Energy saving and scavenging in stand-alone and large scale distributed systems

by

Xuejing He

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in the University of Michigan
2015

Doctoral Committee:

Associate Professor Robert P. Dick, Chair
Associate Professor Jason Mars
Professor Trever N. Mudge
Professor Dawn M. Tilbury

ACKNOWLEDGMENTS

I would like to thank Prof. Robert Dick for his help and guidance through all my graduate school years. His broad knowledge across different fields and insightful thoughts has provided me the opportunities to explore different projects. He also teaches me scientific methods of doing research, which helps me make achievements in my thesis.

I give my thanks to my collaborators. Professor Russ Joseph has provided help and guidance for the power deregulation and ambient energy aware routing projects. His innovative thoughts and experience in system design greatly inspire my design and experiments. I would also like to thank Professor Sharon Hu, Professor Tam Chantem, and Doctor Xiaobo Fan, who provide useful insights and suggestions on my projects. Thanks also go to Troy Renken, Tim Powers, and John Vong with ZPower for providing silver-zinc battery samples and helping with their characterization in Chapter 2. I want to express my thanks to Professor Li Shang, who guided me through my internship and continued following up with my research projects. Thanks also go to my group mate Xi Chen, who helped me start the HAMS project (Chapter 4) and shared several useful design techniques. I would also like to thank Srinath Arunachalam, Yue Ma, and Yudong Gao for their collaborations. Thanks also go to Professor Donald Winsor, who gained me access to EECS servers for my crash test in Chapter 5.

I would give my thanks to my other committee members, Professor Jason Mars, Professor Trever Mudge, and Professor Dawn Tilbury, for their time and helpful suggestions to my thesis work.

Thanks also go to my research groupmates: Lan Bai, Xi Chen, David Bild, Lide Zhang, Yun Xiang, Tao Zhao, Yue Liu, and Andrew DeZeeuw. Their inputs in group meetings and research discussions greatly inspire my ideas in research. Their friendship also supports me through many difficulties in graduate school.

I also want to thank all my friends at University of Michigan. Their company makes my graduate school years full of fun and memory. Special thanks go to my former roommates Sarah Nowaczyk and Lois Smith. Their friendship and generous holiday invitations make my life as an international student much easier and happier.

Finally, I want to thank my parents. Even during my most difficult times, they still have faith in me. Without their support and encouragement, I would not have finished this dissertation.

TABLE OF CONTENTS

Acknowledgments . . . . . **ii**

List of Figures . . . . . **vi**

List of Tables . . . . . **ix**

Abstract . . . . . **x**

Chapter

1 Introduction **1**

1.1 Solutions 4
1.1.1 Hardware and Software Codesign of Deregulated Energy Delivery Systems 4
1.1.2 Routing Protocol Design for Sensor Networks Powered by Opportunistic Energy Sources 5
1.1.3 Energy and Performance Optimization Considering Resource Sharing in Heterogeneous Data Centers 5
1.1.4 Minimizing Data Center Cooling Energy Under Reliability Constraints 6
1.2 Thesis Overview 7

2 Embedded System and Application Aware Design of Deregulated Energy Delivery Systems **9**

2.1 Introduction 9
2.2 Related Work and Background 12
2.3 Power Deregulation 13
2.4 Lifespan for Battery-Powered Systems 14
2.5 Model 15
2.5.1 Power Consumption and Performance Model 15
2.5.2 Battery Model 16
2.5.3 System Level Model 19
2.6 Characterization and Analysis of a New Silver-Zinc Battery Technology . 20
2.6.1 Charging and Discharging Characteristics 20
2.6.2 Internal Resistance 21
2.6.3 Impedance Test 22
2.6.4 Summary of Silver-Zinc Battery Evaluation 23
2.7 Factors Affecting System Lifetime 23

2.7.1 Internal Resistance . . . 24
2.7.2 Lowest DVFS Voltage . . . 25
2.7.3 Impact of Core Count . . . 26
2.7.4 Processor Equivalent Resistance . . . 27
2.8 Deregulated System Design Procedure . . . 27
2.8.1 Battery Lifespan Comparisons . . . 30
2.9 Discussion, Caveats, and Future Directions . . . 32
2.9.1 Linear Regulators . . . 32
2.9.2 Analog Components . . . 33
2.9.3 Future Battery Technologies . . . 34
2.10 Conclusions and Acknowledgments . . . 34

3 Spatially- and Temporally-Adaptive Communication Protocols for Zero-Maintenance Sensor Networks Relying on Opportunistic Energy Scavenging . . . 35

3.1 Introduction . . . 35
3.2 Related Work And Motivation . . . 38
3.2.1 Engery Scavenging with Battery Assistance . . . 38
3.2.2 Why Battery-Less Energy Scavenging? . . . 39
3.2.3 Node Design and Protocol Support . . . 40
3.3 Problem Definition . . . 41
3.4 Design and Implementation . . . 43
3.4.1 Sensor Node Architecture . . . 44
3.4.2 Precomputed Information . . . 46
3.4.3 Routing Protocols . . . 47
3.5 Evaluation . . . 50
3.5.1 Experiment Setup . . . 50
3.5.2 Application Based Evaluation . . . 53
3.5.3 Delivery Rate for Varying Parameters . . . 55
3.6 Protocol Selection . . . 59
3.6.1 Channel Utilization . . . 59
3.6.2 Per-Node Packet Delivery Rate . . . 61
3.6.3 Protocol Selection Strategy . . . 62
3.7 Discussion and Caveats . . . 63
3.7.1 Long-term Wind Speed Variation . . . 64
3.7.2 Online Adjustment Of Node Activity Rate . . . 64
3.7.3 Protocol Extensions . . . 65
3.8 Conclusion . . . 65

4 A Scheduler For Performance and Energy Optimization in Data Centers with Heterogeneous Tasks or Machines . . . 66

4.1 Introduction . . . 66
4.1.1 Contributions . . . 69
4.2 Problem Definition . . . 70
4.2.1 Performance Metrics . . . 70
4.2.2 Energy Optimization . . . 72

4.3 Design and Implementation . . . 73
4.3.1 Task Pre-Characterization and Performance Modeling . . . 74
4.3.2 Throughput Prediction . . . 77
4.3.3 Model Accuracy . . . 79
4.3.4 Impact on Data Locality . . . 81
4.3.5 Other Design Considerations . . . 82
4.4 Power Model . . . 83
4.5 Evaluation of the Task Scheduler . . . 85
4.5.1 Experimental Setup . . . 86
4.5.2 Benchmarks . . . 87
4.5.3 Performance Evaluation For All Benchmarks . . . 89
4.5.4 Balancing Machine Resource Utilization . . . 89
4.5.5 Energy Consumption Reduction . . . 92
4.6 Additional Feature Considering Data Locality . . . 94
4.6.1 Models . . . 96
4.6.2 Scheduler Design . . . 99
4.6.3 Experimental Results . . . 100
4.7 Related Work . . . 102
4.8 Conclusion And Future Work . . . 103

5 Reliability-Aware Cooling Energy Saving Through Task Assignment in Heterogeneous Data Centers . . . 105

5.1 Introduction . . . 105
5.2 Related Work . . . 110
5.3 Problem Definition . . . 111
5.3.1 System Overview . . . 113
5.4 Modeling . . . 114
5.4.1 Energy and Thermal Modeling . . . 115
5.4.2 Reliability Constraints . . . 117
5.4.3 Wear State Measurement . . . 119
5.5 Methodology and Design . . . 121
5.5.1 Core Wear-State Test . . . 121
5.5.2 Task Scheduler Design . . . 123
5.6 Experiments . . . 126
5.6.1 Wear State Measurement . . . 126
5.6.2 Circuit Critical Path Model . . . 127
5.6.3 Task Scheduler Deployment . . . 129
5.6.4 Result Discussions . . . 134
5.7 Conclusion . . . 136

6 Conclusion and Future Works . . . 138

Bibliography . . . 140

LIST OF FIGURES

1.1 Relationship of research directions of this thesis 3

2.1 PCB of an iPod Nano [7]. Voltage regulation circuitry and components take up to 25% of the PCB area. 10
2.2 Discharge curves and internal resistances of different types of batteries with capacities of 100 mAh. Data for the silver-zinc battery comes from measurements. Data of other battery types comes from the *Handbook of Batteries* [8]. . 10
2.3 The operating timeline of power deregulation. 14
2.4 Battery model used for a regulated system. 18
2.5 Discharge curve for a 100 mAh silver-zinc cell. 21
2.6 Internal resistance test. 21
2.7 Battery lifespans for power deregulated and regulated systems with varying battery internal resistance values. 25
2.8 Battery lifetime changes with processor load resistance. 26
2.9 Design and battery-processor selection for power deregulated and regulated systems. The numbers correspond to the steps described in Section 2.8. 29
2.10 Lifespan comparison for sensor network apps. 31
2.11 Lifespan comparison for multi-media apps. 32

3.1 Battery-less energy scavenging sensor network operation. A node S transmits its packets to base station B through temporally intersecting active subsets. . . 43
3.2 Packet delivery rate comparison for four protocols under different applications. 52
3.3 Transmission latency comparison of four protocols when network scale changes. Arrows indicate that latency exceeded application latency constraint. 57
3.4 Transmission latency comparison of four protocols when sample size changes. Arrows indicate that latency exceeded application latency constraint. 57
3.5 Transmission latency comparison of four protocols when transmission range changes. Arrows indicate that latency exceeded application latency constraint. 58
3.6 Transmission latency comparison of four protocols when threshold wind speed changes. Arrows indicate that latency exceeded application latency constraint. 59
3.7 The node-level activity rate given by the statistical data. 60
3.8 The node channel utilization for Ambient Energy Aware routing. 61
3.9 The node level packet delivery rates using four different routing protocols. . . . 62

4.1 Dependence of latency on the maximum concurrent tasks constraint. The latency numbers are measured on one machine running two example workloads with different task compositions. The optimal latency value depends on the particular workload. 67
4.2 CPU and IO loading on a cluster of six machines running a group of workload. The workload consists of 40 CPU intensive tasks and 40 IO intensive tasks. The cluster consists of two types of computers: machine 1–3 and machine 4–6. The IO loadings for machine 4–6 are approximately zero. 68
4.3 System architecture of HAMS. 74
4.4 Performance model for one Map task. Task latency is dependent on resource utilizations. 76
4.5 Task throughput prediction procedure. 80
4.6 Workload execution times for HAMS and existing Hadoop schedulers on two clusters. 90
4.7 Distribution of resource utilization vectors when running one benchmark using *ori* and HAMS. We omitted the results for the other schedulers because *simple* is similar to *ori* and *hetero* is similar to HAMS for this example. 90
4.8 The number of concurrently running tasks on the cluster. The labels mark the job finish times for different schedulers. 91
4.9 Execution time comparison of schedulers for lightly-loaded benchmarks on a 30-node cluster. 92
4.10 Energy savings with task concentration for *ori* and HAMS. *init* does not use task concentration. 93
4.11 Energy consumption–latency relationship for Benchmark 6. 94
4.12 Experiment setup for the motivating example. The input data of all tasks reside in node 1. We test the assignment of these 20 tasks starting from running all tasks on node 1 (all local) to running all tasks on node 2 (all remote), increase the number of remote task by 1 each time. 95
4.13 Change of workload durations as the number of remote tasks changes. 95
4.14 Experiment setup for building the network transfer model. 97
4.15 Network transfer model for one task. 97
4.16 Overview of the task scheduler. 100

5.1 Relationship between data center energy consumption, temperature, and processor lifetime. 107
5.2 This figure shows a voltage–temperature dependency curve of a processor. This can be retrieved through circuit modeling of the critical path of that processor. Using this curve, one can determine the crashing frequency f_1 under normal operating voltage and temperature (T_1, V_1) by tracing back from the extreme operating condition (T_2, V_2, f_2) at which there is a processor core crash due to timing error. 109

5.3 Thermal transfer network model for an air-cooled data center. Each core is modeled as a power source P_{core_i}. Server fans and the chiller are modeled using their thermal resistances R_{fan_i} and R_{air}. T_{c_i}, T_{hs_i}, T_{room} and T_{air} are the core temperature and heat sink temperature of core i, server room temperature, and outside air temperature. 112
5.4 System diagram of wear state test and reliability aware task scheduler. 114
5.5 Using measured wear states and the circuit critical path model to determine the processor temperature setpoint. 121
5.6 The flow chart of the task assignment algorithm. It consists of three main parts: 1. Task utilization and response time prediction ; 2. Per-core temperature, power consumption, reliability, and task response time calculation; and 3. Cooling energy calculation. 124
5.7 The temperature-delay relationship from the critical path model simulation. . . 129
5.8 Total energy consumptions for three algorithms. 130
5.9 CPU utilization of different benchmarks. 131
5.10 The temperature setpoints of the three algorithms. The dots are temperature setpoints of different cores in the Varying Temperature method. The "minimum" line is the temperature setpoint of the Universal Temperature method, and the "fixed" line is the temperature setpoint of the Fixed Temperature method. 132
5.11 Energy consumption comparison of three algorithms when the fixed operating temperature changes. This affects the energy consumption of the Fixed temperature method, but has little influence on the other two. 133
5.12 Total energy consumption of the cluster when the mean of core MTTF changes. 134
5.13 Total energy consumption of the cluster when the variance of core MTTF changes. 135

LIST OF TABLES

2.1 HTC Smart Phone Power Consumption Breakdown 32

3.1 Applications And Their Parameters 54

4.1 Task Pre-Characterization Overhead 78
4.2 Server Power Consumption Breakdown 85
4.3 Description of Tasks and Benchmarks 88
4.4 Average Data Locality And Job Finish Time 101

5.1 Voltage = 1.47 V, frequency = 3.16 GHz 127
5.2 Voltage = 1.41 V, frequency = 2.67 GHz 128

ABSTRACT

This thesis focuses on energy management techniques for distributed systems such as handheld mobile devices, sensor nodes, and data center servers. One of the major design problems in multiple application domains is the mismatch between workloads and resources. Sub-optimal assignment of workloads to resources can cause underloaded or overloaded resources, resulting in performance degradation or energy waste. For example, improperly distributed workloads in battery-powered embedded systems result in inefficient utilization of batteries, which shortens battery lifetimes in systems without voltage regulators. In batteryless sensor nodes, communication frequently becomes unreliable under these conditions. Further, without a method to predict resource utilization times, workload mismatch in data centers causes resource contention or idle resources. Finally, significant energy can be wasted not only on data center servers, but also on data center cooling.

