilon N$)
2.     $\Delta I = 2Q_{bat} \times \Delta N/\Delta T$
3. elseif ($\Delta SoC > \varepsilon SoC$)
4.     $\Delta I = 2Q_{bat} \times \Delta SoC/\Delta T^2$
5. endif
6. if ($\Delta I < I_{bat,i}^{charge}$)
7.     Reduce $I_{bat,i}^{charge}$ by $\Delta I$
8.     Update $EB_i^{charge}$
9. else
10.    Reduce $\Delta I$ by $I_{bat,i}^{charge}$ and $I_{bat,i}^{charge} = 0$
11.    if ($\Delta N > \varepsilon N$)
12.        Reduce $I_{bat,i}^{discharge}$ by $\Delta I$
13.    elseif ($\Delta SoC > \varepsilon SoC$ )
14.        Increase $I_{bat,i}^{discharge}$ by $\Delta I$
15.    endif
16.    Update $EB_i^{charge}$ and $EB_i^{discharge}$
17. endif
18. $\Delta EH = EH_i - EB_i^{charge}$ – $ES_i^{charge}$
19. if (ΔEH < 0)
20.    Reduce $EB_i^{charge}$ and $ES_i^{charge}$
21. else
22.    Increase $ES_i^{charge}$
23.    Increase $EB_i^{charge}$ if $\Delta N < 0$ and $\Delta SoC < 0$
24. endif
25. Update $EB_i^{discharge}$e and $ES_i^{discharge}$
26. Update QoS = lookup($EB_i^{discharge} + ES_i^{discharge}$)
```

the cases in which battery's ΔN and ΔSoC is beyond a safe bound and in a long run, it can lead to battery lifetime constraint violation. In these cases, the system has used or has stored more energy in the battery than the offline planning. Applying the observation above, the algorithm computes the corresponding adjustment in charging/discharging currents needed to bring battery's N and SoC to be within bound (line 1-5). To achieve this target reduction, the charging current to the battery is first reduced and the charge budget for battery in slot *i* is updated (line 6-8). If the target reduction in current is not met, the discharging current from the battery is adjusted next. From the observation, the discharging current has opposite effects on battery's N and SoC. If effective cycles N has to reduce, the algorithm reduces the discharge current from the battery. If battery SoC has to reduce, the algorithm increases the discharge current from the battery instead. The budget for battery charge and discharge is updated (line 10-16).

Once the adjustment to meet battery lifetime constraint is carried out, the algorithm checks the harvesting condition (line 18). Energy harvesting for slot *i* is updated using short-term prediction algorithm [17]. If updated energy harvesting is less than the planned charge budget to battery and supercapacitor, the algorithm reduces charge budget to battery which in turn helps to reduce both battery N and SoC (line 19-20). If energy harvesting is more than what was expected, the charge budget to the supercapacitor is increased up to its limit. In addition, if $\Delta N < 0$ and $\Delta SoC < 0$, there are some slack cycles to charge more energy to the battery (line 22-23). The discharge budget from supercapacitor/battery has a corresponding adjustment (line 25). Finally I update application QoS according to the new discharge budget form supercapacitor and battery (line 26).

The proposed algorithm is an efficient algorithm that opportunistically harvests, stores and improves application QoS under battery lifetime constraint and dynamic changes in energy harvesting profile.

5.5.3 Online Charge/Discharge Controller

At run time, given the harvesting power, the controller decides whether to charge the supercapacitor or the battery. The decision is based on the guideline given by the offline and semi-online phases. The controller charges the supercapacitor first up to its charge budget, then it switches to the battery. However, it discharges from the battery first up to its discharge budget then switches to the supercapacitor. Delaying charging to the battery and expediting discharging from the battery help to reduce the battery SoC_{avg} while keeping effective cycles N the same.

5.6. Experiments

In this section, I first explain the experimental setup and then evaluate our proposed energy management middleware for hybrid energy storage harvesting systems.

I evaluated our approach under various weather conditions for a period of seven days in two typical seasons, summer and winter. I compare with existing work on adapting data quality and hybrid energy storage management and show that it is necessary to consider application QoS optimization and system lifetime constraint in an orchestrated manner in both offline planning and online phase of an energy harvesting system.

5.6.1 Experimental Setup

I implemented our hybrid energy storage management in a simulated harvesting system consisting of a *220mAhr* Li-on battery and a *50F* supercapacitor. The simulation is conducted in state-of-the-art commercial networked system simulator, Qualnet [10]. A network of wireless sensor nodes is simulated; each node has a temperature sensor, a humidity sensor, an acoustic sensor and a low-power image sensor.

I model the physical phenomenon such as temperature as a random process between a lower and upper bound, with a random variation in each step. Query model includes both periodic queries and Poisson-distribution sporadic queries. Specific settings are provided in our previous work [6].

Table 5.1 Quantized QoS Levels for Temperature Sensor QoS level

QoS level	Error margin	Energy consumption per slot (mJ)
1	7.0	37901.79
2	6.0	37950.02
3	5.0	38057.57
4	4.0	38257.28
5	3.0	38717.11
6	2.0	39876.28
7	1.0	44342.70
8	0.0	51122.13

Through extensive simulation, I quantify different levels of QoS, each with an error margin and energy consumption (see Table 5.1). Energy consumption is average value from multiple simulation runs.

The low-power image sensor has a QoS model defined by resolution and frame rate. [13] shows that different resolution and frame rate changes the video quality perceived by users.

Table 5.2 QoS Levels for Low-power Camera QoS Level Configuration Energy Consumption per Slot (mJ)

QoS level	Configuration	Energy consumption per slot (mJ)
1	15 fps, 640x480	4298
2	60 fps, 352x288	22447
3	90 fps, 320x240	39272

I select *3* modes and corresponding energy consumption shown in Table 5.2 for a lower power image sensor (Aptina [11]). Assuming power consumption per pixel is constant, corresponding power for QoS level 1-3 is *80mW*, *106mW* and *120mW* respectively. Assume it takes *2ms* to capture an image and the system is idle until the next capture.

I test our approach with solar profiles including a winter week (February) and a summer week (August) at Elizabeth city, North Carolina. Data is retrieved from National Renewable Lab website [15]. The solar irradiance is converted to harvested energy by linear conversion considering solar panel size of *30.3 cm × 20.2 cm*, solar cell efficiency of *10%* and harvesting efficiency of *80%*. The winter days in this data set consistently has lower solar irradiation and thus, lower energy harvesting than summer days. During the summer days, although the irradiation is higher, I also observe significant fluctuation within each day and very low harvesting on a rainy day.

I integrate a long-term solar energy prediction algorithm which gives slot-based harvesting prediction of the next day based on the past *3* days [12]. The inaccuracy of this per-slot prediction algorithm goes up to *40%* in our experiments. The short-term energy prediction for the next slot based on previous *3* slots is adopted from [17].

5.6.2 Experimental Results

To show the importance of orchestrating application QoS and hybrid storage management, I compare our work with two baseline approaches coupling with other existing techniques to enhance them.

- Battery-aware approach: The offline planning is only aware of energy storage capacity, similar to other work such as [6][9][8]. In addition, I adopt the charging and discharging controller in [4] to switch between battery and supercapacitor based on their voltages.
- Online-SoH approach: Inspired by the battery SoC model in 0[3], I incorporate an online SoH-aware heuristic. The heuristic keeps track of accumulated cycles N, SoC_{avg} and SoC_{dev} at run time and allows charging to the battery only if projected battery lifetime degradation based on current battery status meets the lifetime constraint.

The target lifetime in our experiments is *3* years. The metrics I use are application QoS (normalized to the maximum possible QoS in a day) and SoH degradation.

Figure 5.8 shows the results comparing application QoS and daily battery lifetime degradation of our approach with two variations above for a week in winter. Since the goal of the other two baselines is to optimize application QoS without being constrained by the battery lifetime, it is not a surprise that they have better QoS than our approach. On average, Battery-aware approach has *15%* higher QoS and Online-SoH has *14%* higher QoS than our approach.