This work specifically focuses on the heterogeneity in system hardware components and workloads. It includes energy management solutions for unregulated or batteryless embedded systems; and data center servers with heterogeneous workloads, machines, and processor wear states. This thesis describes four major contributions: (1) This thesis describes a battery test and energy delivery system design process to maintain battery life in embedded systems without voltage regulators. (2) In battery-less sensor nodes, this thesis demonstrates a routing protocol to maintain reliable transmission through the sensor network. (3) This thesis has characterized typical workloads and developed two models to capture the heterogeneity of data center tasks and machines: a task performance model and a machine resource utilization model. These models allow users to predict task finish time on individual machines. It then integrates these two models into a task scheduler based on the Hadoop framework for MapReduce tasks, and uses this scheduler for server energy

minimization using task concentration. (4) In addition to saving server energy consumption, this thesis describes a method of reducing data center cooling energy by maintaining optimal server processor temperature setpoints through a task assignment algorithm. This algorithm considers the reliability impact of processor wear states. It records processor wear states through automatic timing slack tests on a cluster of machines with varying core temperatures, voltages, and frequencies. These optimal temperature setpoints are used in a task scheduling algorithm that saves both server and cooling energy.

CHAPTER 1

Introduction

Both small and large scale distributed systems are widely used across different areas. Small distributed systems are embedded systems used in various applications, e.g., personal hand-held devices and wireless sensor networks. These applications have great influence on people's everyday life. For example, there are 2 billion smartphone users worldwide; this number will continue increase in the near future. Large distributed systems includes data centers, which consist of thousands of server computers that provide high computational power for both computational and data intensive applications. These applications support both research grade and commercial facilities such as universities, research labs, and companies. For example, Facebook uses over 20,000 servers in one of its data centers to support the traffic of its website.

Energy efficiency and energy saving are important to such distributed systems. Embedded systems have tight constraints on space and weight, which constrains the size and capacity of their energy sources. These constraints cause systems to sacrifice throughput or lifetime if designed improperly. Data centers have both high computational capacity and energy consumption. They require 13 billion dollars of electricity every year in the United States, 50% of which goes to data center cooling systems [1]. Saving energy for such systems while maintaining system performance can achieve significant economic benefits.

One of the major design challenges in energy saving for embedded systems and data centers is the mismatch between workloads and resources. Sub-optimal assignment of

workloads to resources can cause underloaded or overloaded resources, resulting in performance degradation or energy waste. Improperly distributed workloads in battery-powered embedded systems result in inefficient utilization of batteries, which shortens system lifetimes. Workload mismatch in data centers causes resource contention or idle resources. Resource contention significantly slows down workload execution, decreasing data center performance, while idle resources waste up to 40% of the total server energy [2].

This problem is particularly severe in a heterogeneous environment. Failing to consider workload and resource heterogeneity can aggravate the mismatch between workloads and resources, leading to performance loss or energy waste. This thesis considers the matching between workloads and two types of heterogeneous resources: the energy sources and computational resources.

First, we consider heterogeneity in energy sources in embedded systems. Non-ideal energy sources, such as batteries and scavenged energy sources, have spatial and temporal variations. This heterogeneity in non-ideal energy sources greatly impacts performance and lifespan of embedded systems. Failing to consider this heterogeneity causes problems. Battery voltage decreases and internal resistance increases as a battery is drained. These changes reduce system throughput and/or decrease system lifetime if workloads do not adapt to these changes. Wireless sensor networks operating in the wild can completely eliminate their batteries and operate only using scavenged energy from the environment. These scavenged energy sources, e.g., wind power, solar power, and water flow power, however, are available at random times and locations. If workloads are assigned at the time/location at which the scavenged energy sources are insufficient, the sensor network suffers from long transmission delay or transmission failure.

Second, heterogeneity in computational and cooling resources in data centers also makes workload assignments a challenge. In data centers, energy consumption is wasted when or where computation is low. On the one hand, a large portion of the computational energy on server machines is spent keeping the machines active even when the machines are idle or

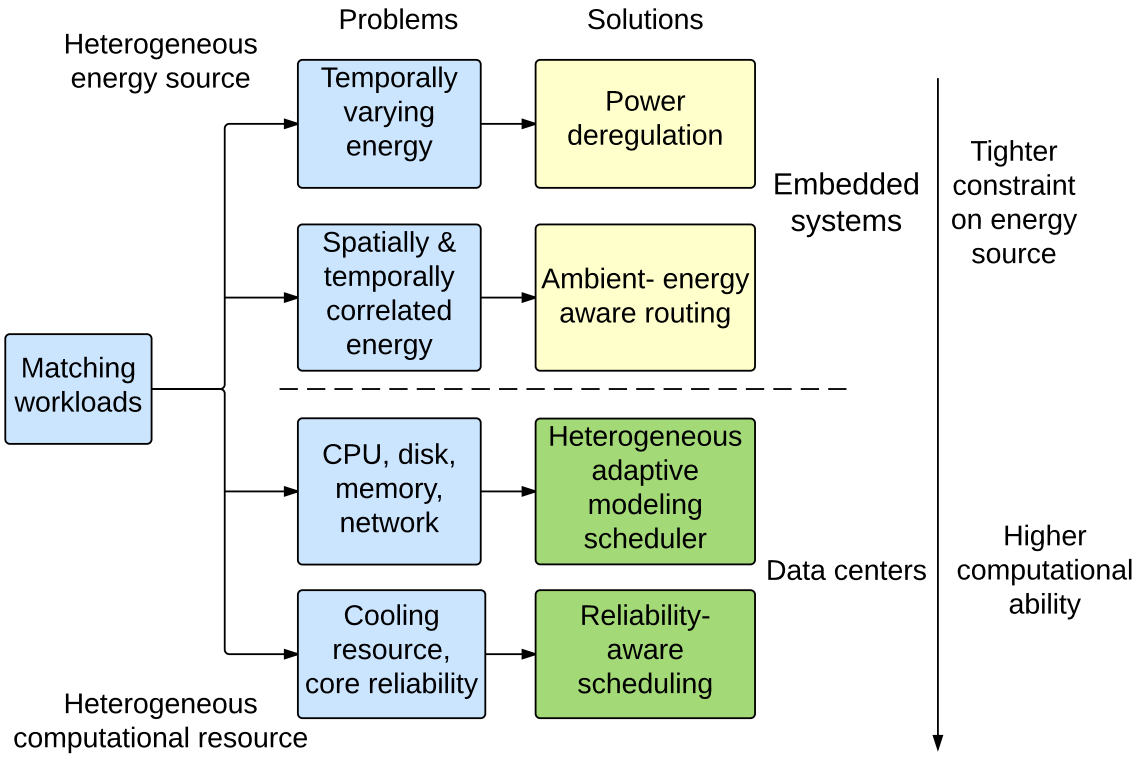


Figure 1.1: Relationship of research directions of this thesis

running light workloads. Existing heterogeneity-blind assignments aggravate this problem. These assignments can easily overload or underload computational resources such as CPU, disk, and memory, creating contention in some resources while leaving others idle, wasting energy. On the other hand, additional cooling energy is wasted keeping processors running at unnecessarily low temperature setpoints. These sub-optimal temperature setpoints are caused by overlooking the difference in processor process variation and wear states.

As a result, finding the optimal workload assignment on heterogeneous resources is an important, yet unsolved problem. In this thesis, I verify this thesis statement:

Across a wide range of applications, workload assignment policies that consider the matching between spatially- and temporally- heterogeneous workloads and resources improve system lifetime, energy efficiency, and reliability, compared to existing heterogeneity-blind policies.

1.1 Solutions

I divide the above problem into two branches: matching workloads with heterogeneous energy sources, and matching workloads with heterogeneous computational and cooling resources. I break these problems into four subproblems and solve them using specific approaches. The organization is described in Figure 1.1. Below is a list of specific subproblems and my solutions.

1.1.1 Hardware and Software Codesign of Deregulated Energy Delivery Systems

Battery-powered embedded systems are constrained by energy source capacity, size, weight, etc. Researchers have reduced printed-circuit-board (PCB) areas by removing components from the system, e.g., removing voltage regulators. However, this can degrade system lifetime as it reduces the system lifetime.

I studied the impact of *power deregulation*, a technique that removes voltage regulators from embedded systems to save PCB area, upon embedded system design decisions. When they operate directly on raw batteries, processors suffer from single-thread performance degradation as their operating voltages decrease with the battery voltage. Power deregulation compensates for throughput by activating additional cores when battery voltage is below a certain threshold. This system can suffer from significant lifetime degradation if designers do not consider battery discharge characteristics and internal resistance changes.

I developed a battery-system codesign process that explores the matching between a battery and a power deregulated system. This process explores the relationships among battery discharge voltage, internal resistance, processor operating voltage, and processor equivalent resistance. I also provide a procedure for selecting the best-matching battery type for an embedded system.

1.1.2 Routing Protocol Design for Sensor Networks Powered by Opportunistic Energy Sources

Some sensor nodes operating in wild, outdoor environments operates without batteries completely to prevent the effort of removing dead batteries or to prevent environmental pollution from degraded batteries [3]. They rely on energy scavenging from the environment, e.g., solar power, wind power, and water flow power, etc. These energy sources are only available intermittently. Therefore, improperly designed routing protocols wake up sensor nodes when scavenged energy sources are not available, degrading or disabling data transmission.

I designed an Ambient Energy Aware (AEA) routing protocol for energy scavenging sensor networks that utilizes the spatial and temporal correlation among energy sources to maximize the probability of successful data transmission. It detects the energy source availabilities at the location and time of sensor node activation and uses a probability function to calculate the awake/sleep schedule of sensor nodes. Sensor nodes with available energy sources along the path from source nodes to the base station are selected for reliable data transmission.

1.1.3 Energy and Performance Optimization Considering Resource Sharing in Heterogeneous Data Centers

Data centers suffer from throughput degradation caused by resource contention. This is the result of task assignments that overlook machine and task resource utilization. Also, server machines in data centers consume a significant amount of electricity energy, up to 40% of which is idle energy consumption [2]. People proposed a task concentration technique that consolidates tasks on a subset of machines and shuts down the rest [4]. This reduces idle energy consumption, but can significantly increasing job execution time due to resource contention among consolidated tasks. Another source of data center performance

degradation is remote task execution, i.e., running tasks on machines that do not contain their input data. One common solution to this problems is to always assign tasks to machines containing the input data of the tasks. Unfortunately, this can result in poor resource sharing among tasks. The trade-off between data locality and resource contention needs to be considered.

I designed and implemented Heterogeneous Adaptive Modeling Scheduler (HAMS), with the purpose of improving performance and reducing energy consumption in data centers. It uses a resource utilization model and a task performance model to select task assignments that do not result in significant resource contention. The two models in this scheduler capture task and machine heterogeneity of a server using resource utilization vectors. This method allows task concentration without much increasing job execution time, while still reducing idle energy consumption.

I also evaluate a method that determines whether to assign a task to a machine containing its input data or to a remote machine. It considers overheads of both migrating tasks and data using a network data transfer model and resource utilization models. The data transfer model captures the delay of migrating a data set by considering disk utilization and the network transfer delay.

1.1.4 Minimizing Data Center Cooling Energy Under Reliability Constraints

HAMS only focuses on the computational energy consumptions of server machines. I extended my work to minimizing cooling energy by selecting the most appropriate operating temperature setpoints of processors. Lower processor temperature setpoints require larger amounts of cool air or lower chiller temperatures, increasing cooling power consumption. These low temperature setpoints are used to prevent reliability problems in the processors. However, determining the reliability constraint of processors is challenging because processor wear states are unknown. Therefore, processor temperature setpoints are set to much

lower values to leave sufficient margins, wasting cooling energy.

I describe a cooling energy saving algorithm that uses task assignment to allow processors to run at optimal temperature setpoints. The impact of candidate assignments on data center energy consumption is calculated using a data center thermal transfer model and a cooling energy model. Assignments resulting in minimal energy consumption are then selected. The resulting task scheduler achieves on average 18% reduction of total data center energy consumption.

The optimal temperature setpoint of each processor is given by its reliability constraint and wear state. While the reliability constraint is specified by users, the processor wear state is measured through an automatic test process. This test stresses the processor under different temperature, voltage, and frequency combinations, and records the operating conditions resulting in system crashes. The processor operating states of this crashes are used to calculate the processor timing slack, which is used to estimate processor wear states.

1.2 Thesis Overview

The organization of this thesis is described below.

Chapter 2 describes a design flow for *power deregulation*, a technique that removes voltage regulators from systems while still maintaining system throughput. We present a battery and system model that describes regulated and deregulated system behaviors during battery discharge, and analyzes the impact of several system parameters on battery lifespan. We also present the result of discharge, resistance, and impedance tests of a new battery technology suitable for power deregulation.

Chapter 3 describes a routing protocol for reliable communication in battery-less sensor networks. This design incorporates spatial- and temporal-correlation of scavenged energy sources. We also discuss the feasibility of this design across different sensor network applications.

Chapter 4 presents *HAMS*, a task scheduler that considers machine and task heterogeneity for MapReduce tasks in data centers. We also describe task models that describe task execution time under different background loadings. We then describe how this technique can be used to aid *task concentration* technique to save data center energy consumption by shutting down idle machines. Finally, we develop a network data transfer model, and integrate it with HAMS to account for remote task execution.

Chapter 5 describes reliability-aware cooling energy saving through task scheduling. We present a task assignment scheme that selects the optimal temperature setpoints of processors to minimize cooling energy, while not violating reliability constraints. We introduce a method of measuring the per-core reliability constraint through an automatic wear state measurement on server machines.

CHAPTER 2

Embedded System and Application Aware Design of Deregulated Energy Delivery Systems

2.1 Introduction

Bulky, inefficient power regulation circuitry imposes large overhead on embedded systems. Embedded systems often have size, energy consumption, and battery lifetime constraints. The (inappropriate) use of voltage regulators can cause an embedded system design to violate each of these constraints.

First, the voltage regulator can take up to 30% of printed-circuit board (PCB) area in embedded systems. Figure 2.1 demonstrates the PCB photo of one example embedded system.

Second, commercial switching voltage regulators suffer from energy loss during conversion. The resulting conversion efficiency of commonly used buck converters is 85%, while that of buck-boost converters is around 70% [5]. Furthermore, switching regulators have high power consumptions compared to the rest of the system in sleep mode. The quiescent current of voltage regulators in sleep mode ranges from tens of microamperes to several milliamperes, wasting energy [6].

Third, peripheral components in DC–DC converters also add inductance to the power delivery network, exposing the circuit to $L\mathrm{d}I/\mathrm{d}t$ effects, potentially causing reliability problems.