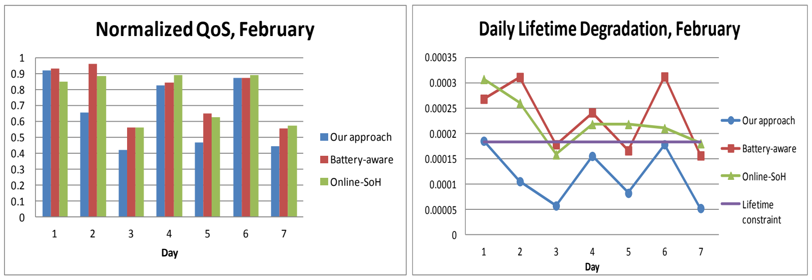


Figure 5.8 Winter QoS and Daily SoH/Lifetime Degradation

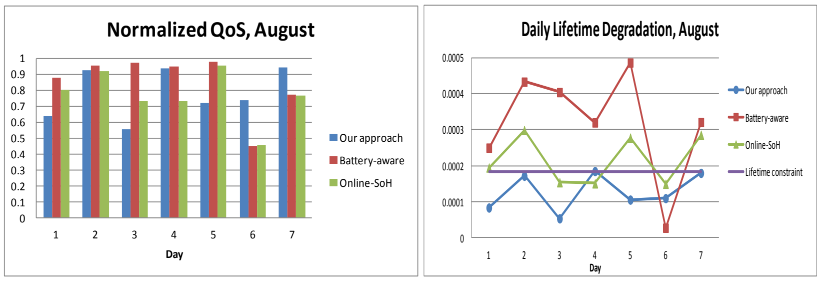


Figure 5.9 Summer QoS and Daily SoH/Lifetime Degradation

However, as shown in Figure 5.8, our approach has significant less daily SoH degradation. On average, our daily SoH degradation is *50%* lower compared to Battery-aware approach and *48%* lower compared to Online-SoH approach. The Online-SoH approach tries to reduce the daily battery lifetime degradation at runtime. However, because of the complicated correlation between SoH parameters including effective cycles *N*, average SoC and SoC deviation, online heuristics are not sufficient to guarantee battery lifetime constraints. Both Battery-aware and Online- SoH approaches often miss battery lifetime constraint as shown in the Figure 5.8. On the other hand, our approach always meets the battery lifetime constraint without degrading application QoS. This reinforces our approach of orchestrating application QoS and hybrid storage management in both offline and online phase.

Figure 5.9 shows the comparison results for a week in summer. On average, Battery-aware approach has only *7%* higher QoS compared to our approach while Online-SoH is *1%* lower in QoS than our approach. This partly comes from the sixth day of the summer week where harvesting profile is significant low while energy harvesting prediction is high.

Battery-aware and Online-SoH aggressively optimize QoS and consume large amount of energy consumption till the short-term adaption realizes the shortage in harvested energy compared to prediction. Their application QoS is dropped significantly after this and it affects the next day when energy reserve at the beginning of the seventh day is low. In terms of daily battery SoH degradation, our approach consistently has lower degradation compared to the other two. On average, our approach has *50%* less degradation than Battery-aware approach and *38%* less degradation than Online-SoH approach.

If this trend in the summer and winter continues, our approach is expected to have *2x* longer lifetime compared to Battery-aware approach and *1.6x-1.9x* longer lifetime compared to Online-SoH approach. Our approach always meets the daily SoH degradation constraint and is very likely to have useful lifetime of more than *3* years. On the other hand, the average daily SoH degradation of Battery-aware and Online-SoH approach suggests that their system can only sustain *1.7-2.5* years.

5.7. Conclusion

In this chapter, I propose a two-phase hybrid energy storage management and application quality adaptation for harvesting capable devices. The offline phase plans energy charge and discharge based on prediction of energy harvesting profile in order to maximize application QoS while meeting energy storage lifetime constraint. The online phase implements a heuristic to control and adapt charging, discharging activities to the hybrid energy storage. Our experiments prove that offline planning and online adaptation tightly coupled with knowledge of hybrid energy storage is crucial to realize application quality optimization on a sustainable harvesting device.

References

[1] X. Qing et. al, "State of health aware charge management in hybrid electrical energy storage systems." In DATE 2012.

[2] M. Pedram, N. Chang, Y. Kim, and Y. Wang, "Hybrid Electrical Energy Storage Systems", in ISLPED 2010

[3] A. Millner, "Modeling Lithium Ion Battery Degradation in Electric Vehicles", in CITRES 2010.

[4] P. Chulsung, P. H. Chou. "Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes." In SECON 2006.

[5] D. Carli, D. Brunelli, L. Benini, M. Ruggeri, "An effective multi-source energy harvester for low power applications", in DATE 2011

[6] N. Dang, E. Bozorgzadeh, N. Venkatasubramanian "QuARES: Quality-aware data collection in energy harvesting sensor networks", in IGCC 2011

[7] A. Mirhoseini, F. Koushanfar, "HypoEnergy: Hybrid supercapacitor-battery power-supply optimization for Energy efficiency", in DATE 2011

[8] D. K. Noh, L. Wang, Y. Yang, H. K. Le, and T. Abdelzaher, "Minimum Variance Energy Allocation for a Solar-Powered Sensor System", in DCOSS 2009

[9] C. Moser, J.-J. Chen, and L. Thiele, “Power management in energy harvesting embedded systems with discrete service levels”, in ISLPED’09

[10] Qualnet, http://web.scalablenetworks.com/content/qualnet

[11] Aptina MT9V11, http://www.aptina.com

[12] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, V. Raghunathan, “Adaptive Duty Cycling for Energy Harvesting Systems,” in ISLPED 2006

[13] S. Mohapatra et. al. “Integrated power management for video streaming to mobile handheld devices”. In ACM Multimedia 2003

[14] Ping Zhout, Youtao Zhang, Jun Yang, “The Design of Sustainable Wireless Sensor Network Node using Solar Energy and Phase Change Memory”, in DATE, 2013

[15] National Renewable Lab, http://www.nrel.gov/midc/

[16] Y. Wang et. al, “Minimizing State-of-Health Degradation in Hybrid Electrical Energy Storage Systems with Arbitrary Source and Load Profiles”, in DATE 2014.

[17] J. Recas Piorno, Carlo Bergonzini, David Atienza, and T. Simunic Rosing. "Prediction and management in energy harvested wireless sensor nodes." In Wireless VITAE 2009.

Chapter 6 A Unified Stochastic Model for Energy Management in Solar-Powered Embedded Systems

Uncertainties in energy harvesting, non-ideal characteristics of harvesting circuits and energy storage (battery or supercapacitor), and application demand dynamics add more complexity to a system. In this chapter, I present a unified model based on discrete-time Finite State Markov Chain to capture the dynamicity and variations in both energy supply from solar irradiance and energy demand from the application. The probabilistic models for energy harvesting and application state capture the variations of physical environment which the embedded system interacts with through a set of Finite State Markov Chains, enabling a complete cyber physical model and closed loop control. In this chapter, I exploit the temporal and spatial prediction in solar energy and propose a deterministic profile with stochastic process to reflect the fluctuation due to unexpected weather condition. Energy harvesting circuit characteristics, the leakage of the energy storage, and efficiency loss in voltage regulators are captured during transition among the states in this model. Optimal policy to maximize expected total QoS is derived from the presented model using probabilistic dynamic programming approach. Compared to state-of-the-art deterministic energy management framework, our proposed approach outperforms in term of QoS and energy sustainability (with less shutdown time) of the system.

6.1 Introduction

To achieve energy sustainability in embedded systems, energy harvesting technology is emerging as a promising solution to provide perpetual energy for systems to operate autonomously. However, design and operation of such systems have faced various challenges. This chapter focuses on proposing a formal model and optimal energy management policy at software level to address the complexity of such systems more efficiently.

Renewable energy sources such as solar energy often exhibit both temporal and spatial variations, which cause uncertainty in energy availability. There is an increasingly growing research to characterize the solar energy's spatial and temporal variability, mostly exploiting its diurnal and seasonal patterns to extrapolate its future availability [7-9]. However, unexpected fluctuations caused by the stochastic variability of atmospheric conditions are hard to capture effectively, leading to errors in prediction results.

Furthermore, an energy harvesting system requires dedicated circuitry and components including harvester (e.g., solar panel), MPPT circuit, and energy storage (supercapacitor or rechargeable battery). Supercapacitors have exponential leakage while battery lifetime is challenged by frequent charge and discharge and their peaks. DC-DC converters have varying efficiency depending on supercapacitor voltage and load voltage and current [13]. Energy harvesting/storage management schemes need to account for energy efficiency and the significant loss in energy harvesting storage and circuitry.

On the other hand, there is variation in energy consumption (or load) due to unpredictable behavior of physical environment/phenomenon that applications are monitoring or interacting with. For example, in smart camera systems or sensor platforms monitoring smart spaces, it is hard to predict when important events happen. When such events occur, energy management tools should enable more computation tasks and data processing/transfer in order to meet the application quality requirements.

In this chapter, I present a unified model to enable the orchestration and tight integration between energy harvesting/storage management and energy consumption management. This unified model based on discrete-time Finite State Markov Chain captures the dynamicity and variations in both energy supply from solar irradiance and energy demand from the application. System states enclose energy harvesting rate (Energy Harvesting process), energy status of the system (Energy Storage process), and energy consumption required to meet the current application QoS (Application process). To the best of our knowledge, this is the first formal unified model for solar-powered supercapacitor-based embedded systems using Finite State Markov Chain to model the dynamics of the systems.