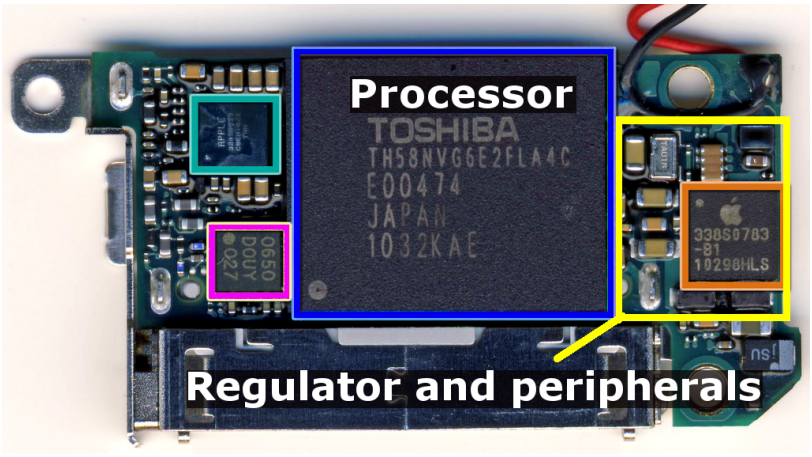
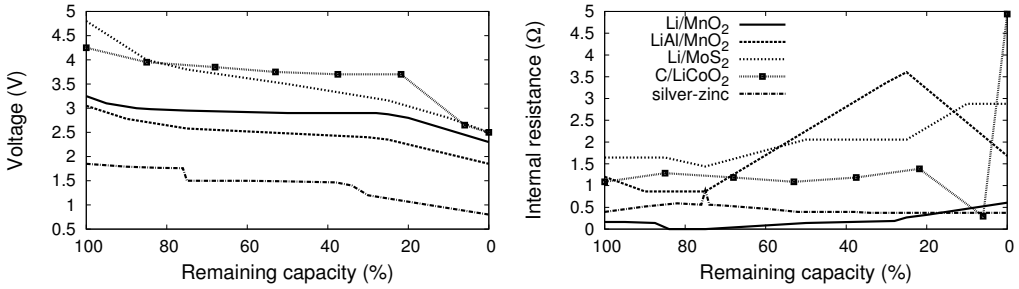


Figure 2.1: PCB of an iPod Nano [7]. Voltage regulation circuitry and components take up to 25% of the PCB area.



(a) Discharge curve.

(b) Internal resistance.

Figure 2.2: Discharge curves and internal resistances of different types of batteries with capacities of 100 mAh. Data for the silver-zinc battery comes from measurements. Data of other battery types comes from the *Handbook of Batteries* [8].

People have proposed system designs to remove voltage regulation circuitry. However, conventional systems without voltage regulators are usually single-core, low power embedded systems that have limited performance [9]. Such systems directly power processors using batteries with voltages that decrease over time (Figure 2.2a). To allow reliable operation, designers generally run processors at the frequency appropriate for the lowest battery voltage, leaving a large performance margin when the battery voltage is high.

In contrast, *power deregulation* [10] provides a method to maintain performance on multi-core systems without voltage regulators. It maintains throughput by activating additional processor cores as battery voltage decreases, taking advantage of thread-level parallelization.

Power deregulation is most appropriate for multi-core embedded systems, e.g., those running signal processing or multi-media applications, whose workloads have good parallelism and can thus benefit from performance compensation by power deregulation.

However, when designed improperly, the battery lifespans of power deregulated systems degrade. The influences on battery lifespan in power deregulated systems are complex. Battery properties, such as depth-of-discharge (DoD) dependant changes in voltage and internal resistance influence the performances and lifespans of deregulated systems. Previous work overlooks or underestimates these time-dependent variations. For example, prior work does not consider the impact of battery internal resistance [10, 11]. This has similar magnitude to the equivalent resistance of a processor (as shown in Figure 2.2b), causing significant internal voltage drop for batteries. In power deregulated systems, additional cores are activated to compensate for performance loss. Determining the battery DoD at which a new core needs to be activated requires, again, knowledge of both the transient behavior of the processor and the battery DoD.

We introduce an electrical battery model and a corresponding system model to describe the time-dependent changes in unregulated systems. We focus our models on power deregulation, as it out-performs other unregulated systems. The battery model captures time-dependent changes to battery characteristics. The embedded system model captures the relationship between battery state, processor voltage, and system power consumption. Together, these models enable rapid evaluation of different battery–processor combinations during the design process. We also analyze characteristics of batteries and processors that influence system lifetime, and give battery selection guidelines for designers of power deregulated systems.

Widely used rechargeable batteries, such as Lithium-ion, may not be suitable for power deregulation, because their voltages during the plateau region, a relatively flat region of the voltage–DoD curve, are significantly higher than the optimal-energy operating voltages of modern processors. However, there are new, high energy density, battery technologies that

are good candidates for deregulated systems. We provide test results for one such candidate, the recently commercialized silver-zinc secondary cell technology.

This chapter makes the following contributions.

1. We describe and evaluate battery and system models that are suitable for the design of power deregulated systems. They consider battery DoD dependent parameters including battery voltage and internal resistance.

2. We provide a design process for the codesign of deregulated embedded systems and their energy delivery subsystems. We give guidelines for battery and processor selection, and indicate the operating conditions favorable for power deregulation and conventional regulated systems.

3. We provide the first third-party characterization of the new silver-zinc battery technology for embedded applications, including data on discharge curves, impedances, and internal resistances. The characterization results are used in battery model validation and lifetime simulation.

2.2 Related Work and Background

The rest of this chapter focuses on evaluating power deregulation, because it is more general than, and often outperforms, other unregulated design techniques. It is possible to instead use uniprocessors with large performance margins at high initial battery voltages, thereby imposing tight limits on performance or resulting in short battery lifespans. For example, the Telos ultra low-power wireless node [9] uses an unregulated MSP430 RISC processor. In contrast, power deregulation exploits multi-core processors to improve lifespan and performance.

We consider the heterogeneity of batteries and workloads. Some existing work also considers the match between workloads and non-ideal energy sources, e.g., batteries and ul-

tracapacitors. Cho et al. [12] proposed a method called dynamic voltage regulator scheduling (DRS) that selects either DC–DC converters, linear regulators, or bare batteries depending on processor frequency and voltage. Cao et al. [11] designed a battery/ultracapacitor hybrid system for electric drive vehicles. The match between the output voltage of a battery and the discharge current of the workload is considered during system design. Their design is more appropriate when the system is powered by multiple energy sources, while we focus on single-battery systems and more thoroughly develop the relevant modeling and system design techniques. Choi et al. proposed DC–DC converter-aware dynamic voltage scaling [13]. This technique selects the most energy optimal voltage point and gate sizing for both processors and the converter. While this work improves efficiency of the voltage regulator, it does not aim to reduce PCB area.

2.3 Power Deregulation

In this section, we briefly describe power deregulation, the process of removing voltage regulators and their peripheral capacitors and inductors from an embedded system, allowing it to operate directly on battery power. The battery output voltage thus serves as the operating voltage of processors.

The time-dependent decreasing battery voltage limits the operating frequency of processors, but need not reduce system performance. Constrained by timing requirements, the processor frequency scales in proportion to its operating voltage, which decreases over time as the battery discharges. Figure 2.3 shows how power deregulation works during the battery discharge cycle. At the beginning of discharge, the battery output voltage is higher than the normal processor operating voltage using a voltage regulator, resulting in higher operating frequency and therefore higher throughput than necessary. This wastes energy. Therefore, a portion of cores in the multi-core system are turned off to save energy, leaving just enough active cores to maintain throughput. As discharge continues, battery voltage

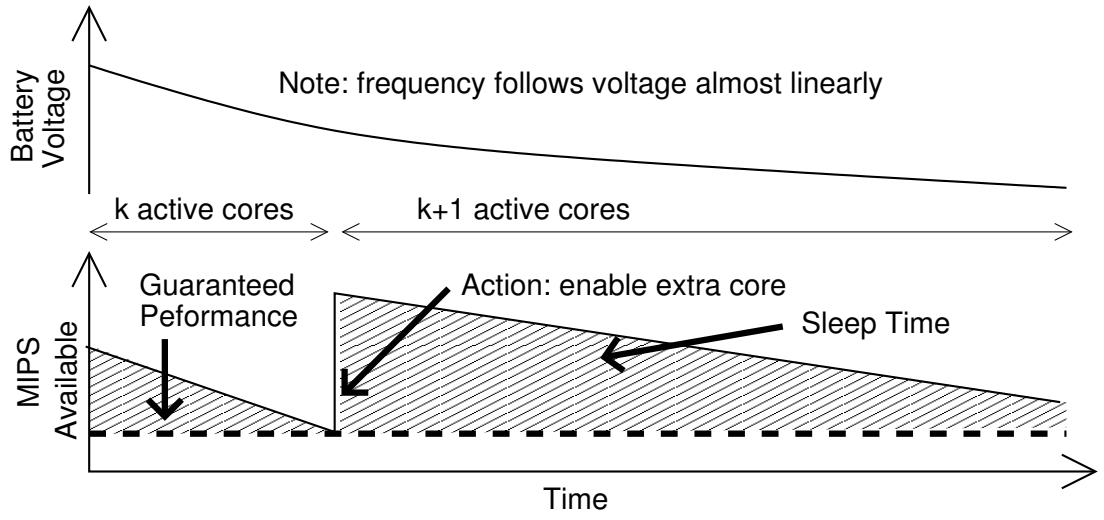


Figure 2.3: The operating timeline of power deregulation.

decreases slowly. Processor frequency decreases accordingly, reducing the overall system throughput until it is inadequate. The system then activates another core to compensate for performance degradation. This process continues, until the processor cut-off voltage is reached. Short-term temporal fluctuations in battery voltage can be neglected, except when the battery voltage is near the threshold voltage for activation of additional cores. Hysteresis within the control software can be used to prevent frequent core activation and deactivation, as described in Subsection 2.5.2.

2.4 Lifespan for Battery-Powered Systems

Battery energy capacity and average power consumption can be used to roughly approximate the total lifespan of an embedded system. However, this is only a first-order approximation. Generally, only a portion of the battery energy can be used because the system shuts down at some cut-off voltage. Such effects need to be considered when evaluating battery lifespan. Previous work has simultaneously considered delay and battery lifespan [14], but at the cost of using a fixed operating voltage that depends on a voltage regulator.

System lifespan is influenced by battery capacity, as well as the time-varying battery

internal resistance and embedded system power consumption. The battery discharge effect can be expressed as follows:

$$capacity = \int_0^T (P_{int}(t) + P_{sys}(t))\,\mathrm{d}t. \tag{2.1}$$

Battery internal power consumption $P_{int}(t)$ changes with battery output voltage and internal resistance. In power deregulated systems, the embedded processor power consumption $P_{sys}(t)$ is affected by core activation. These time-varying effects must be considered when estimating lifespan. Short-term variations of battery discharge current on battery lifespan are discussed separately in Subsection 2.5.2.

2.5 Model

In this section, we first present battery voltage dependent power and performance models for regulated and power deregulated systems. We then describe a battery model that is suitable for computing system lifespan. Finally, we combine these two models to compare conventional regulated and power deregulated systems.

2.5.1 Power Consumption and Performance Model

In power deregulated systems, battery voltage determines processor voltage, which in turn constrains frequency [15]:

$$f = k(V_{dd} - V_{th})^a / V_{dd}. \tag{2.2}$$

In this work, we set the performance constraint of a power deregulated system to that of a regulated chip-level multiprocessor (CMP) system with the same number of cores operating at the optimal-energy DVFS voltage. The optimal DVFS voltage of a regulated system allows it to operate with the lowest power consumption but still meet its performance requirement. In this work, we mainly consider a regulated system using non-linear buck

converters. We refer to this as the "regulated system" in the rest of this chapter. In general, this performance tolerance can be adjusted based on workload properties and design goals. For the same workloads, the deregulated and regulated systems must satisfy the same performance requirements:

$$N_{dereg} \cdot f_{dereg} = N_{reg} \cdot f_{reg} \propto Performance. \tag{2.3}$$

This model tracks the system performance, and accounts for the influence of single-thread frequency (f) on the number of functioning cores (N) in a multi-core system.

Using Equation 2.2, we can substitute the voltage for frequency in Equation 2.3 for every operating frequency point. The performance constraint then depends on the relationship between battery external voltage (for power deregulation) and regulator output voltage (for regulated systems):

$$N_{dereg} \cdot V_{external} = N_{reg} \cdot V_{reg}. \tag{2.4}$$

This relates core activation to battery external voltage: a new core is activated when $V_{external}$ drops below a threshold.

2.5.2 Battery Model

There are several battery models for embedded systems. Chen et al. proposed a runtime electrical battery model [16] consisting of resistor-capacitor networks. It captures the short-term and long-term response of the battery. Rao et al. developed a charge well based battery model that captures the recovery effect [17].

The battery model used in our evaluation is based on the classic model developed by Lawrence et al. [18]. We select this model because it is suitable for most batteries used in mobile embedded systems. This model consists of time-varying resistance and a voltage source in series. This level of complexity is necessary because these are the two major factors that affect long-term battery discharge behavior and external voltage. It is sufficient

to accurately model battery external voltage over long time scales. Here we ignore the temperature dependence of battery internal resistance, as the battery temperature fluctuation is small in most low-power embedded systems [16]. The model parameters can be measured during battery characterization. Our testing results on several silver-zinc battery samples show that this battery model captures the discharge behavior (see Section 2.6).

During our evaluation, we only consider battery voltage changes due to the long-term discharge effect, e.g., stable power management state changes and workload changes. We then divide the battery discharge process into several phases. Within each phase, we use the average current to model the discharge current to the load processors. Therefore, we assume the systems operate with constant workload, i.e., the equivalent resistance of one core does not change within one battery discharge phase. We model the processor equivalent resistance using a fixed resistance, and ignore short-term changes in processor power consumption due to runtime workload variation. This approach is valid for the following reasons.

- The fluctuations in current resulting from millisecond-scale [19] changes to workload power consumption are filtered out by the large built-in capacitance of batteries, making it unnecessary to adjust to them [20]. Longer time scale changes in workload power consumption that influence battery voltage can be dealt with in the same way as changes in battery voltage due to discharge.

- We explicitly consider the change in battery efficiency due to short-term variations in discharge current using an efficiency parameter μ ($0 < \mu < 1$) [14], which can be calculated based on the discharge current profile.

- Using average discharge current to model the workload will not bias our evaluation towards either regulated or power deregulated systems. Even given the same average discharge current, there remain differences between the current profiles of different systems due to cache misses, context switches, and application phase changes. However, the resulting variation in battery lifetime between different pulse discharge

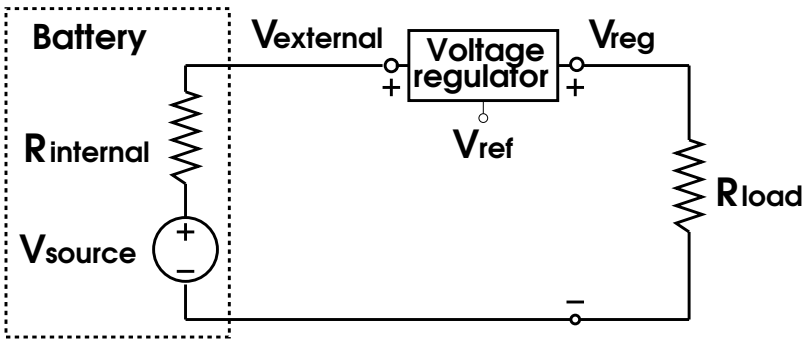


Figure 2.4: Battery model used for a regulated system.

currents is negligible (3% [14] in both systems).

- We calculate the workload, equivalent loading resistance, and battery efficiency parameters separately for each discharge phase during long-term battery discharge.

Figure 2.4 shows the quasi-steady-state battery model used in our evaluation. We model the equivalent resistance of a processor as $R_{load} = R_{core}/N$, in which N is the number of cores that are functioning in the multi-core system and R_{core} is the equivalent resistance of an individual core. The equivalent resistance of a core depends on its operating frequency.

In a conventional regulated system, the voltage converter output voltage V_{reg} is constant for a fixed DVFS voltage point. In a power deregulated system, the voltage regulator in Figure 2.4 is removed, and $V_{external}$ directly powers the load. $V_{external}$ follows:

$$V_{external}(t) = \frac{R_{core}V_{source}(t)}{R_{core} + N_{dereg}R_{internal}(t)}. \tag{2.5}$$

The power consumption of a power deregulated system in the battery internal resistance and external load follows:

$$P_{dereg} = \frac{V^2_{source}(t)}{R_{internal}(t) + R_{core}/N_{dereg}}. \tag{2.6}$$

The power consumption of a regulated system has a similar form, except that the regulator introduces power loss during voltage conversion, which is captured in a conversion efficiency

variable: $conv\_eff$ ($0 < conv\_eff < 1$).

2.5.3 System Level Model

We integrate the performance and battery models for use in design-time battery lifespan estimation of both regulated and power deregulated systems. Power consumption (Equation 2.6), multiplied by a discharge efficiency parameter μ, is used within the battery discharge behavior model (Equation 2.1). Core activation behavior in power deregulation is given by Equation 2.3. There is not an analytical form of the time-dependent parameters $V_{source}(t)$ and $R_{internal}(t)$; they can only be obtained from battery discharge curves, as shown in Figure 2.2b. Therefore, we used a discrete event simulator that captures the time-varying battery discharge effect to obtain the system lifetime T using these models and battery characterization data.

To determine the circumstance in which power deregulation improves system lifespan over conventional power regulation, two main factors must be considered: battery external voltage and system power consumption. The former has been derived in Equation 2.5. The relationship between power consumption on the processors of both systems can be derived from the performance and battery models:

$$P_{dereg,core} = P_{reg,core} \cdot \frac{V_{external}(t)}{V_{reg}} \cdot conv\_eff. \tag{2.7}$$

Whether a power deregulated system consumes less power or has a higher battery external voltage than a regulated system depends on the factor $(V_{external}/V_{reg}) \cdot conv\_eff$, which changes as the battery discharges. Section 2.7 will further explain the factors that influence the battery external voltage discharge curve.

2.6 Characterization and Analysis of a New Silver-Zinc Battery Technology

For a power deregulated system to operate reliably and efficiently, an appropriate battery technology must be used. Battery voltage range becomes more important in the absence of a voltage regulator. Most available high energy density battery technologies, such as lithium-ion, have plateau voltages ranging from 2 V to 4 V, which are too high for current and future deep submicron processors. We have evaluated power deregulation when used with a new, high energy density, low-voltage silver-zinc battery technology, whose two plateau voltages are around 1.8 V and 1.5 V.

We have obtained sample silver-zinc batteries from ZPower, the company commercializing this technology, and have performed internal capacitance and resistance measurement to build and verify the model described in Section 2.5. These measurements have also allowed us to determine whether the conditions for reliable operation of power deregulated systems hold for this battery technology. Battery samples with 27 mAh and 100 mAh capacities were evaluated. It is expected that the silver-zinc battery will be available for commercial use later this year.

2.6.1 Charging and Discharging Characteristics

We measure the battery discharge curve to obtain its plateau voltage range, and thus verify that this voltage range is within the operating voltage range of a processor using current technology.

The discharge curves of the batteries are directly measured using the ZPower Button Cell Charger/Cycler. The tests are performed with a discharge current of 5 mA for 27 mAh cells and 10 mA for 100 mAh cells.

The discharge curve of a 100 mAh cell is shown in Figure 2.5. Silver-zinc batteries have a discharge curve with two voltage plateaus. The discharge voltage of the cell ranges from

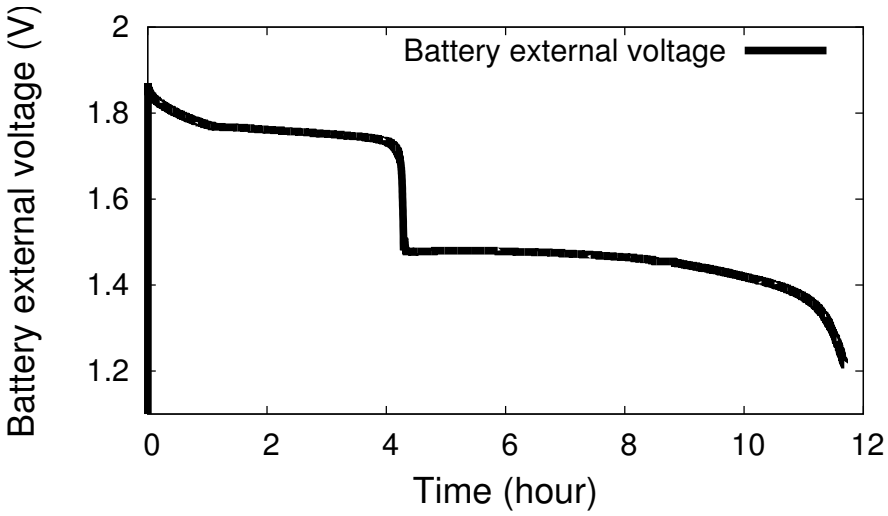
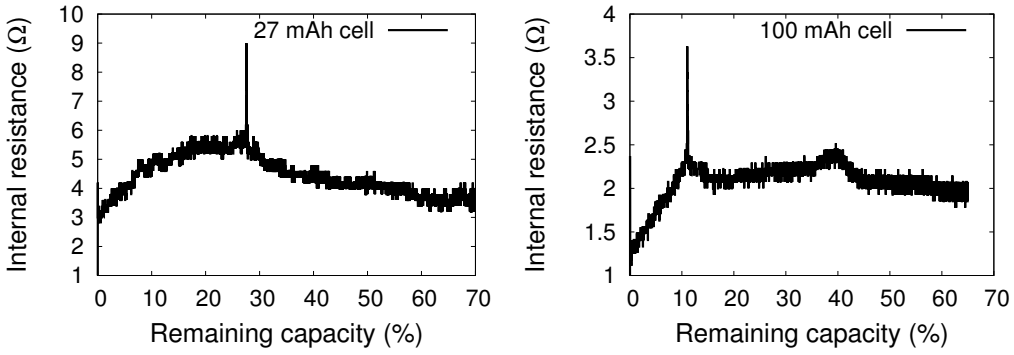


Figure 2.5: Discharge curve for a 100 mAh silver-zinc cell.

1.2 V to 1.8 V, which is suitable for directly powering processors using 180 nm technology or older (e.g., ARM7TDMI–S).

2.6.2 Internal Resistance

We use pulse testing to measure battery internal resistance, to capture its change during discharge. The internal resistance of the 27 mAh cell is measured at a discharge current of 1.5 mA and a pulse current of 0.15 mA, and that of the 100 mAh cell is measured at a discharge current of 10 mA and a pulse current of 1 mA.



(a) 27 mAh cell

(b) 100 mAh cell

Figure 2.6: Internal resistance test.

The internal resistance of the battery reaches a sharp spike the same time as the high-to-low battery voltage transition. These two phenomena, both caused by the chemical phase change, result in core activation in power deregulated systems. This spike lasts for a few minutes, which is longer than the response time for core activation, allowing power deregulation to adapt to the change.

The resistance of a 27 mAh cell is on average 2.2 times that of a 100 mAh cell as shown in Figure 2.6, i.e., the internal resistance decreases with the increasing battery capacity. When doing lifetime simulation (see Section 2.7) of batteries with different energy capacities, we determine the internal resistances using a linear function of battery capacity based on the two measured capacities [21].

2.6.3 Impedance Test

We performed battery internal impedance tests and calculated the battery internal capacitance and inductance. This result allows us to conclude that LdI/dt effects will not become more severe when power deregulation is used.

The impedance of the battery is measured by applying alternating voltage to the battery at different DoDs, and measuring the phase of the discharge current. We use an Agilent 33250A function generator to generate an alternating voltage with fixed frequency. This voltage is applied across the battery in series with a 10 kΩ resistor, whose resistance is much larger than the battery internal resistance and thus keeps the amplitude of the discharge current constant. The discharge current is sampled using an NI USB 6210 data acquisition card.

Our measurements show that the magnitude of the battery impedance reduces and the phase becomes positive as the frequency increases, indicating that the impedance of the battery is capacitive instead of inductive. To develop a more detailed model of the internal capacitance of the battery, we calculated the capacitance for the 27 mAh battery using an RLC model [18], which we simplified by considering only one resistance (R_c). The

capacitance remains the same (approximately 100 mF) during discharge, with only 0.05% variation. Therefore, we consider the capacitance independent of battery DoD.

2.6.4 Summary of Silver-Zinc Battery Evaluation

The measurement results in this section show that the silver-zinc battery is a good candidate for power deregulated systems. Its two-level output voltage falls in the operating voltage range of a modern processor. Neither the internal resistance spikes nor the battery impedance is a source of reliability problems when directly using the battery to power the workload. We use the internal resistance measurement values in our system lifetime simulation, and the result in Section 2.9 shows that this silver-zinc battery does not degrade system lifetime compared to regulated systems. Nevertheless, there are drawbacks to the silver-zinc battery technology. At present, only ZPower and Yardney rechargeable silver-zinc batteries are commercially available later this year. To summarize, silver-zinc batteries provide an example of a battery technology appropriated for power deregulation.

2.7 Factors Affecting System Lifetime

Using the models introduced in Section 2.5, we are able to discuss how the choice of battery and processor types affect system lifetime of both power deregulated and regulated systems.

A power deregulated system operates until the battery voltage drops below the processor's minimum operating voltage. It is important to consider two phenomena that can reduce system lifespan. (1) If significant energy remains in the battery when its output voltage reaches the minimum operating voltage of the processor, the remaining energy cannot be used. (2) If the battery voltage is significantly higher than the nominal voltage of the processor, power consumption is higher than necessary for a given performance level, reducing battery lifespan. Either of these two factors can reduce system lifespan.

The above two factors and our battery and system models can be used to inform design

decisions for power deregulated systems. The lifetimes of both regulated and power deregulated systems are influenced by processor and battery characteristics, including battery internal resistance and discharge curves, the minimum operating voltage of the processor, the equivalent resistance of the processor, and the number of processors in the system. We evaluate each variable in this section.

2.7.1 Internal Resistance

We evaluate the influence of $R_{internal}$ values on battery lifetime and system design. As can be seen in Figure 2.2b, $R_{internal}$ increases at the end of the battery life cycle for most batteries. This results in a rapid decrease in $V_{external}$, as described in Equation 2.5, resulting in termination when the processor (or regulator) cut-off voltage is reached.

$R_{internal}$ is, in general, inversely related to battery capacity [21]. *Due to the change in battery internal resistance, system lifetime does not increase linearly with battery capacity.* Furthermore, larger battery capacity does not necessarily prolong system lifetime for power deregulation, because the system lifetime does not change monotonically with the corresponding internal resistance change. To allow designers to fairly compare the difference in battery lifespans given different battery capacities, we consider the scaling effect in battery internal resistance when selecting different battery capacities in system lifespan evaluation.

Figure 2.7 demonstrates the change in battery lifetime normalized to the battery capacities of both regulated and power deregulated systems when $R_{internal}$ increases. In the presence of large $R_{internal}$, there is significant voltage drop across $R_{internal}$ in both regulated and power deregulated systems, greatly reducing battery external voltage and causing the cut-off voltage to be reached quickly. A power deregulated system has longer battery life compared to a regulated system due to its lower cut-off voltage and absence of regulator overhead. Systems using buck-boost converters suffer less from lifespan degradation, as they have much lower cutoff voltages. But the lifespan increase is limited due to their high energy loss during voltage conversion. When $R_{internal}$ is very small, power deregulation reduces

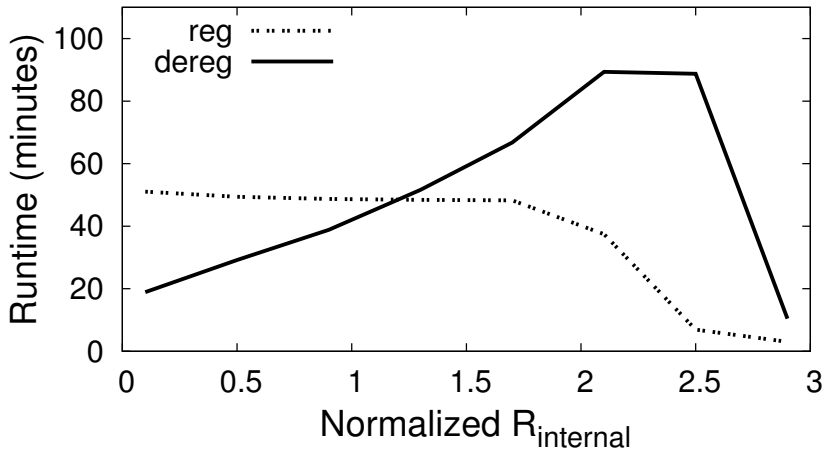


Figure 2.7: Battery lifespans for power deregulated and regulated systems with varying battery internal resistance values.

lifespan. The internal voltage drop is ignorable, and $V_{external}$ is similar to V_{source}, increasing system power consumption quadratically. Therefore, power deregulation becomes power inefficient and drains battery capacity faster, reducing the system lifetime.

2.7.2 Lowest DVFS Voltage

The processor's lowest DVFS voltage V_{min} determines the minimum output voltage of a voltage regulator. Power deregulated systems work well when processors have V_{min} values close to the battery plateau voltage.

When V_{min} is comparable to the voltage of the plateau region of the battery external voltage curve, in a power deregulated system $V_{external}$ can decrease to V_{min}. In this case, power deregulation results in similar or lower power consumption than voltage regulation, as described in Equation 2.7. This benefit can exceed the negative effect of sometimes using a voltage higher than the most power-efficient value. In contrast, when V_{min} is much lower than the plateau region of $V_{external}$, the additional processor power consumption will reduce battery life in the power deregulated system.