In Energy Harvesting process, I consider patterns in solar energy as well as variation in energy availability. Instead of fully stochastic model or solely deterministic model based on prediction, I propose a model based on deterministic profile integrated with a stochastic process to characterize the energy harvesting rate in the system. The model is a multi-layer parameterized

Markovian model capturing the dynamics of solar energy behavior due to weather change. In this chapter, I target supercapacitor-based energy storage system. In the Energy Storage process, I consider the non-linear leakage in the supercapacitor as well as the varying efficiency of DC-DC converters. For Application process, I determine various states of application in response to events (e.g., event monitoring vs. event processing) and model the energy consumption in system depending on the QoS level of the application in the current state. While Energy Harvesting process and Application Process are independent, the behavior of energy storage process is tightly coupled with the other two processes. Our proposed unified model enables to fuse and capture the variations in solar energy and application QoS together with complex characteristic of energy harvesting circuitry and storage.

Based on our proposed stochastic model, I develop a dynamic-programming-based algorithm to find an optimal policy in order to stochastically maximize expected performance, measured by reward associated with application QoS levels and states. Our proposed model inherently aims for continuous operation and hence, avoids shutdown time and results in better performance. I applied our model and optimal policy on a solar-powered smart camera system (camera with built-in embedded processors for image processing). Compared to deterministic method [15] using prediction of solar energy, our proposed method captures and quickly responds to uncertainty (in Energy Harvesting and Application processes) more effectively and hence, provides a higher QoS reward with fewer or no shutdown time.

6.2 Related Work

A significant amount of research for dynamic power management in non-harvesting (non rechargeable battery-powered) embedded systems is undergoing, that use stochastic model and optimization methods. [1, 2] presented Partially Observable Markov Decision Process based framework for energy savings. Their work captures the manufacturing process and temperature variations, and the uncertainty of identifying or predicting processor state. [1] mainly targeted energy saving while [2] extended it to consider serving requests and stabilize request queue occupancy.

A considerable number of work in energy harvesting communication systems, wireless sensor networks, and body area networks have also considered Markovian Model [3-6]. [3] proposed both a stationary model and generalized (non-stationary) Markovian model for piezoelectric and solar energy. [5] proposed a simple Markov chain for energy harvesting wireless sensor networks in which harvesting has only two states (active and inactive) and assuming harvesting power is equal to load power. [4] applied Lyapunov optimization technique to maximize utility of communication channels. In these work, the harvesting process is modeled as a fully random process or a stationary Markov Chain for a short period of time. Furthermore, the model of the energy storage and underlying hardware components are too simplistic or simply neglected.

There are approaches for energy management in harvesting systems that do not rely on stochastic model but prediction of harvesting in the future. Exploiting the diurnal and seasonal patterns of solar irradiance profile, prediction algorithms of energy harvesting based on Moving Average have been proposed [7-9, 19]. Because of variations in solar profile and uncertainty of weather, it is inherent that these prediction algorithms have error. In our measurement, prediction algorithms can have up to *10-30%* inaccuracy. Deterministic approaches [10,11,15] that are based on harvesting prediction will suffer from misplanning because of prediction inaccuracy, either resulting in system shutdown or energy harvesting under-utilization. Other instantaneous approaches rely on near future prediction [12] or involve optimization techniques at runtime that have high overhead [13]. Albeit adapting quickly to variations, these runtime technique might not obtain optimal result due to its local optimization nature.

6.3 Overview of Proposed Energy Management Framework

Figure 6.1 shows the overview of a solar-powered embedded system node with proposed energy management middleware framework as a software component running on top of the embedded processor. The embedded system is powered by solar energy harvested through a solar panel, which transforms solar irradiance to electrical energy. An energy harvesting circuit controls the operation of the solar panel and maximizes its efficiency by setting the optimal solar panel operating voltage according to a Maximum Power Point Tracking (MPPT) method [16]. The generated energy is stored in a supercapacitor whose terminal voltage varies according to its energy storage status. Therefore, to avoid degrading solar panel efficiency, a DC-DC converter isolates solar panel from the supercapacitor. Similarly, another DC-DC converter is used to match supercapacitor voltage with the required operating voltage of the embedded system's processor.

The terminal voltage of the supercapacitor changes according to its energy storage status, and hence, the efficiency of DC-DC converters can widely vary.

The upper half of Figure 6.1 shows the main components of proposed energy management framework. A unified model for solar-powered embedded systems is built from a probabilistic model for energy harvesting, a probabilistic model for application state and a model for the harvesting circuit and energy storage. The probabilistic models for energy harvesting and application state capture the variations of physical environment which the embedded system interacts with through a set of Finite State Markov Chains, enabling a complete cyber physical model and closed loop control. It also models the non-ideal behaviors of hardware components such as DC-DC converters and supercapacitor. Each of them will be presented in detail in section 4.

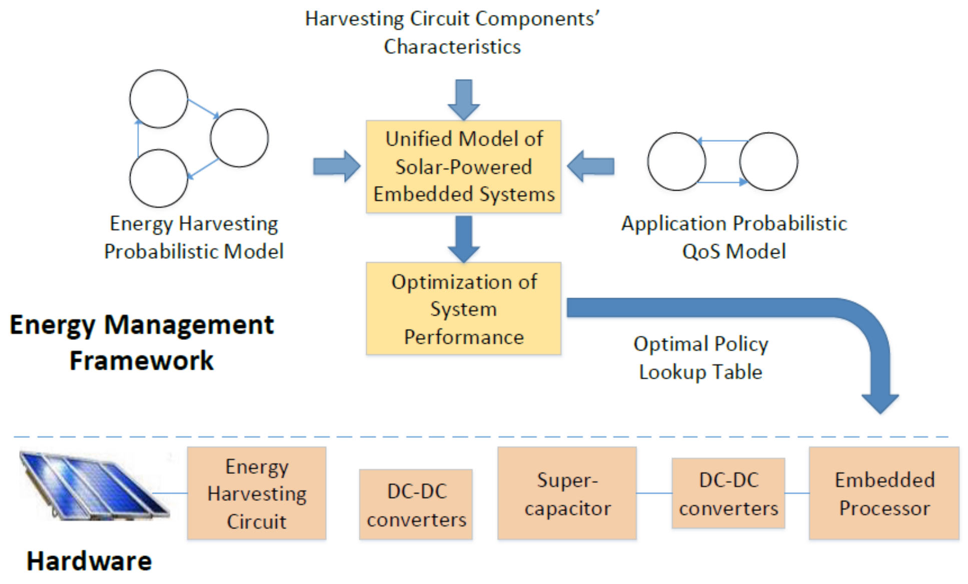

Harvesting Circuit Components' Characteristics
Unified Model of Solar-Powered Embedded Systems
Energy Harvesting Probabilistic Model
Application Probabilistic QoS Model
Optimization of System Performance
Energy Management Framework
Optimal Policy Lookup Table
Energy Harvesting Circuit
DC-DC converters
Super-capacitor
DC-DC converters
Embedded Processor
Hardware

Figure 6.1. Overview of Energy Management Framework for Solar-Powered Supercapacitor-based Embedded Systems

The unified model for solar-powered embedded system is input to a system performance optimization algorithm based on Markov Decision Process. The system performs actions subject to available QoS levels and application state. Each action completed earns a reward, representing system performance. The goal of the optimization is to maximize the expected performance in the next harvesting period (which is a day for solar-powered systems) given the dynamics of harvesting, application, and characteristics of circuit and energy storage. The output of this optimization phase is an optimal policy lookup table which guides the system to take the right action once the actual harvesting state, application state, and energy storage state are detected at run time.

As opposed to deterministic planning approaches [15], our framework does not assume the exact energy harvesting availability nor application QoS state in the future but rather their probabilities and distributions. Actions are not determined at the time of planning, instead the best action are picked at every control time unit to quickly adapt to variations in harvesting and application QoS. Yet the optimal policy guarantees to maximize the expected total system performance in the long run.

In section 6.4, I formally define each component of the system model, and describe our optimization solution using Markov Decision Process in section 6.5.

6.4 A Unified Model for Solar-Powered Embedded Systems

The unified model for the system is a Finite State Markov Chain defining system states and their probabilities of transiting from one state to another. Each system state contains information about current state of harvesting process, application process, and energy storage. Formally I define a system state as a tri-tuple $<H_k,S_k,Q_k>$ where H_k is the harvesting state, S_k is the energy storage state of supercapacitor, and Q_k is the application state at time k. Slotted time is assumed where the considered harvesting period is divided into N slots of duration T. The duration T of each time epoch is small enough for the system state to be stable. The length of T therefore depends on the time granularity of harvesting process, application process, and energy storage which altogether determine how often the system should react. After each duration T, the system may evolve in to a new system state and new action must be taken to respond to harvesting, energy storage, or application state changes (as illustrated in Figure 6.2).