In addition to satisfying the requirement improved by V_{min}, the battery should have a low enough voltage for reliable operation. To power processors using deep sub-micron

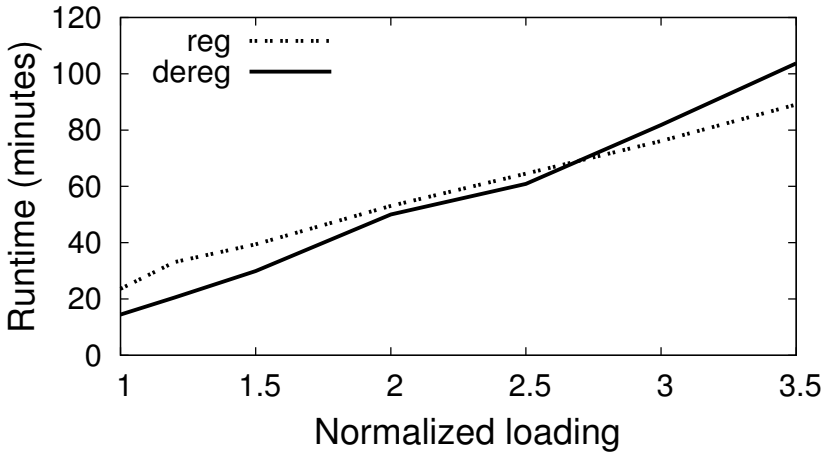


Figure 2.8: Battery lifetime changes with processor load resistance.

technologies, batteries should have nominal voltages around 1 V. Most existing batteries do not meet this requirement. Some new battery technologies, however, are suitable for power deregulation. For example, the plateau voltages of recently commercialized silver-zinc batteries range from 1.8–1.2 V, which is close enough to processor nominal voltages for energy-efficient operation. We characterized silver-zinc batteries to determine whether they were appropriate for power deregulation (see Section 2.6).

2.7.3 Impact of Core Count

The number of available cores in the system determines the granularity with which a power deregulated system can adapt to changing battery voltage by enabling new cores. The system performance constraint in Equation 2.3 is set based on the performance of the regulated system when operating at the optimal energy DVFS voltage. When a new core in a power deregulated system is activated, the instantaneous increase in performance might potentially exceed requirements, wasting power. Having more cores tends to reduce this discretization problem. On the contrary, a regulated system does not benefit in this way from more cores.

2.7.4 Processor Equivalent Resistance

In a power deregulated system, the equivalent resistance of the processor must be large enough to prevent a substantial voltage drop across the battery internal resistance. The equivalent resistance of embedded processors, although varying with different applications, usually tens of ohms or fewer, which is smaller than the internal resistances of some batteries. Enabling additional processor cores reduces the load resistance. This reduces the battery external voltage, as explained in Section 2.5. If the equivalent resistance of the processor is comparable to $R_{internal}$, this sudden voltage drop will lead to enabling more processors to maintain the performance. This causes further $V_{external}$ drop, which in turn shortens the system lifetime(Figure 2.8). Thus, when selecting a processor, the designer should determine whether the equivalent resistance of the processor is large enough to sustain an adequate operating voltage during most of the battery discharge curve.

2.8 Deregulated System Design Procedure

We now summarize the design procedure when both conventional regulated and power deregulated system design styles are considered. The goal is to maximize battery lifetime. Note that even when a power deregulated system has no or slight lifetime advantage, it may still be appropriate due to reduced PCB area and system cost. We assume that the designer has the freedom to choose among several types of batteries with different capacities and prices.

Figure 2.9 can help with the design process. For each type of battery, the constraints imposed by battery internal resistance (and hence battery capacity), processor equivalent resistance, number of cores, and the lowest DVFS voltage point of the system are considered. The designers should follow the steps below.

1. The change in $R_{internal}$ due to the scaling of battery capacity must be considered. According to Figure 2.7, the normalized battery lifetime depends on the (time-varying)

battery internal resistance. Thus, the designer should refer to this plot to determine the range of $R_{internal}$ for which battery lifespan is acceptable. The range of appropriate battery capacities can then be determined.

2. The designer should consider the equivalent resistance of the processor. Based on the range of $R_{internal}$ values, the designer can confirm that processor resistance is high enough ($> 3\times$ according to Figure 2.8) relative to $R_{internal}$ to prevent severe voltage drop across the battery internal resistance when enabling new cores.

3. The number of available cores is already determined by the processor type. However, the designer must verify that the core count is sufficient to prevent lifespan shortening due to coarse granularity core activation in response to reducing battery voltage.

4. The designer should verify that the voltages of the batteries being considered are compatible with the minimum DVFS voltage V_{min} of the processor. At this point, the designer has tentatively selected one or a few processors and battery types. Based on the known V_{min} value of the processor, the designer should check each battery plateau voltage to determine whether this V_{min} leads to a longer lifetime using the power deregulated system.

5. If the power deregulated system satisfies all these requirements, it is entered into a list of candidate designs. A regulated system using the same type of battery is also considered, and is added to the candidate list if appropriate.

6. When all types of batteries have been considered, the designer should refer to other system constraints (e.g., the operating temperature of the battery or total weight and size of the system) and rule out inappropriate candidates.

The remaining designs in the candidate list satisfy the designer's requirements. As a result, the one most appropriate given the designer's battery lifespan, PCB area, price, and reliability requirements is selected.

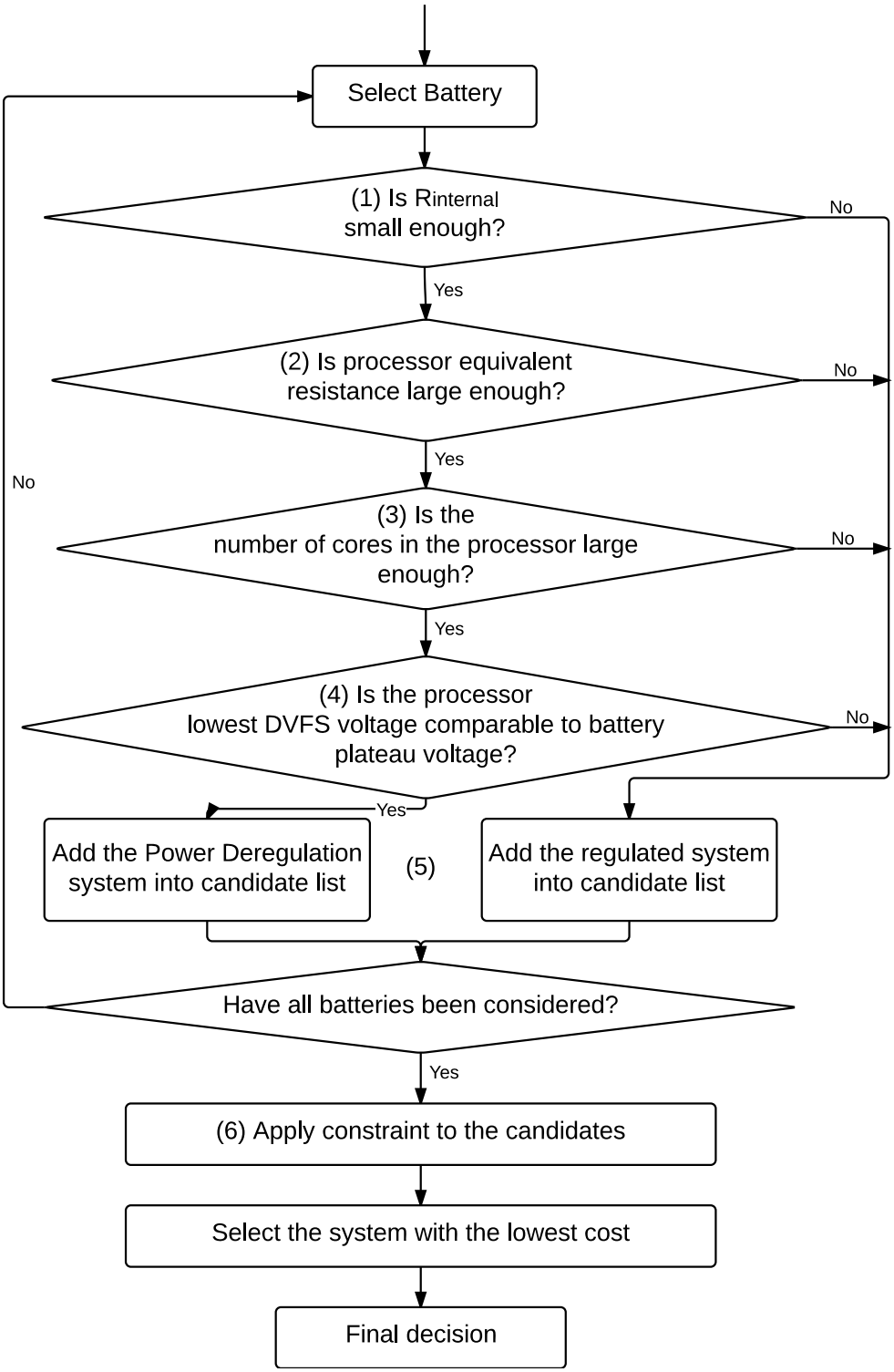


Figure 2.9: Design and battery-processor selection for power deregulated and regulated systems. The numbers correspond to the steps described in Section 2.8.

2.8.1 Battery Lifespan Comparisons

We performed system lifetime simulation for five battery types. The simulation is done for regulated systems using buck/buck-boost converters and power deregulated systems. We leave out comparisons across other unregulated systems because they focus on low performance applications and fail or significantly shorten battery lifespans for multi-threaded applications [9]. In contrast, power deregulation works for both low and high performance applications. In these lifetime comparisons, we use processors with the most suitable number of cores, and the processor operating voltage matches the plateau voltages of the battery, i.e., we considered system designs for which power deregulation is likely to be appropriate.

The system used for evaluation is designed to be suitable for operating with all five battery types. The minimum DVFS voltages of the processors range from 1.2 V to 1.6 V for the first four batteries, while that voltage of processors used with silver-zinc batteries is 1.0 V. The buck-boost converter has a cut-off voltage of 0.8 V. The conversion efficiency is 85% for the buck converter, and 72% for the buck-boost converter [5]. Recent research shows that the peak efficiency of DC-DC converters can reach 98% [22]. However, this only happens under low loading current; efficiency is more typically 80%. The quiescent current of both buck and buck-boost voltage regulators during sleep mode is set to 5 mA [6].

We perform system lifetime evaluations on two applications: multi-core sensor network applications and multi-media devices. Sensor networks are common applications suitable for power deregulation. Such applications limit available PCB area and are constrained by battery energy. Multicore processors can be useful for parallelizable applications such as signal processing [23], making power deregulation especially favorable. In our simulation, we set the awake/sleep duty cycle to 1%, which is common in sensor network applications [24].

Multi-media applications are also potential candidates for power deregulation. They generally use more cores, and can achieve better parallelism when running multi-threaded programs. This reduces throughput loss when activating new cores. Both these applications have good thread-level parallelism. In our simulation, we use a parameter (0.85) to describe

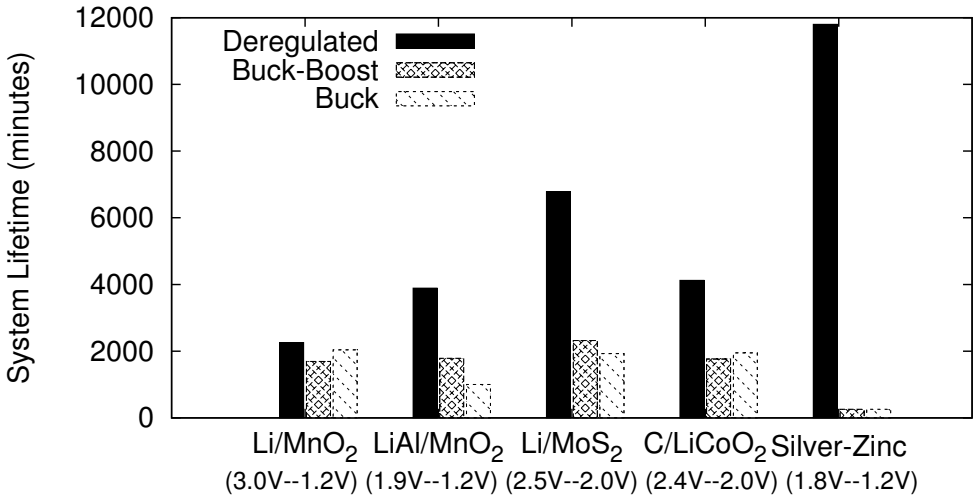


Figure 2.10: Lifespan comparison for sensor network apps.

the performance loss due to non-ideal parallelism.

Figure 2.10 compares system lifetimes of sensor network applications for five battery types. Power deregulation clearly out-performs both regulated systems for all batteries, achieving an average system lifespan improvement of 9.8× over buck-boost regulated systems and 10.2× over buck regulated systems. This benefit in system lifetime mainly comes from eliminating energy loss in voltage regulators, especially during sleep mode.

Figure 2.11 shows the system life comparisons for multi-media applications. Multimedia application lifetimes are shorter than sensor network application lifetimes, because there is no sleep cycle in these multi-media application. In addition, battery discharge currents are higher. Given appropriate processor–battery matches, the power deregulated systems outperform buck converter regulated systems for each battery type, having on average 17% longer battery life, while having a lifetime at most 25% less than the buck-boost converter regulated system. The power deregulated system has similar or better battery life compared to a buck-boost regulated system when $V_{external}$ is close to the optimal energy DVFS voltage of the processor, or when the battery discharge curve have a flat tail, leading to a higher cut-off voltage. On the other hand, buck-boost regulated systems outperform deregulated systems when batteries have high plateau voltages. Note that even though power

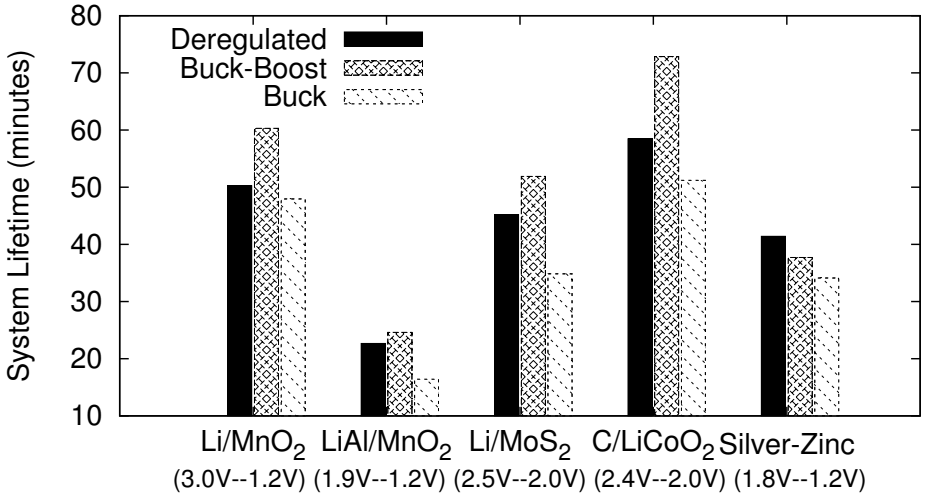


Figure 2.11: Lifespan comparison for multi-media apps.