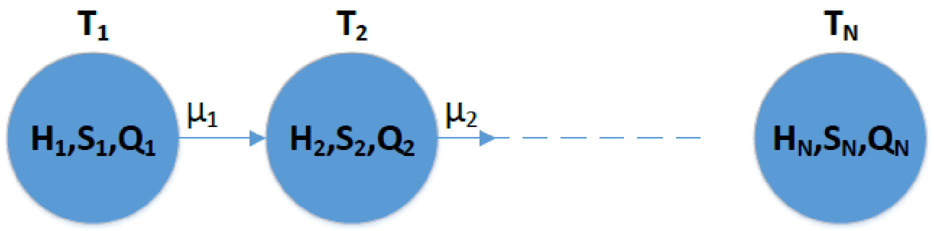


Figure 6.2. System State Evolution in a Harvesting Period

The transition probability from state $<H_k,S_k,Q_k>$ to $<H_{k+1},S_{k+1},Q_{k+1}>$ is

$$\begin{aligned} &Pr_{<H_k,S_k,Q_k><H_{k+1},S_{k+1},Q_{k+1}>}(\mu_k) = \\ &Pr\{S_{k+1} \mid H_k, S_k, Q_k, \mu_k\} \times Pr\{H_{k+1} \mid H_k\} \times Pr\{Q_{k+1} \mid Q_k\} \end{aligned} \quad \textbf{(6-1)}$$

The harvesting process and application process evolve independently. However, the energy storage process of supercapacitor evolution depends on these two processes, the circuit and energy storage characteristics, and the action taken at each time unit T. These are reflected in the probability of transitions between the system states.

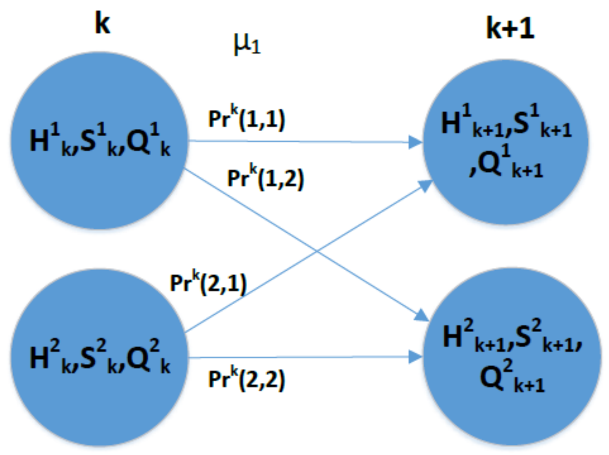


Figure 6.3. An Example of Non-stationary Markov Chain with System States and Transition Edges

This model is a non-homogeneous Markov Chain as the state transition probability is a function of time index *k*. Figure 6.3 shows an example of system state at time *k* moving to system states at time *k+1* with different probabilities, assuming there are *2* possible system states. Next, I present details on modeling of the processes.

6.4.1 Harvesting Process

The solar irradiance profile is composed of a deterministic profile with some fluctuations due to weather (as illustrated in Figure 6.4). The deterministic profile can be built by astronomical model (for e.g., [17]) based on solar panel efficiency, orientation, longitude, latitude, air/pollution attenuation level, daily shadow effects from static objects such as building, trees, and under typical weather condition and temperature. This deterministic profile is represented as $\{I_1, I_2,\ldots I_N\}$ where I_k is the typical solar irradiance at time *k* during a day and *N* is the number of slotted time in a day.

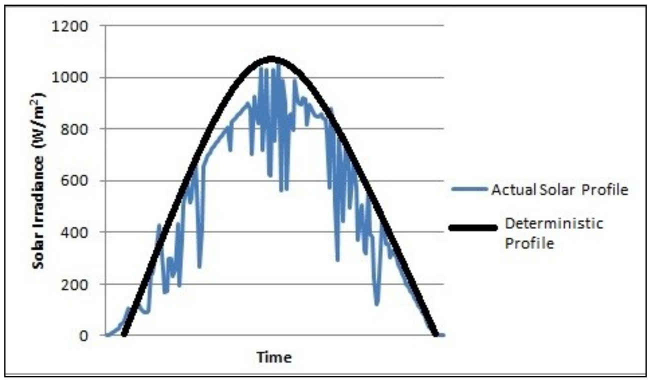


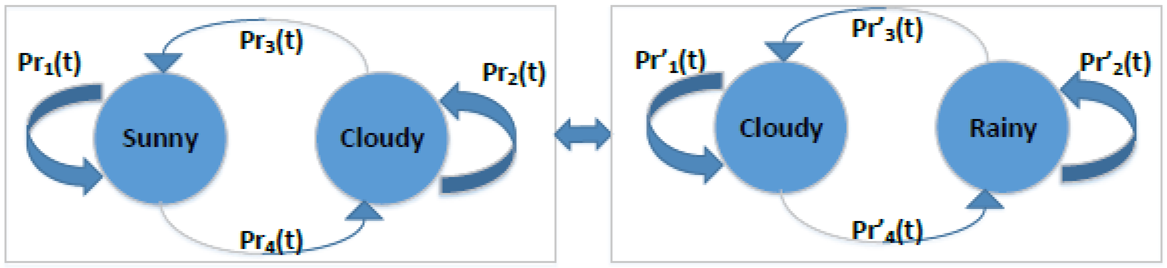
Figure 6.4. Solar Profile Combined of a Deterministic Curve and Fluctuations

The dynamic fluctuations in harvesting profile is captured by a weather process which models weather condition such as sunny, cloudy, and rainy. This weather process is denoted as $\{W_k\}$. Each weather state is associated with an attenuation level, i.e. $A(W_k)$ denoting the effect of weather on actual harvesting. The Markov Chain model for weather process (see Figure 6.5) is non-stationary as the transition probability $Pr^k\{ W_{k+1} \mid W_k \}$ can change over time. It is possible to have multiple Markov Chains for weather process, such as one for a normal day, and another for a rainy day. Selection of the Markov Chain for weather process can be based on weather forecast or by another stochastic process.

These Markov Chains for weather process, attenuation levels, and state transition probability matrix can be trained from the history of harvesting profile at a location. It can also be updated each day according to the weather forecast. References on modeling harvesting process using Markov chain can be found in the literature such as [3].

Given a weather state at time *k*, the harvesting irradiance can be computed as

$$H_k = I_k \times (1 - A(W_k)) \qquad \textbf{(6-2)}$$



a) Normal day b) Rainy day

Figure 6.5. Weather Process Markov Chains

Given the solar irradiance H_k, the output voltage and current of the solar panel, $V_{solar}(H_k)$ and $I_{solar}(H_k)$ at its maximum power point can be obtained by profiling the solar panel operation or by analytical model [17].

6.4.2 Application and Action Processes

Application state changes in response to physical world. For example, application state of a smart camera system can be regular monitoring vs. event processing (after a detected event occurrence). The actions (tasks to perform) can be different in each of these states. For example, low resolution images can be accepted in regular monitoring state while higher resolution images are more desirable after an event such as motion or human is detected. In event processing state, the application may need to perform more computational tasks such as face detection or object contour detection that consumes more energy than in regular monitoring state. The QoS levels and energy consumption in each mode are therefore different.

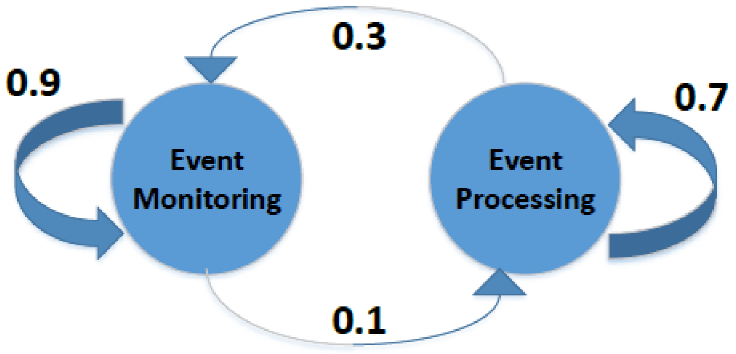


Figure 6.6. Application Process Markov Chain

The application process has a stochastic nature because it is hard to predict exactly when and for how long events will happen. Therefore, I use a Finite State Markov Chain model to capture the application process which is denoted as $\{Q_k\}$. This model captures the correlation over time that is typical of the physical event dynamics. Figure 6.6 shows an example of such model where the system has two states: event monitoring and event processing. The transition edges from one state to another state are associated with probability of event occurrence leading to application state changes.

A simplified version of this model is applications with a single state [10-11], which assume systems always respond in the same way.

In each application state i, I assume the system has a list of available QoS levels $\{QoS_{i1}, QoS_{i2},....QoS_{iM}\}$. Each QoS level represents a different set of actions to take in response to physical environment in the current application state i. Each QoS_{ij} level is associated with a required operating voltage and current $V_{load}(QoS_{ij})$, $I_{load}(QoS_{ij})$ of the embedded processor, and a reward $R(QoS_{ij})$. I assume higher QoS level has higher energy consumption and higher reward as it improves the application's accuracy or quality.

Action process is denoted as *{μ_k}* , $\mu_k \in 0 \cup \{QoS_{i1},...QoS_{iM}\}$ given $Q_k=i$, where *0* represent the possibility of shutdown due to energy outage or as a controller decision. Action is decided by the embedded processor at each time epoch. Action can be any of the QoS levels in the current application state, provided that system has sufficient energy storage to supply the action energy demand. By completing an action, the system gains a reward associated with that action.

6.4.3 Energy Storage Process

In case of battery-less system, the energy storage is a (an array of) supercapacitor. Given the nominal capacity *C*, the maximum energy that can be stored in supercapacitor is

$$E_{\max} = \frac{1}{2} C V_{\max}{}^2 \qquad \textbf{(6-3)}$$

where V_{max} is the maximum rating voltage of the supercapacitor. I choose to represent energy storage process of the supercapacitor by its voltage as it has direct relation with energy as shown in Equation 6-3. Let S_k be voltage of the supercapacitor at time slot k. S_k is updated in each slot according to its current value, harvesting process, application process, and actions taken. In addition, S_k is affected by supercapacitor leakage and converter losses as explain next.

Supercapacitor leakage: The advantages of supercapacitor as compared to battery are higher power density and no aging effect. However, supercapacitor has non-ideal behavior which is leakage that grows exponentially with its voltage. The leakage of the supercapacitor can be approximated as

$$P_{leakage} = \alpha e^{\beta V} \qquad \textbf{(6-4)}$$

where α and β are empirical constant [14].

DC-DC converter efficiency: Because the voltage of the supercapacitor varies widely according to its energy, the circuit needs DC-DC converters to match supercapacitor voltage with required output voltage and current. These converters for charging and discharging could work in either buck mode or boost mode, i.e. reducing or increasing the voltage output of the supercapacitor to match with the optimal voltage for MPPT of solar panel or required operating voltage of the embedded processor.

$$g_{charge}(H_k,S_k) = f_{charge}(S_k, V_{solar}(H_k), I_{solar}(H_k)) \tag{6-5}$$

$$g_{discharge}(S_k,\mu_k) = f_{discharge}(S_k,V_{load}(\mu_k), I_{load}(\mu_k)) \tag{6-6}$$

The power loss due to converter is a function of input voltage, output voltage, and output current. In case of charging, it is a function of S_k, V_{solar}, and I_{solar} as in Equation 6-5 In case of discharging, the power loss is a function of S_k, processor voltage V_{load} and its current I_{load} as in Equation 6-6. For detailed formulation of power loss, the readers are referred to previous work in literature [13]. Figure 6.7 shows that the loss due to converter is significant, ranging from *20%* to *80%*. The larger the gap between input and output voltage of a converter, the higher the loss is.

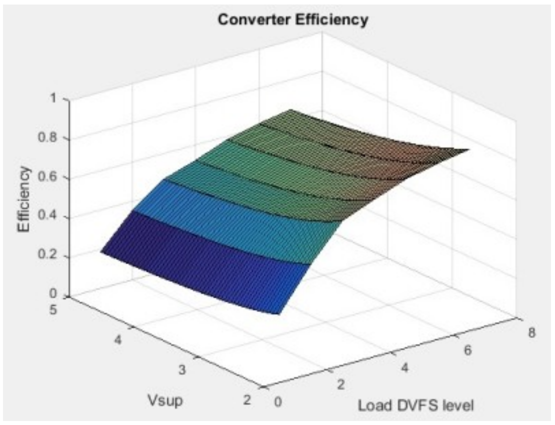


Figure 6.7. Converter Efficiency

Energy storage update and quantization: Formally, the next state S_{k+1} is computed according to Equation 6-7 to 6-9 below.

$$\begin{aligned} P_k = V_{solar}(H_k)\times I_{solar}(H_k) - V_{load}(\mu_k)\times I_{load}(\mu_k) - P_{leakage}(S_k) \\ -g_{ch\arg e}(H_k, S_k) - g_{disch\arg e}(S_k, \mu_k) \end{aligned} \quad \textbf{(6-7)}$$

$$E_{k+1} = \frac{1}{2}CS_k^{\ 2} + P_k T \quad \textbf{(6-8)}$$

$$S_{k+1} = \sqrt{\frac{2E_{k+1}}{C}} \quad \textbf{(6-9)}$$

To evaluate exactly the next system state and transition probability, there is a large amount of information to keep track of. Therefore I propose to quantize the energy storage state of the supercapacitor, i.e. its voltage to enable a Finite-State Markov Chain representation.

The supercapacitor voltage range [*0..V$_{max}$*] is partitioned into K non-overlapping intervals using *K-1* thresholds $\{0, \upsilon_1, \upsilon_2, ..., \upsilon_{K1-1}, V_{\max}\}$. The voltage range $[v_i, v_{i+1}]$ in interval *ith* is denoted as *intv(i)*. Steady state probability of energy storage state *i* is

$$\pi_i = \int_{\tau_i}^{\tau_{i+1}} f(s)ds \quad \textbf{(6-10)}$$

where *f(s)* is probability density function of voltage.

The transition probability that system is moving from state *<H$_k$,E$_k$,Q$_k$>* at time *k* to state *<H$_{k+1}$,E$_{k+1}$,Q$_{k+1}$>* at time *k+1*, given the action μ_k is

$$Pr_{<H_k,S_k,Q_k><H_{k+1},S_{k+1},Q_{k+1}>}(\mu_k) = \frac{\int_{\operatorname{int} v(H_k)} f_S(s) \int_{\operatorname{int} v(H_{k+1})} f_S(S_{k+1} \mid H_k, S_k, Q_k, \mu_k) ds ds}{\int_{\operatorname{int} v(H_k)} f_s(s) ds} \times Pr\{H_{k+1} \mid H_k\} \times Pr\{Q_{k+1} \mid Q_k\} \quad \textbf{(6-11)}$$

Numerical computation of this double integral is time consuming. Therefore, I use Monte Carlo simulation method to obtain the state transition probability. The simulation method is briefly described below:

- For each state $<H_k,S_k,Q_k>$ and action μ_k, a long sequence of supercapacitor voltage in the range of $intv(S_k)$, and simulated output state S_{k+1} according to Equation 6-7 to 6-9 are generated.
- For each state input S_k, the number of occurrences that state output belongs to interval of state S_{k+1}.is found. Then this number is normalized by the number of occurrences in state input. This is P$r\{S_{k+1}|H_k,S_k,Q_k, \mu_k\}$.

6.5 Optimal Policy to Maximize Expected System Performance

In the previous section, I presented the unified Markov Chain model for solar-powered embedded systems. In this section, I discuss an optimization framework based on dynamic programming to maximize expected rewards.

Each time epoch, the system needs to make decision as to which action to take. The actions are possible QoS levels in the current application state. The selected action dictates the embedded processor to perform certain processing tasks. The cost is the corresponding energy consumption of the action that changes the energy storage state of the system. In return the system gains a reward which is accumulated over the harvesting period, representing the system performance in that period. Since a current action can change the energy storage, careful decision making is required to avoid energy outage.

The goal of the optimization is to maximize the expected reward associated with actions taken each time epoch in the next harvesting period *N.*

$$Maximize \sum_{N} E\{R(\mu_k)\} \tag{6-12}$$

6.5.1 Optimal Policy

The optimization problem can be solved using a back-ward probabilistic dynamic programming for finite horizon [18]. The result is a policy that maps a system state to an action that maximizes the expected total reward given all the variations in harvesting process or application process.

I denote $J_k(H_k, S_k, Q_k)$ as the maximum expected total reward from time slot *k* to *N*.

$$J_{N+1}(H_{N+1}, S_{N+1}, Q_{N+1}) = 0 \tag{6-13}$$

$$J_k(H_k, S_k, Q_k) = \max_{\mu}\{\mathrm{R}(\mu) + E(J_{k+1}(H_{k+1}, S_{k+1}, Q_{k+1}))\} \tag{6-14}$$

The optimal action that maximizes $J_k(H_k, S_k, Q_k)$ in Equation 6-14 is saved in a table, called optimal policy look-up table. At run time, the system keeps track or detects the current system state and picks the right action using this optimal policy look-up table.

Complexity: The running time of this dynamic programming is $O(N|\mu||H|^2|S|^2|Q|^2)$ where $|\mu|$ is the number of actions, |H| is the number of harvesting states, |S| is the number of quantized superpcapacitor voltages, and |Q| is the number action modes.

As shutdown (μ_k =0) is undesirable in harvesting systems, I associate high reward to actions the system can take and no reward if energy storage is not sufficient to actuate any action. This inherently makes dynamic programming to choose actions that lead to minimal shutdown. In addition, at the end of each day, the system needs to maintain certain level of energy storage to start the system at the beginning of next day when harvesting is low. Equation 6-13 encourages the dynamic programming to utilize all energy to optimize system performance. Instead, I modified Equation 6-13 to associate some reward incentive π to system states whose S_N is greater or equal to minimum energy storage at the end of a day, as shown in Equation 6-15.