Table 2.1: HTC Smart Phone Power Consumption Breakdown

	LCD	CPU	Wifi	GPS	Audio	3G
Power (mW)	412.042	196.392	234.083	0.0	248.130	0.0

deregulation does not significantly improve multi-media application lifespans, it reduces PCB area compared to regulated systems.

2.9 Discussion, Caveats, and Future Directions

In this section, we discuss some problems and concerns that are not covered in our earlier evaluation of power deregulation. This includes comparison with linear regulated systems and power deregulation of systems with analog devices.

2.9.1 Linear Regulators

Although a linear converter regulated system can save significant PCB area relative to a buck or buck-boost converter regulated system, it results in an inferior system lifespan compared to a power deregulated system (which saves even more PCB area). First, linear

regulators commonly have conversion efficiencies of 60% [25]. Although the efficiency can be better when the input voltage is close to the output voltage, this condition is also favorable to power deregulated systems. Second, linear regulators have an input-to-output voltage difference requirement, i.e., they have higher cut-off voltages than power deregulated systems, reducing the amount of usable battery energy. Third, the capacitance of a linear regulator is even smaller than that of a non-linear regulator [26], and as a result is smaller than that of a battery. Therefore, systems using linear regulators may be more susceptible to $L\mathrm{d}I/\mathrm{d}t$ effects than Power Deregulated systems.

2.9.2 Analog Components

Our discussions thus far have focused on the digital processor within an embedded system. Real systems often also contain analog components that rely on a separate power supply. Table 2.1 lists the average power consumption breakdown of five users of Android phones when browsing videos on the YouTube website [27]. The digital components, including CPU and audio devices, account for 41% of the total power consumption, with an additional 38% if a digital display is used. This reduces the power savings due to power deregulation. It may be possible to replace some of the analog components with parts that may be powered directly from batteries. For instance, a typical LCD relies on backlights, which require a high drive in voltage (higher than 5 V) between electrodes. In contrast, OLED (Organic LED) displays do not require backlights to function, making it conceivable that such a display could compensate for some decrease in battery voltage during operation. OLED displays can operate at 2.55 V [28], which can be supplied by most batteries. If some components require very precise control of supply voltage, it may be necessary to use a conventional regulated design for the relevant portion of the embedded system.

2.9.3 Future Battery Technologies

Our discussion on example battery technologies has focused on the silver-zinc battery, the voltage range of which is compatible with current processor technology. For future processor technologies with lower voltage, use of nanoelectrolytes and nanoelectrodes may lead to new battery technologies with appropriate output voltages that are lower than 1 V. Wu et al. [29] introduced a silicon nanotube battery anode with a plateau voltage of 0.2 V. Armstrong et al. [30] described a lithium anode with a plateau region at 0.1 V while maintaining high energy density.

2.10 Conclusions and Acknowledgments

Battery-system codesign is necessary to maximize the battery lifespans of power deregulated systems. We have shown that designers should ensure that battery discharge curves, capacities, and internal resistances are compatible with processor equivalent resistances and power consumptions. The silver-zinc battery technology is a candidate for power deregulation of deep submicron processors. Power deregulation can reduce PCB areas and, if designed properly, maintain similar or greater lifespans compared to conventional regulated systems.

CHAPTER 3

Spatially- and Temporally-Adaptive Communication Protocols for Zero-Maintenance Sensor Networks Relying on Opportunistic Energy Scavenging

3.1 Introduction

Long unattended lifespans are important for wireless sensor networks because they are often deployed in locations that are difficult to access. Replacing sensor network node components or retrieving nodes can be prohibitively expensive. Therefore, sensor node lifetime is of central importance. Most sensor nodes last a few months; some last a few years. This implies that component or node replacement is necessary in long-term deployments. Our work provides a novel method of designing wireless sensor networks to operate for decades without periodic repair or replacement of sensor nodes.

Several factors constrain sensor node life time. The battery lifespan constraint is typically encountered first. Rechargeable batteries used in many sensor nodes wear out faster than other components. Even when energy constraints are loose, batteries have lifetimes ranging from 1.5 year to 6 years [8, 31]. Battery lifetime is even worse among sensors deployed in outdoor environments because the batteries suffer from ambient temperature and humidity

changes, which can degrade the battery lifespans by 75% [31]. Frequent depletion/recharge cycles can result in 20% decrease in battery lifespans. Consequently, sensor nodes must periodically have their batteries replaced, wasting time and effort. Even for sensor nodes that use batteries with lifespans exceeding that of the application, it is still commonly necessary to eventually collect the nodes because many batteries contain toxic materials such as heavy metals [3]. The elimination of batteries opens the possibility of designing wireless sensor nodes that are suitable for long-term, one time deployment. Battery-less wireless sensor nodes using energy scavenging devices as power supplies can meet this requirement.

Eliminating batteries can potentially make long-term deployment maintenance free. However, existing battery-less sensor network designs still have problems. Previous works do not eliminate the need for lifetime-constraining energy storage devices [32, 33]; such nodes use supercapacitors or battery-supercapacitor hybrid systems for energy storage, which still constrain the sensor node lifespan [34, 35]. In contrast, we consider a wireless sensor node design that completely eliminates the need for lifetime-constraining energy storage devices. The sensor nodes use ambient power sources, such as solar power, wind power, and water flow as their energy source, replacing the battery with long-term, stable energy scavenging devices. As a result, the sensor nodes no longer suffer under batter-imposed lifetime constraints.

Eliminating batteries requires changing the sensor node architecture and modifying the sensor network protocol design to specifically adapt to the changes in sensor node activities caused by using ambient power sources: sensor nodes wake up at imperfectly predictable times. Therefore, existing protocols that rely on pre-scheduled data transmission do not work. Storage and time synchronization constraints in the sensor node architecture design also limit the use of battery-less energy scavenging. We propose a protocol that takes power source availability and remaining memory into consideration.

We consider applications in which the sensing target moves infrequently, as is often the case for long-term environmental monitoring. They fall into one of the two categories: (1)

the wireless sensor network only needs to sense when events occur, and these events also provide energy or (2) sensor nodes are deployed in an environment that provides access to time-varying energy sources that are event-independent. These properties are commonly seen in existing distributed sensing applications.

We propose the following modifications to existing wireless sensor network architectures: (1) replace the power supplies of sensor nodes with energy scavenging devices, which may be wind or water turbines, piezoelectric generators, or solar panels, depending on the application and available energy sources; (2) adapt routing decisions based on the spatial and temporal distributions of power availability for nodes; and (3) store intermediate results to non-volatile memory, when appropriate, to compensate for loss of power.

Based on these changes, we describe a new sensor network design that is well suited to energy scavenging. Our major contributions follow.

- We describe a novel routing protocol that works well for sensor networks using battery-less energy scavenging. The routing protocol reacts to imperfectly predictable changes in ambient energy sources. It minimizes the end-to-end latency of packet transmission and achieves 1.3–3$\times$ performance improvement over existing designs for four environmental sensing applications.

- We describe architecture changes to sensor nodes that make them more appropriate for use with intermittent and imperfectly predictable power supplies.

- We categorize commonly used sensor network applications and provide application dependent guidance for designers considering battery-less energy scavenging.

We will discuss the details of our design in later sections and compare it with existing design strategies.

3.2 Related Work And Motivation

Large scale, long-term monitoring applications rely on low-maintenance sensor nodes because in-field repairs and replacement are expensive. Energy scavenging nodes offer a compelling solution for reducing maintenance costs; they gather ambient energy from the environment and consequently eliminate the need for battery replacement. However, existing energy scavenging nodes still have practical limitations that either constrain their lifetimes or prevent them from being used to build large-scale network applications. In this section, we point out shortcomings in existing design techniques, summarize our contributions, and argue for new sensor node and network architectures.

3.2.1 Engery Scavenging with Battery Assistance

Many researchers have proposed using sensor nodes powered by a combination of rechargeable batteries and scavenged energy including power derived from the sun, ambient vibration, wind, water flow, and the motion of animals. Among these, solar-power is most widely used due to its high and stable energy density.

Raghunatha et al. [36] describe a procedure for designing efficient solar-powered sensing systems. Taneja et al. [37] provided network architecture and node design guidelines for micro-solar powered sensor networks. Researchers have also developed routing protocols suitable for solar-powered sensor networks. Voigt et al. [38] proposed and compared two such protocols. Existing work has used solar-powered sensors for environmental monitoring. Mainwaring et al. [33] developed a wireless sensor network using solar-powered Mica Motes for habitat monitoring on Great Duck Island. They used solar panels that can provide between 60 and 120 Watts in full sunlight. The sensor networks in this chapter used rechargeable batteries to store scavenged energy. However, the batteries themselves constrain the sensor node lifetime and hence limit the applicability of these nodes in very long deployments.

3.2.2 Why Battery-Less Energy Scavenging?

Batteries are the primary energy storage devices in many sensor nodes. However, they typically have short lifespans and frequently limit the lifespan of the whole node. Consequently, researchers have proposed the following methods of eliminating them.

Replace the rapidly degrading rechargeable battery with a supercapacitor. Minami et al. designed Solar Biscuit [32], a battery-less sensor network that only relies on a solar panel and a supercapacitor to power the sensor node. They also developed a routing protocol suited to the long charging time of the supercapacitor. The use of supercapacitors can extend the lifespan of sensor nodes, but only to a point; supercapacitors also degrade. Studies have reported that supercapacitors have 13%–15% capacity degradation and double their internal resistance after one year of power cycle testing [34, 35]. Moreover, the lifetimes of supercapacitors are temperature-dependent. The expected lifetime halves with every 10Celsius increase of ambient temperature [39, 40]: supercapacitors degrade when deployed outdoor. In addition, supercapacitors are 4–10 $\times$ more expensive than rechargeable batteries with the same energy capacities and densities. Supercapacitors remain unsuitable for our goal of long-term deployment, although this may change in the future if their reliable lifespans are increased.

Reduce the number of charge/discharge cycles in the battery. This can be done by attaching a supercapacitor or energy scavenging device to the battery, and only discharging the battery when the other energy supplies fail. Jiang et al. proposed a multi-stage energy transfer system that uses a solar panel together with a super-capacitor as the first stage and a rechargeable battery as the second stage [41]. They argued that, when ambient power is sufficient to power the sensor node, the system can avoid discharging or charging the battery. This approach can increase the sensor node lifetime to 4 years given a 10% duty cycle. However, it does not eliminate batteries, which must eventually be gathered from the environment. We argue that the sensor node lifetime problem should be solved by removing the battery entirely.

A few researchers have proposed battery-less sensor nodes with the goal of increasing sensor node lifetime. Philipose et al. [42] attached an RFID to a battery-less sensor node, powering the sensor node via the RFID reader. Their work completely removed the energy storage device. Vyas et al. [43] and Patel et al. [44] combine a battery-less, wireless tag and a low power sensor node for use in a passive sensor. However, these works require that energy be directed to each active sensor from an external radio frequency energy source. This prevents use in distributed applications. Ng et al. [45] design a near-body network with battery-less wearable biomedical sensors to monitor patient physiological state. This solution is appropriate for body-range transmission and consists of only a few nodes. Our work focuses on applications requiring larger scale distribution of sensors.

3.2.3 Node Design and Protocol Support

Existing routing protocols and sensor node architectures are not well suited to energy scavenging sensor networks. We propose an architecture that is based on partial knowledge of the spatial and temporal properties of ambient power sources.

The designers of energy scavenging sensor networks face special difficulties in maintaining functionality and performance. Many environmental power sources, such as solar, wind, and water flow have intermittent availability. This complicates routing protocol and node architecture design. First, the scavenging sensor network is dynamic: its connectivity structure changes dynamically depending on environmental power source status of each node. Routing protocols must adapt to these changes. Second, the wake-up schedule of sensor nodes cannot be controlled by the designer. Algorithms that rely on coordinated activations at pre-determined times to sense, transmit, or receive cannot be used. Third, sensor nodes lose their power sources at imperfectly predictable times, leaving little time for nodes to react by transmitting data or preserving it in non-volatile memory. The designers of energy scavenging wireless sensor networks must consider these domain-specific challenges.

How well would existing sensor node and communication protocol designs fare in a

battery-less environment? Prior work has proposed flooding-based routing techniques for energy scavenging sensor networks [32, 36]. These protocols require nodes to wake up at pre-determined times and use redundant transmissions to compensate for the lost messages. They are adequate for small-scale networks. However, they would perform poorly in larger-scale applications such as environmental monitoring. In medium- to large-scale networks, flooding overwhelms the communication channels, resulting in high latencies and data loss rates. Geographic routing is another popular candidate protocol; it is easy to implement and may not require pre-determined schedules for transmissions. However, only using geographic information for routing in energy scavenging sensor network can cause packets to be trapped in inactive nodes (as described in detail in Section 3.3). The main weakness of existing geographic routing protocols in this application is their failure to account for the fact that nodes frequently become unavailable at imperfectly predictable times and some nodes are available more frequently than others. This limits network scale, and prevents operation when many nodes are frequently inactive. Other candidate protocols require scheduling nodes to transmit at precise times, and therefore cannot be used in battery-less energy scavenging networks.

3.3 Problem Definition

Our goal is to provide routing protocol and node design techniques suitable for indefinitely deployed sensor networks. We begin by eliminating the use of energy storage devices with highly constrained lifespans. Given temporary losses in node power, we attempt to determine the design techniques yielding the highest end-to-end successful data delivery rate under a (designer-specified and application-specific) constraint on acceptable latency.

We now describe our model of an energy-scavenging sensor network. Ideally, when all nodes have access to sufficient power, they form a connected graph N containing $|N|$ nodes, in which there is a directed edge between two nodes if the first can successfully transmit

directly to the second. In this situation, nodes can transmit sensed data to the base station using existing routing protocols. However, the probability of all nodes concurrently having power at any particular time is small. In each time interval, only a subset of sensor nodes have enough scavenged power to operate. The graph of these nodes, N_i, is referred to as the *active subset* for the ith time interval and does not change within the interval. Note that intervals can be defined to end whenever the active subset changes.

We model the network packet transmission latency using active subsets. It is likely that two temporally adjacent active subsets N_i and N_{i+1} have a non-empty intersection due to the temporal correlation of the power source (described in detail in Section 3.5). As shown in Figure 3.1, packets from a faraway node S can travel through multiple active subsets N_{i0} through N_{i3} via their intersections to finally arrive at B.

The packet transmission delay is the sum of t_{trans_i} and t_{int_i} for all active subsets along the path. t_{trans_i} is the transmission latency to populate packets within an individual active subset N_i. t_{int_i} is the time interval between active subset N_i and the next active subset N_{i+1}, which is the time when sensor nodes are inactive.