$$J_{N+1}(H_{N+1}, S_{N+1}, Q_{N+1}) = \pi \tag{6-15}$$

The optimal policy lookup table is only needed to be computed one time and it only needs updates when harvesting process or application process change their states and probabilities. The overhead is therefore small. Furthermore, the optimal policy look-up table enables the system to react quickly to dynamic changes in harvesting and application processes at run time, yet maximizes expected system performance in the long run.

6.6 Experimental Results

To demonstrate the effectiveness of our model and optimization framework, I implement a simulator in Matlab to model the solar-powered embedded system. The experimental setup is described in section 6.6.1 followed by our results in section 6.6.2.

6.6.1 Experiment Setup

I assume that the system has two supercapacitors with *400F* capacitance each, $V_{max}=5V$. In the Monte-Carlo simulation (section 6.4.3), the voltage of the supercapacitor is quantized into *10* equal intervals. The empirical constants for supercapacitor leakage are $\alpha=1.26e\text{-}10$ and $\beta=10.43$ according to measurement in [14]. The parameters for DC-DC converters are obtained from [13] and from the datasheet of the corresponding buck-boost converters. The required voltage at the end of each day is set to be *3.5V*.

I assume the load processor is similar to PXA270 with V_{dd} of *1.55V* and load current is set according to QoS level. I simulate a smart camera application which captures images by a number of frames per second (set by QoS level). For each frame, it performs image processing tasks such as background subtraction (in event monitoring state) and object contour detection (in event processing state).

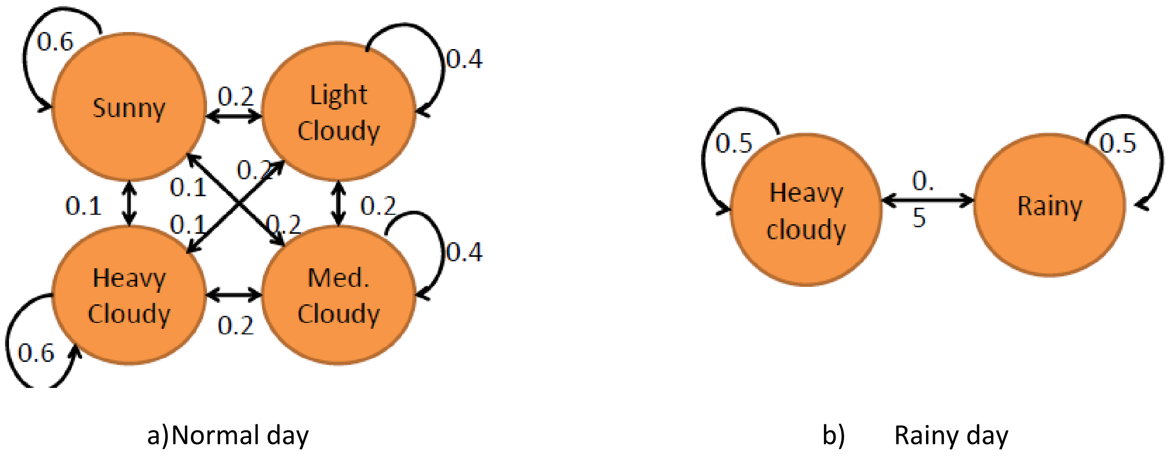
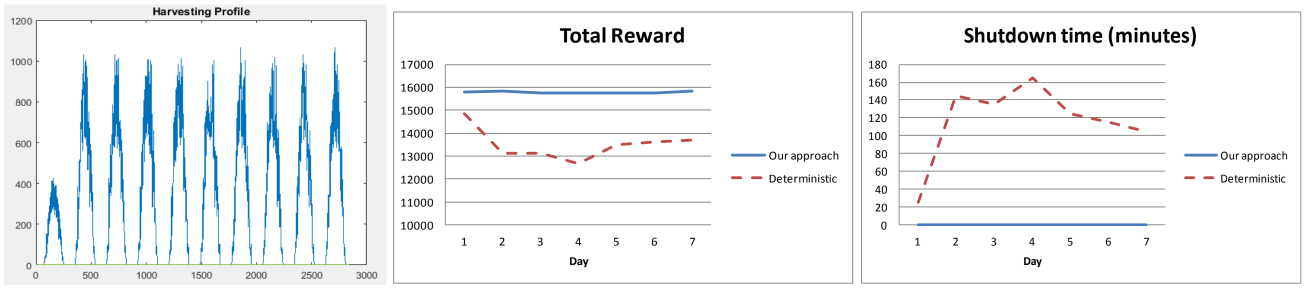


Figure 6.8. Weather Process Markov Chain



a) Harvesting Profile b) Total Reward Comparison c) Shutdown time comparison

Figure 6.9 Comparison with Deterministic Approach for a Week, Single Application

I proposed a general model of harvesting process in section 6.4.1. For the experiments, I assume to have a harvesting probabilistic model as shown in Figure 6.8. Figure 6.8a is Markov Chain for weather process during a normal day and Figure 6.8b is Markov Chain for a rainy day. The attenuation in each mode and the transition probability between weather states are denoted in each state and on each edge. This weather process model can be obtained from training the parameters with real data. From this model and a deterministic profile in Figure 6.4, I randomly generate harvesting profiles as shown in Figure 6.9a, Figure 6.10a and 6.10b.

Since there is no direct related work for comparison, I adapt a related work in energy management for supercapacitor-based energy harvesting systems [15]. This work aims to maximize duty cycling on a harvesting sensor node. I adapt it to maximize total rewards of the system. It relies on energy harvesting prediction to plan activities for the next harvesting period. In order to consider DC-DC converter efficiency, it quantizes the voltage of the supercapacitor into L intervals ($L=100$ in our experiments). A dynamic programming is then employed to plan duty cycling (QoS and reward in our adaptation) for the next harvesting period, considering the predicted energy harvesting, supercapacitor voltage, and DC-DC converter efficiency in each slot.

It however considers single application state and energy consumption for each QoS level is fixed (a special case of our general application process model in section 4.2). I call this Deterministic approach.

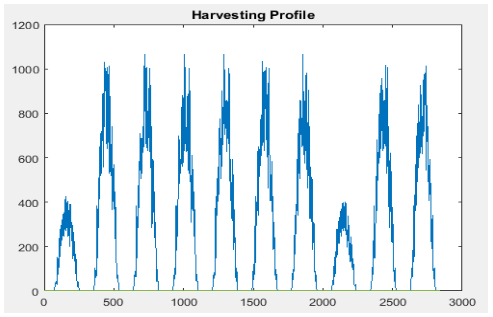
6.6.2 Results

In the first set of experiments, I assume the system has only one application state. The system has *7* QoS levels with corresponding current (in *mA*) *{51.765, 128.889, 242.609, 312, 422.222, 515.172, 596.774}*. The rewards for each QoS levels are *{100, 110, 120, 130, 140, 150, 160}*. Figure 6.9 shows the comparison of our approach against the Deterministic approach. The metrics used are total reward which reflects system performance and shutdown time which reflects system sustainability. Shutdown time is the duration during which the supercapacitor voltage falls below the minimum operating voltage. During the shutdown time, the processor is not powered but the supercapacitor is still charged by harvesting power if any. Figure 6.9 shows *17%* improvement on average in total reward of our approach as compared to Deterministic approach. In addition, our system has no shutdown time while Deterministic approach has *25-170* minutes shutdown time each day. Deterministic approach relies on prediction which has inaccuracy. Their plan is either optimistic at times, assigning high QoS level while harvesting is less than prediction or pessimistic at other times, assigning low QoS level while harvesting or energy storage is abundant.

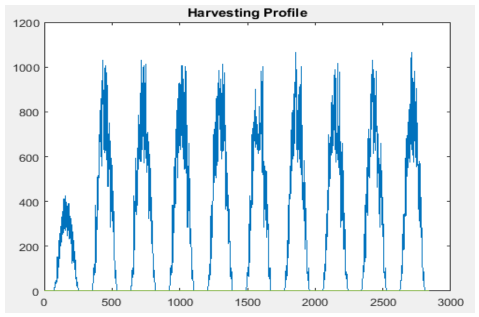
In the second set of experiments, I assume the system has two application states. The two states are event monitoring (state 1) and event processing (state 2). The Markov Chain model for the application process is shown in Figure 6. The system has *7* QoS levels as in the first experiment but in event processing state, application executes more tasks and consumes *1.5* times the load current in state 1. Since the Deterministic approach is not aware of different stochastic states of the application process, I define two variations, Deterministic 1 and Deterministic 2. Deterministic 1 optimistically assumes the application is in state 1 most of the time and therefore its plan is aggressive. Deterministic 2 assumes the system can be in state 2 at any time for safety and hence, the planning is less aggressive.

Furthermore, the Deterministic approach is coupled with an online adaptation algorithm (QUARES[10]) to improve the adaptation ability of Deterministic approach at runtime when there is a gap between prediction and actual energy harvesting. The online adaptation algorithm keeps track of energy harvesting and energy storage status to adjust application QoS accordingly. Note that this online adaptation is aware of the actual application states at runtime.

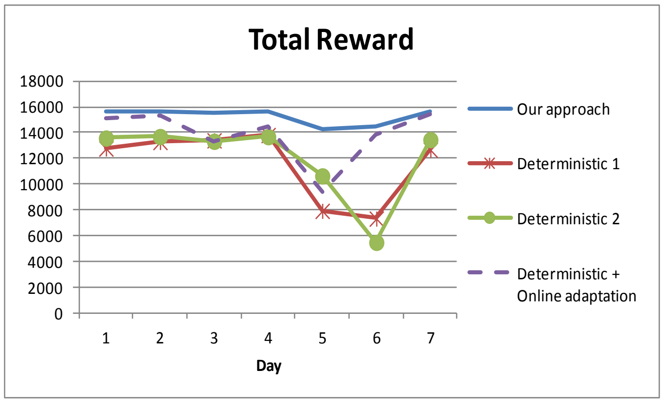
Figure 6.10 shows the result running our approach and two Deterministic approach variations for 7 days according to the harvesting profiles on the left side of Figure 6.10 (the first 3 days are for prediction for Deterministic Approach). The first harvesting profile is a series of normal days while the second harvesting profile contains one rainy day with significant less energy harvesting potential. Our approach has 28% improvement in total rewards on average compared to Deterministic 1, and 27% compared to Deterministic 2. Because Deterministic 2 assumes a less aggressive planning, we would expect it to have lower total reward. Counter to intuition, Deterministic 2 has higher total rewards, thanks to its less aggressive planning which reduces the effect of prediction inaccuracy, leading to less shutdown time. The system under Deterministic 2 therefore provides QoS for a longer time than Deterministic 1 and attains higher total rewards.



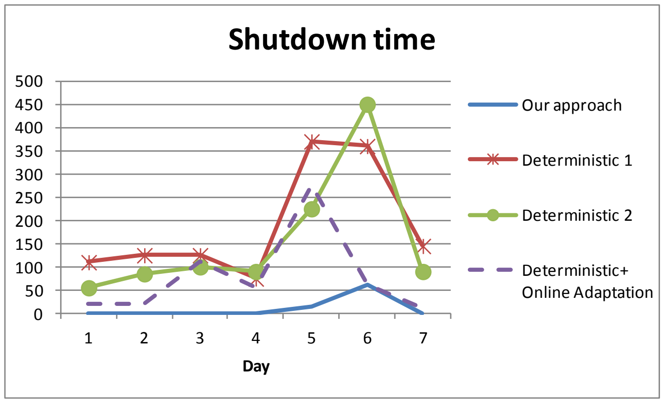
a) Harvesting Profile 1



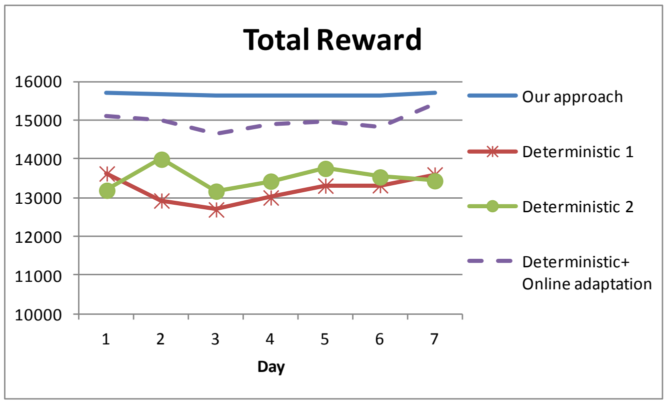
b) Harvesting Profile 2



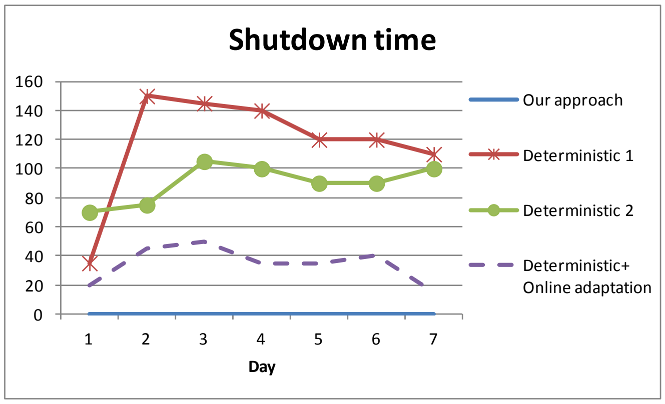
c) Total Reward Comparison 1



d) Shutdown Time Comparison 1



e) Total Reward Comparison 2



f) Shutdown Time Comparison 2

Figure 6.10 Comparison with Deterministic Approach for a Week, Multiple Application

The Deterministic + Online adaptation approach improves both total reward and shutdown time as compared to Deterministic 1 and 2. The online adaptation effectively adjusts QoS when there is a gap between prediction and actual harvesting, helping reducing shutdown time and as a result, improving total rewards. However, this improved deterministic approach is still unable to tackle large fluctuations, such as on rainy days (e.g., on day *5*, harvesting profile *1* during which it shuts down for *4* hours). Compared to Deterministic + Online adaptation, our stochastic approach has on average *4-12%* higher total rewards and zero to minimal shutdown time. This gap is expected to be larger with more variations in the solar harvesting profiles and the application demand.

Overall, our approach outperforms Deterministic approaches in any setting and in both defined metrics, higher total reward and lower or no shutdown time. Under rainy condition, it is unavoidable that supercapacitor runs out of energy, our approach shuts down for *15* minutes at the end of the rainy day and another *60* minutes at the beginning of the next day before the supercapacitor recovers above the minimum operating voltage.

6.7 Conclusion

In this work, I propose a unified stochastic model based on Finite State Markov Chain that captures both energy supply and energy demand variations and the complexity of harvesting system components. This unified model enables a complete cyber physical model and closed loop control for solar-powered supercapacitor-based systems. The proposed optimization framework aims to maximize the expected performance of the systems. Compared to state-of-the-art deterministic energy management framework, our proposed approach outperforms in term of QoS and energy sustainability (with less shutdown time).

References

[1] Mohammad Ghasemazar and Massoud Pedram, “Variation Aware Dynamic Power Management for Chip Multiprocessor Architectures”, in DATE 2011

[2] Hwisung Jung, and Massoud Pedram, “Uncertainty-Aware Dynamic Power Management in Partially Observable Domains”, in IEEE Transaction on VLSI Systems, 2009

[3] Chin Keong Ho, Pham Dang Khoa, and Pang Chin Mang, “Markovian Models for Harvested Energy in Wireless Communications”, in IEEE International Conference on Communication Systems 2010

[4] Longbo Huang and Michael J. Neely, “Utility Optimal Scheduling in Energy Harvesting Networks”, in Mobihoc 2011

[5] Alireza Seyedi and Biplab Sikdar, “Modeling and Analysic of Energy Harvesting Nodes in Wireless Sensor Networks”, in Annual Allerton Conference on Communication, Control, and Computing, 2008

[6] Joan Ventura and Kaushi Chowdhury, “Markov Modeling of Energy Harvesting Body Sensor Networks”, in IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 2011

[7] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. "Adaptive duty cycling for energy harvesting systems." In ISLPED 2006

[8] J. Recas Piorno, Carlo Bergonzini, David Atienza, and T. Simunic Rosing. "Prediction and management in energy harvested wireless sensor nodes." In Wireless VITAE 2009.

[9] Jun Lu, Shaobo Liu, Qing Wu and Qinru Qiu, “Accurate Modeling and Prediction of Energy Availability in Energy Harvesting Real-Time Embedded Systems”, in IGCC 2010

[10] Nga Dang, Elaheh Bozorgzadeh, and Nalini Venkatasubramanian. "QuARES: Quality-aware data collection in energy harvesting sensor networks." In IGCC 2011.

[11] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. "Power management in energy harvesting embedded systems with discrete service levels." In ISLPED 2009.

[12] Shaobo Liu, Qinru Qiu, and Qing Wu. "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting." In DATE'08, pp. 236-241. IEEE, 2008.

[13] Y. Choi, N. Chang and T. Kim, “DC-DC Converter-Aware Power Management for Low-Power Embedded Systems”, in TCAD, 2007

[14] Azalia Mirhoseini, and Farinaz Koushanfar. "HypoEnergy. hybrid supercapacitor-battery power-supply optimization for energy efficiency." In DATE 2011.