We define the latency of data transmission in the network to be the time required for all nodes to send their sensed data packets to the base station, i.e., the maximum packet transmission delay. In reality, some packets will be dropped due to channel overuse or collision. Others will be trapped in nodes that wake up infrequently. Thus, we define *packet delivery rate* to be the percentage of packets that reach the base station at a particular time. When we compare protocols in later sections, we will compare transmission latencies associated with particular packet delivery rates.

The limited predictability of ambient power sources reduces designer control of wireless sensor networks. It prevents the designer from using pre-computed routing paths and requires routing protocols that adapt to changes in the ambient power source. Pre-computed routing may falsely send packets to nodes that are not in the temporal intersection of two adjacent active subsets, preventing packets from further transmission. As can be seen in Figure 3.1,

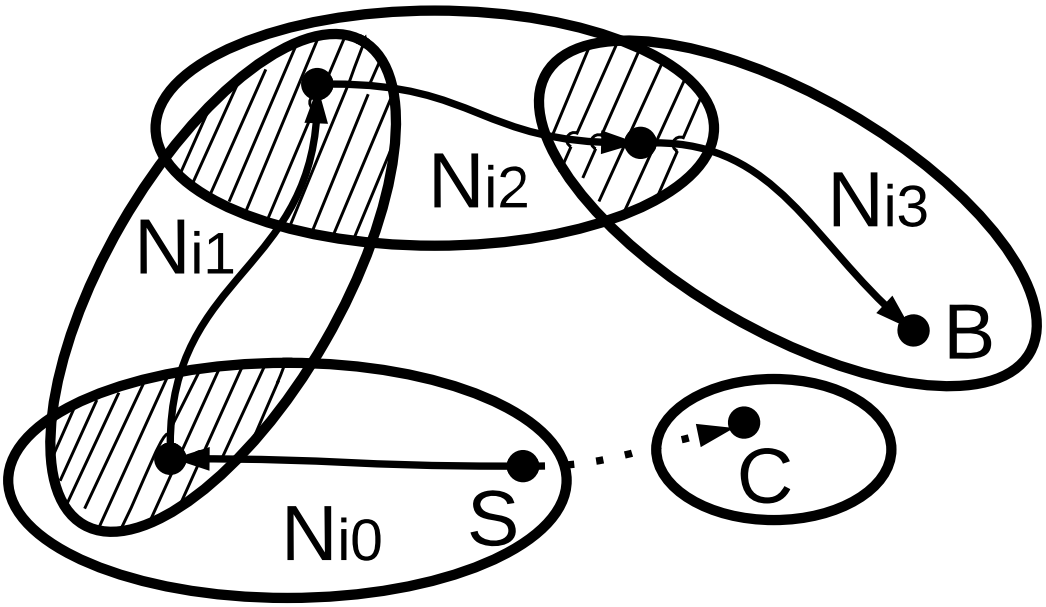


Figure 3.1: Battery-less energy scavenging sensor network operation. A node S transmits its packets to base station B through temporally intersecting active subsets.

node S may send its packet to another node C, which is geographically closer to the base station B but is in a rarely awake, isolated subset. This can delay packet transmission; the 4-hop path via N_{i0} through N_{i3} is faster. Instead, we consider the temporal and spatial statistics of ambient power sources to dynamically change the routing for every active subset and avoid trapping packets.

We will further describe the design and implementation of our battery-less wireless sensor network routing protocol in the following sections.

3.4 Design and Implementation

We have developed a routing protocol for sensor nodes that become active at imperfectly predictable times. Sensor node activation events occur at random times prescribed by the temporally and spatially correlated statistical processes used to characterize ambient energy sources. Therefore, nearby nodes have similar behavior. We describe a protocol informed by these properties, and then discuss four variations of it appropriate for a range of sensing

applications.

We will first describe a network architecture suitable for energy scavenging environmental monitoring applications. The sensor nodes are widely deployed, and a group of sensor nodes share a base station. The nodes transmit their sensed data to the base station, potentially indirectly via other nodes. The base station uses a high-capacity power supply that will require replacement every few years at much lower cost than recovering all sensor nodes. In the remainder of this section, we describe how we perform routing given this network structure.

3.4.1 Sensor Node Architecture

We modify the sensor node architecture to support battery-less energy scavenging and to guarantee that the sensed data contain valid time stamps.

3.4.1.1 Energy Scavenging Devices

We consider two major modifications to a conventional wireless sensor network node: (1) removing the battery and (2) attaching an energy scavenging device, e.g., a solar panel, wind turbine, water turbine, or piezoelectric device [46]. Our design relies on non-volatile memory to save states between active intervals. Most existing sensor nodes contain non-volatile memory. Depending on the power source distributions, it may also be appropriate to use a higher-performance processor and network interface than is typical, in order to quickly finish processing and communication tasks when power is available.

3.4.1.2 Time Synchronization

Sensor nodes must generally associate times with samples. The local timer in a sensor node stops working when power is lost, and it needs to be re-synchronized when power is available. Activation events happen at random times, meaning that sensor nodes cannot determine how long their timers have been inactive when reactivated. A node can synchronize with the

base station or its neighbors, which have valid local timers. However, this synchronization still has some delay because a node can be synchronized only when it is in the same active subset with other nodes maintaining correct times. The samples gathered during the synchronization delay will have incorrect time stamps.

We now describe a technique to compensate for time uncertainty in intermittently powered sensor nodes. Nodes are desynchronized by power loss. They attempt to resynchronize with neighboring nodes, but until that time they mark data samples with time ranges that are later used by the base station, together with other sample time stamps, to more accurately estimate when the data were gathered. This approach can achieve time stamp errors of less than 80 minutes for 91% packets in a medium scale (500 nodes) sensor network. The environmental monitoring applications we consider usually gather samples several times a day (as described in Table 3.1). Therefore, samples with time stamp errors of minutes or even hours are acceptable.

Our proposed process works in the following steps.

1. The local timer of a sensor node contains an invalid value at the beginning of deployment. Every sensor node will first try to synchronize with the base station to obtain the correct time.

2. The sensor node refreshes its local timer value (stored in a non-volatile memory) every fixed time interval (e.g., one minute) and at every time it is synchronized to nodes with correct timers. When a sensor node again has access to power, it restarts its local timer using the stored time stamp in the memory.

3. Every node attaches a node identifier and a unique packet identifier maintained in non-volatile memory to each packet, guaranteeing that for a particular node, packets with smaller packet identifiers are always produced earlier than the ones with larger packet identifiers. Packets also carry time stamp upper bounds (initialized to the latency constraint) and lower bounds (initialized to the value of the local timer). A

valid bit is also included, which is set to "true" only when the local timer is known to have low error at the time of packet generation.

4. The base station refines the time stamps of packets, working within a fixed time interval. This time interval should be long enough such that most packets generated at a similar time arrive at the base station before the end of the interval. The generation time of a packet (with identifier p) is estimated by examining the packets with the closest smaller and larger packet identifiers (noted as l and u). If these packets have valid time stamps, the upper and lower bounds for the packet of interest are refined as follows:

$$packet_p.lower = packet_l.upper,$$
$$packet_p.upper = packet_u.lower. \tag{3.1}$$

3.4.2 Precomputed Information

Our routing technique bases decisions on temporal power source distributions. Each node knows the probability of having sufficient power for computation and transmission and its distance to the base station. This information can either be precomputed from the power source distribution or gathered after deployment.

The *activity rate* P_{active} of a sensor node is the probability of it having sufficient power to compute and transmit data [46]. If a sensor node is powered by a wind turbine, it will only be activated when the wind speed exceeds a threshold value providing enough power. The probability of a sensor node being awake is $\int_0^{w_{th}} f(x, \lambda, k)$, where w_{th} is the threshold wind speed and $f(x, \lambda, k)$ is the wind speed distribution at that location. The node activity rate can be computed from historical wind speed distribution data.

The distance d from the sensor node to the base station is the number of wireless communication hops, assuming all nodes are active. This distance can be gathered during

network deployment.

3.4.3 Routing Protocols

In order to determine whether existing protocols are sufficient for indefinitely deployed energy scavenging sensor networks, we make comparisons between several existing routing protocols: simple flooding, geographic routing, buffer size dependent routing, and undirected routing. We also evaluate the Ambient Energy Aware routing protocol we designed specifically for this problem.

Flooding is the most commonly described routing protocol for energy scavenging sensor networks [32, 36]. It is easy to implement in sensor nodes with limited computation power and has adequate performance for small-scale networks in which the data generation rate is low. Multiple nodes keep copies of the same packet; thus, even when some of the nodes lack power and become inactive, the redundant copies of the packet are transmitted by other nodes. In larger networks, flooding faces two problems: (i) limited resources and (ii) undirectional transmission. Simple flooding creates redundant packets that can exceed sensor node memory capacities resulting in dropped packets. In addition, flooding protocols suffer greatly from limited channel capacity. The network-wide channel capacity is constrained in battery-less energy scavenging sensor networks by the possibly frequent deactivation of nodes that are (temporarily) without power. Flooding creates duplicate packets and easily overwhelms the network. This is especially problematic when nodes wake up infrequently.

Based on the above observations, we now consider routing protocols appropriate for battery-less energy scavenging sensor networks. These protocols aim to avoid the poor performance caused by limited buffer size, packet collision, and the randomness of node active intervals. These protocols have the following characteristics.

- *Acknowledgment.* We enable acknowledgment by both sender and receiver nodes. Receivers acknowledge packet acceptance. When a sender receives the first acknowledgment, it broadcasts a drop request to its neighbors, allowing all but the node that

transmitted the first acknowledgment to drop their copies of the packet. This avoids unnecessarily use of memory and communication resources for duplicate packet copies in multiple nodes, while preserving at least one copy of the packets. The acknowledgment delay described later reduces the probability of acknowledgments collision.

- *Directional transmission.* Packets transmit along the path with the smallest expected latency to reach the base station. When receiving a packet, instead of acknowledging immediately, a receiver use a *ranking function* to delay the acknowledgment. The time delay is set to give priority to nodes with higher probability of successfully reaching the base station. The choice of ranking function is a key design feature. Later in this section, we will discuss the selection of the ranking function in greater detail.

- *Random Back-Off.* Nodes perform random retransmission back-off on packet collisions to avoid future collisions.

Given these starting properties, we consider several candidate routing protocols. Three of the protocols (geographic routing, buffer size dependent routing, and undirected) adapt well-known algorithms to the energy scavenging sensor networks considered in this work.

Geographic routing [47]. Nodes always accept packets from the neighbors that are geographically further away from the base station. This increases the probability of the packet reaching the base station. The ranking function is $delay_i = t_{unit} \cdot d_i$, in which $delay_i$ is the delay of the ith node to acknowledge, t_{unit} is a unit time period, and d_i is the number of hops from the ith node to the base station, assuming all nodes are active. As mentioned earlier in this section, this distance d can easily be gathered during node deployment. One significant drawback of this approach is the likelihood of creating holes in the network: some nodes are geographically closer to the base station, but rarely active. Packets will sometimes be transmitted to these nodes shortly before they become active and then remain trapped for a long time.

Buffer size dependent routing. Nodes accept packets from neighbors with less free space in their message buffers. This avoids buffer overflows, which may result in data loss. The ranking function is $delay_i = t_{unit} \cdot (b_{max} - b_i)$, where b_{max} is the maximum buffer size and b_i is the remaining size in the node receiving buffer. Unfortunately, this protocol does not consider the importance of transmission directions. Nodes that frequently wake up or are closer to the base station receive more packets. As a result, the ranking function will assign these nodes longer delays, forcing packets to be forwarded to nodes from which packets are less likely to reach the base station.

Undirected protocol [47]. Nodes are assigned random priorities using ranking function $delay_i = t_{unit} \cdot random(n_i)$, where $random(n_i)$ returns a random number between 1 and n_i, the number of one-hop neighbors of node i. This protocol has the benefit of simplicity but is usually inefficient because random prioritization results in slow, indirect paths for many packets.

The final protocol is a novel approach which is specifically designed to overcome challenges in battery-less sensor networks:

Ambient Energy Aware protocol. Nodes with the highest probabilities to be (possibly indirectly) connected to the base station have the highest probabilities of accepting packets from their neighbors. This protocol makes use of the statistical data on power source availability and the node activity rate (as described in Subsection 3.4.2) to compute the ranking function. Our goal is to combine the best attributes from the four protocols described above, while also using available information on the statistical properties of the ambient power source. The ranking function is $delay_i = t_{unit} \cdot (b_{max} - b_i) \cdot d_i / P_{active,i}$, where b_{max} is the maximum buffer size, b_i and d_i are the remaining buffer size and distance from the base station of node i, and $P_{active,i}$ is the activity rate of node i. The drawback of this protocol is its requirement for additional memory on sensor nodes to store power source statistical properties. Fortunately, this is not a problem in practice because the statistical data can be preprocessed and reduced to a single number: the node activity rate.

The most appropriate protocol depends on application characteristics. Therefore, we will compare the existing and new protocols described above under a variety of operating conditions.

3.5 Evaluation

In this section, we provide evaluation results for the protocols discussed in the previous section. We first evaluate the protocols when used in different sensing applications. Then we determine the sensitivity of protocol quality metrics to variations in application characteristics.

3.5.1 Experiment Setup

Our evaluation considers sensor networks that scavenge energy from wind. To model changes in wind conditions, our simulation takes location-dependent time-varying wind traces as input. However, the raw measured wind speed data for large regions and long duration are not publicly available. Therefore, we generate similar wind traces with statistical properties based on recorded wind speed distributions [48]. Wind speed traces are then fed into the discrete-event simulator described later in this section.

Our generated wind speed traces have the following properties. First, the trace for each particular location has particular temporal correlation values. Second, traces at different locations have spatial correlation. We can represent the wind speed traces by a group of correlated Weibull random variables. To generate these random variables, we make use of two sources of information: (1) a regional wind speed atlas [48] and (2) spatial and temporal correlation models.

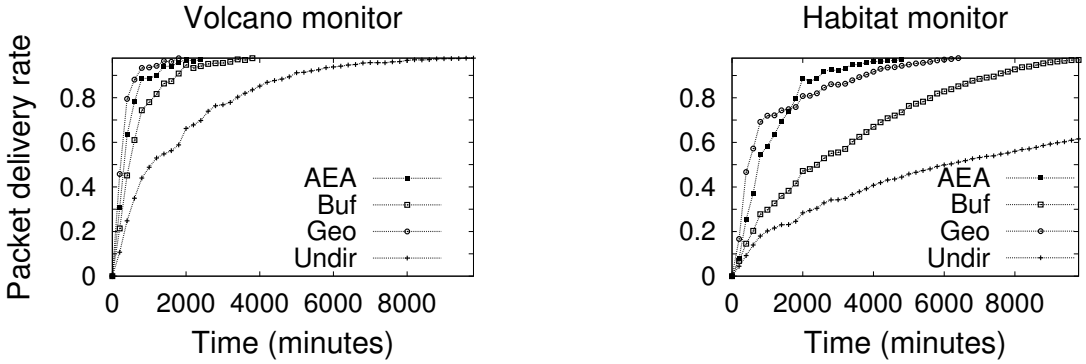
The World of Wind Atlases houses a publicly available archive of wind data from many regions around the world [48]. For a given region, the atlas logs detailed location-specific information about wind patterns. Specifically, for each location on the map, the atlas records

the Weibull parameters (k, λ) that describe variation in wind speed at several altitudes. We select the wind atlas of one island in Denmark as the source of wind data, because its wind atlas is representative of many coastal regions in the world.