[15] Zheng Liu, Xinyu Yang, Shusen Yang, and Julie McCann, “Efficiency-Aware: Maximizing Energy Utilization for Sensor Nodes Using Photovoltaic-Supercapacitor Energy Systems” in International Journal of Distributed Sensor Networks, 2013

[16] Sehwan Kim and Pai Chou, “Techniques for Maximizing Efficiency of Solar Energy Harvesting Systems” in ICMU 2010

[17] Jaein Jeong, “A Practical theory of micro-solar power sensor networks”, Ph.D. thesis

[18] D. P. Bertsekas. “Dynamic Programming and Optimal Control”, Vols I and II. Boston: Athena Scientific, 2005 and 2007

[19] Navin Sharmaa, Jeremy Gummesonb, David Irwinb, Ting Zhuc , Prashant Shenoya , “Leveraging Weather Forecasts in Renewable Energy Systems” in SECON 2010

Chapter 7 Conclusion and Future Work

Energy harvesting is a very promising energy supply alternative for embedded systems. Compared to other alternatives such as plug-ins and batteries, energy harvesting has no generation cost (energy is free from the surrounding environment) and low maintenance cost. Most importantly, the energy sources are clean and replenish-able, making the systems self-sustainable. However, energy harvesting brings new challenges for embedded system energy management. These challenges were not present in the previous systems and therefore, novel energy management techniques must be proposed and adopted for energy harvesting embedded systems. This dissertation focuses on solar energy for its high availability and ease of accessibility.

The main challenges for solar-powered embedded systems are spatial and temporal variations in the solar profiles. Other challenges induced by energy harvesting are the non-ideal behaviors of complex hardware components such as converters and the aging effect of batteries. Furthermore, the variable demand of applications makes it even more challenging for any energy management technique in solar-powered embedded systems.

7.1 Contribution

This dissertation tackles the challenges of solar-powered embedded systems, its optimization goal is to maximize system performance measured by the application Quality of Services. The contribution of this thesis is proposing SQUARES middleware framework for energy management in such systems. In chapter 3 and 4, I explored different knobs of the systems for cross-layer energy management which allows system energy consumption tuning and matching with variable energy harvesting. In particular, I exploited error margin for data collection applications on wireless sensor networks in chapter 3 and a combination of DVFS and real-time task constraints in chapter 4.

In this thesis, I also highlighted the impact of harvesting activities on the energy storage subsystems. Specifically, the challenges are battery aging and supercapacitor leakage under fluctuating charging and discharging activities. In chapter 5, a middleware framework for hybrid energy storage architecture (of batteries and supercapacitors) is proposed to keep battery aging under threshold and to minimize supercapacitor leakage.

Finally, this thesis provides a novel stochastic model to capture solar profile variations and application demand variations. The model is built based on a unified Finite State Markov Chain in which each state captures the harvesting status, application status, and energy storage status. The transition edges among states capture not only the stochastic nature of energy harvesting sources and physical events but also the complex behaviors of harvesting circuit components. This unified Finite State Markov Chain representation of solar-powered embedded systems enables stochastic optimization based on Markov Decision process. The experimental results show improvements in system sustainability and performance compared to deterministic approaches such as prediction-based planning.

In conclusion, this work provides a cross-layer energy management middleware framework, SQUARES for solar-powered embedded systems. It tackles challenges of energy harvesting systems by two approaches, deterministic approach and stochastic approach. It exploits different knobs at the hardware, software layers and the application layer for energy tuning and matching. It reinforces the impact of energy harvesting systems by taking into consideration the energy storage lifetime and efficiency constraints, making the energy harvesting systems truly sustainable.

7.2 Limitation and Feasibility of Accomplished Work

In this section, I look at limitation of my accomplished work from two angles: the applications it can make an impact on and the scalability of the algorithms proposed. I analyze each of this perspective in section 7.2.1 and 7.2.2 respectively. Lastly, I look at the feasibility of implementing this framework on a real test bed and give my recommendation to make it work.

7.2.1 Application Limitation

Data collection applications: I chose approximated data collection applications on Wireless Sensor Network as the focus of my study for energy harvesting communication-intensive systems. However, approximated data collection protocols exploit only error margin (i.e., data accuracy) adaptation for energy consumption tuning. There are other important data qualities such as data timeliness should also be considered or other models to capture data accuracy. Furthermore, in my work, I assumed the relationship between an error margin and the corresponding energy consumption is stationary and it is possible to profile and obtain this information prior to the offline planning phase. In fact, this relation can change dynamically at run-time. In that case, the given offline phase may not work. A model that can capture the dynamic changes in data characteristics (e.g., stochastic model) is needed for the offline phase to work or a complete online algorithm is required.

Smart Camera Networks: Differently from data sensing and collection protocols, many networking applications such as object tracking, object identification require in-network processing in addition to sensing and communicating. Understanding the semantics of these applications, their Quality of Service definitions, demands, and requirements, their patterns of communicating may give advantages to energy management in scheduling tasks, tuning energy consumption, and optimizing application performance.

Real-time applications: I focused on soft real-time systems with (m,k)-firm constraints for my study of computation-intensive systems. Soft real-time systems are just one class of real-time systems, and real-time systems again are just one category of computation-intensive systems. There are many other possible systems to look at. (m,k)-firm constraints assume any task can be dropped as long as the system can meet *m* out of every *k* sequential deadlines. This may ignore the semantics of the applications which may value one task than other tasks or there are dependencies among tasks that make it impossible to drop an important one.

To sum up, I looked at two specific systems: communication-intensive systems and computation-intensive systems. However, there are systems with fairly equal communication and computation tasks. Such systems would require an orchestration of tuning knobs for communication and tuning knobs for computation. An integrated computation and communication model is needed for effective tuning.

7.2.2. Algorithmic Limitation

In chapter 3 and 5, I used Integrated Linear Programming (ILP) which is a NP-complete problem. Although there are effective solvers for ILP, its running time still increased exponentially as the number of variables and constraints increase. There is no bound for running time of NP-complete programs. Thus, if number of variables and constraints are large and there is limited time for running the ILP, heuristic approaches that exploit the structure of the problem is needed.

In chapter 4, I proposed a dynamic programming (DP) solution which is typically a polynomial algorithm. Dynamic programming works if possible choices at each step are known and results at each step are dependable only on previous steps. In addition, system states and actions at each step must be discrete. If system states and actions at each step are continuous, quantization is required and hence there are approximation errors. However approximation errors can be bounded using iterative approximation methods.

Chapter 6 exploits a stochastic approach based on Finite State Markov Chain model. Finite State Markov Chain model is a very flexible and effective model to capture the stochastic nature of physical environment and complexity of harvesting circuits which may pose difficulties for ILP and DP solutions otherwise. However, it relies on the prior knowledge of energy harvesting sources and physical events to build Markov Chain states and transitions. It is not straight forward to build accurate models and to detect accurate state at run-time while inaccurate models will reduce the optimality of the solution. In addition, the number of possible states in the systems and actions affects the running time and size of the optimal policy look-up table. If the size of the optimal policy look-up table is large, it may take up a significant portion of the memory on a limited-memory embedded system. In case the memory is insufficient to store the whole optimal policy table, it could be fragmented and the right fragment is retrieved and stored when needed. Another interesting point is that theoretically, it is straight-forward to extend this stochastic approach for a network of solar-powered embedded systems. However, the number of states, the running time, and the size of the optimal policy table will grow exponentially with number of nodes in the networks, making it infeasible in practice. Smart approaches to tackle this scalability problem are desirable to bring the benefit of stochastic models and optimization methods from the individual node level to the network level.

7.2.3 Feasibility of Implementation

SQUARES middleware framework certainly brings many benefits for solar-powered embedded systems it supports. However, its successful implementation depends on several factors listed below. One thing is the existence of a sensing system that allows each embedded system to read its harvesting power (voltage and current), the energy storage status (charging and discharging current, battery and/or supercapacitor voltages, their State of Charge), and load demand (voltage and current). The second thing is the ability to control the hardware and to tune the systems knobs (such as DVFS) and software knobs (such as OS scheduler). Importantly, it must be able to communicate with and adapt the application (such as tuning data error margin or changing task constraints). These requirements necessitate changes to the application code.

The overhead of the middleware must also be considered for realization of the benefits of this SQUARES middleware on solar-powered embedded systems. Measurement of timing and energy consumption of this middleware's algorithms is needed to decide the frequency and depth of adaptation and controlling.

7.3 Future work

In this section, I discuss about directions for future work. The limitations on applications and algorithms listed in section 7.2 are good candidates for extension of this work. In addition, I listed here other possibilities for extension.

It is necessary to model solar-powered embedded systems or energy harvesting systems in general as complete cyber physical systems to enable close-loop control. Stochastic model is promising, but the challenges in building accurate models must be tackled first. Furthermore, the extension to network level can benefit many new applications whose Quality of Services are based on network-level metric. Yet this poses a scalability issue (mentioned in section 7.2.2) which is an interesting research problem to investigate.

Furthermore, there are emerging technologies in both energy harvesting and energy storage such as micro-engineer batteries which promise to bring the benefits of both batteries and supercapacitors together. These new technologies offer new benefits to explore and yet new challenges to tackle.

This thesis focuses on solar energy. However, there are other renewable sources such as wind, kinetic, thermal energy that can be harvested. Each of this has different characteristics and their applications and contexts are vastly different (e.g., body area networks exploiting kinetic energy while system on chips harvesting thermal energy). It is promising to extend this middleware framework to other renewable energy sources, systems, applications, and environments.