We use existing wind speed spatial correlation models [49–51]. The spatial correlation coefficient for wind speed at two different locations is exponentially dependent on the distance between the locations: $c = \exp\left(-d/d_0\right)$, where d is the distance between the locations and d_0 is called *correlation distance*. This is the distance at which the correlation between two locations equals $exp(-1)$. We model the temporal correlation coefficient as exponentially dependent on elapsed time based on the observations of Archer and Jacobson [49].

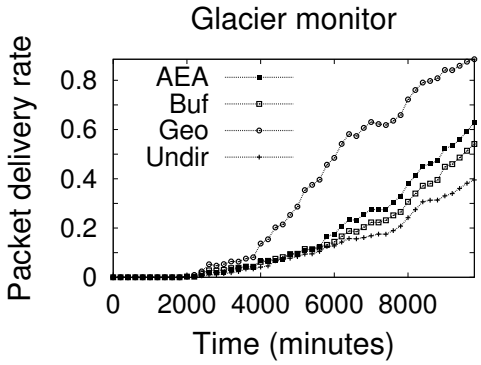
We use a wind trace duration equal to the maximum tolerable latency for data transmission. In environmental monitoring applications, it is common for the base station to processes or transmit data in a daily or weekly pattern [33, 52]. Some packets will carry incorrect time stamps due to the intermittent power loss of the sensor nodes. Their time stamps will be refined after reaching the base station using the method described in Section 3.4, with the largest error being the packet transmission latency. Thus, packets arriving at the base station later than one week are likely to have indistinguishable time stamps. They are considered to be invalid. We set our latency constraint to be one week.

We developed a discrete event wireless sensor network simulator in which changes in wind speed are events. We generate wind speed traces based on the parameters described above [48, 49] and provide them to our simulator. The simulator models the activation and deactivation of sensor nodes when the input wind speeds at the sensor locations change, forming different active subsets. Within each active subset, it simulates the behavior of sensing and data transmission of sensor nodes executing any of the four protocols described in Section 3.4. The network packet delivery rate, node level channel utilization, and transmission latency are recorded.

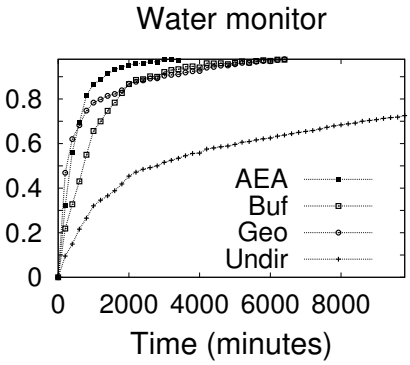


(a) Volcano monitor

(b) Habitat monitor



(c) Glacier monitor



(d) Water quality monitor

Figure 3.2: Packet delivery rate comparison for four protocols under different applications.

3.5.2 Application Based Evaluation

Several sensor network applications are suitable for using energy scavenging techniques. Table 3.1 lists these applications and their properties [33, 52–58]. In the context of these applications, we evaluate four protocols that were discussed in Section 3.4: Ambient Energy Aware routing (*AEA*), geographic routing (*Geo*), buffer size dependent routing (*Buf*), and undirected routing (*Undir*). Our results show that no single protocol is best for all applications.

We consider four wireless sensor network applications from Table 3.1 as our examples for energy scavenging sensor network: habitat monitoring (*habitat*) [33], volcano monitoring (*volcano*) [52, 53], glacier monitoring (*glacsweb*) [54], and meteorology and hydrology monitoring (*water*) [55]. Each application has a set of parameter values that are used in their setup, as described in Table 3.1. These four applications have distinct parameters. *habitat*, *volcano* are both small- or medium-scale sensor networks (10–100 nodes evenly distributed in an area with 2 km radius) in which sensor nodes require a moderate amount of ambient power (100–200 mW), and only perform data sensing several times a day. *water* is a large-scale network (600 nodes) with moderate power requirement. *glacier* is a medium-scale sensor network but requires high power supply to sensor nodes. We first evaluate our protocols on these four applications with fixed parameter values listed in Table 3.1.

Figure 3.2 shows the time-dependent variation in packet delivery rate throughout the whole network for these four applications. Based on the application constraints described in Subsection 3.5.1, we set the maximum tolerable packet transmission latency for all packets to be one week. While some applications (*habitat* and *volcano*) have acceptable delivery rates using simpler routing protocols, others (*water*) require our proposed Ambient Energy Aware Protocol to achieve satisfactory performance (i.e., packet latencies less than a week). For *glacier*, none of the protocols considered achieve the packet delivery rate within the latency constraint.

This difference in performance of different protocols can be briefly explained by ref-

Table 3.1: Applications And Their Parameters

Application	Sensors	Energy source	Project period	Threshold wind speed (m/s)	Required power (mW)	Scale	Node type	Transmission range (m)	Sample rate	Sample size	Buffer size
CORIE	conductivity, temperature, depth	solar	1–3 years	9.0	N/A	large	N/A	N/A	several / day	N/A	N/A
Pipe network	MEMS based accelerometer	water flow	1 year	N/A	N/A	large	Gopher	300–32000	150/sec	N/A	256 kB or 16 GB
habitat	temperature, humidity, thermopile	solar or wind	9 months	3.0	68	small	mica2 mote	100	6/day	2 B	512 kB
Volcano	microphone, seismometer	solar or wind	5 years	4.0	100	small	TMote sky	200–400	1–2/day	128 B	4 MB
Glacsweb	pressure, temperature, orientation, external conductivity, strain	solar	1 year	9.0	500	med-ium	mica2 mote	100	6/day	16 B	512 kB
meteorology and hydrology	water level and temperature	water flow	3 years	4.0	100	large	mica2 mote	100–200	6/day	16 B	512 kB

erence to the abstract model described in Section 3.3. First, let us consider the *habitat* and *volcano* applications. They share common properties: small- or medium-scale and medium power requirement. The first property means that there are always sufficient sensor nodes active at the same time. This results in an active subset N_i that covers a large portion of the network. The second property guarantees that the sensor nodes are likely to have enough memory to store the sensed data without dropping packets. As a result, using simpler approaches such as geographic routing is sufficient. Second, we consider *water*, which has a large-scale sensor network and wider distribution of sensor nodes. This results in a large active subset N_i that has the potential to overwhelm the network with traffic. It is therefore favorable to transmit packets through nodes that are more likely to be active, rather than concentrating the traffic close to the base station. For this application Ambient Energy Aware routing works best. Third, *glacier* has a high node power consumption requirement. Only very high wind speed can provide enough power to activate a node. The resulting active subset N_i is a sparse network, making it less likely to cover the whole network for a given latency constraint. As a result, none of the protocols considered have good performance for this application. To improve the situation, one might switch to a more powerful energy scavenging device, e.g., a larger turbine.

3.5.3 Delivery Rate for Varying Parameters

No single protocol is best for all application scenarios. To assist application developers to select the most appropriate protocol, we now show the application-dependent parameter ranges for which each protocol is best suited.

From the example applications described above, we observe several variable parameters, which are subject to changes due to special requirements of the application or user preference. These parameters include: (1) network scale, which is the total number of nodes in the network; (2) sample size, which is the size of data gathered at every sensing event; (3) required power consumption, which is the maximum power required for a sensor node to

perform data processing and transmission; and (4) maximum direct transmission range.

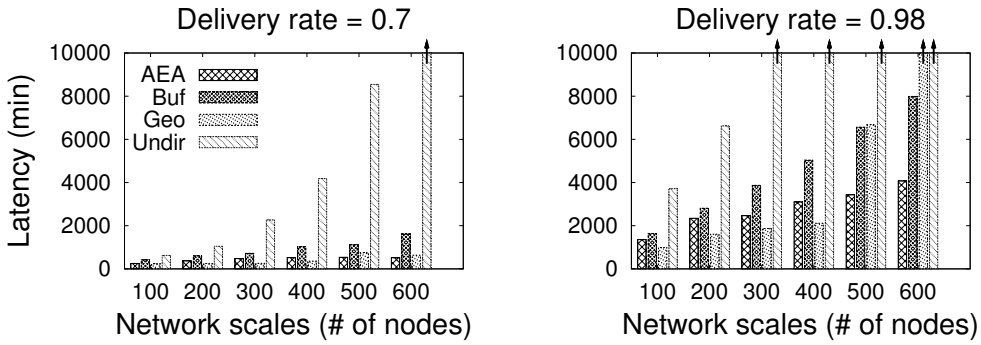
We evaluate how each parameter affects selection of a protocol using a series of parameter studies. The parameter studies are conducted by varying parameters one at a time while keeping others constant. We select a set of constant values for these parameters: medium scale network (300 nodes distributed in a $3\,\text{km} \times 3\,\text{km}$ region), small sample size (16 B), medium required power consumption (100 mW), and transmission range for commonly used nodes (400 m).

The properties of the sensor network influence the selection of protocols. On the one hand, in large-scale sensor networks, the edge of the network is far from the base station, requiring more hops to send packets to the destination. Since the wind speed at each sensor node location varies randomly over time, the probability of transmitting a packet from a distant node to the base station depends on the node activity rate along the transmission path. It will often be best to route through a longer path with higher activity rate, rather than the shortest path. Ambient Energy Aware routing uses knowledge of ambient power source statistical parameters, giving it an advantage over other routing techniques in large-scale sensor networks. On the other hand, simpler protocols that simply send many redundant messages may be sufficient in small-scale networks.

Figure 3.3 shows transmission latencies for different network scales. When the packet delivery rate requirement is loose, all protocols have similar performance. However, performance differs under stricter packet delivery rate requirements. Geometric routing works well for small- and medium-scale networks, while buffer size dependent routing works well only for large networks. In contrast, Ambient Energy Aware routing works well for small, medium, and large networks.

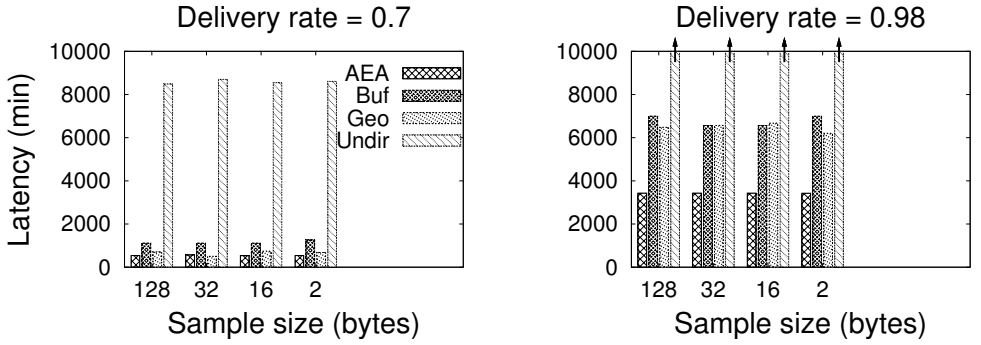
3.5.3.1 Sample size

Sample size is the amount of data gathered by a node per sample. The sample size affects how long a sensor node spends sensing and transmitting data. If the data gathering and



(a) Low delivery rate (b) High delivery rate

Figure 3.3: Transmission latency comparison of four protocols when network scale changes. Arrows indicate that latency exceeded application latency constraint.

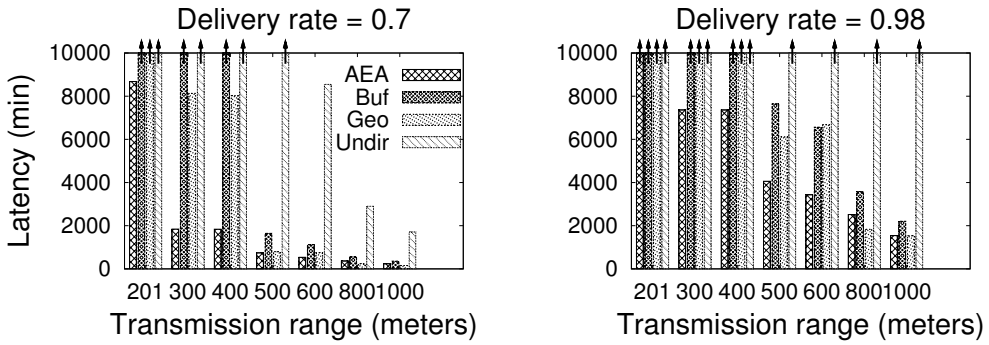


(a) Low delivery rate (b) High delivery rate

Figure 3.4: Transmission latency comparison of four protocols when sample size changes. Arrows indicate that latency exceeded application latency constraint.

transmission time is too long, the sensor node will not have enough time for data transmission, reducing the packet transmission rate. We now discuss the effect of sample size on the network latency.

Figure 3.4 shows packet finish time as a function of sample size for the four protocols. The packet delivery rates of all protocols stays almost the same when sample size changes. This is because the sensing time takes up only a small portion of the sensor node active time even for the largest sample size that we consider. We conclude that the sample size for similar applications will not significantly affect the packet transmission latency.



(a) Low delivery rate

(b) High delivery rate

Figure 3.5: Transmission latency comparison of four protocols when transmission range changes. Arrows indicate that latency exceeded application latency constraint.

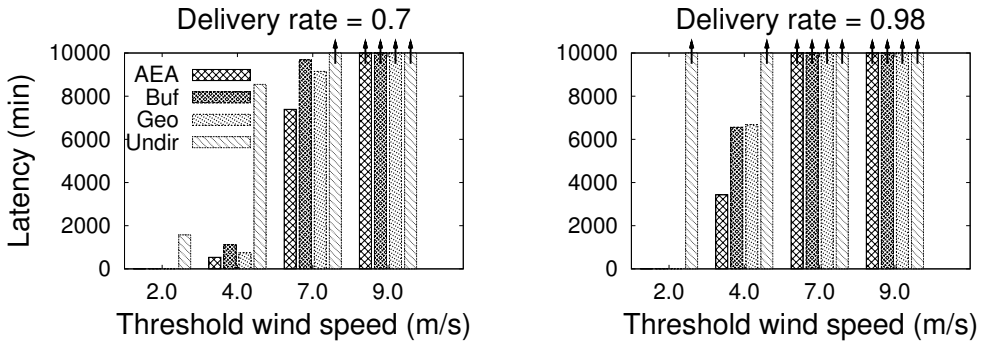
3.5.3.2 Transmission range

The transmission range of sensor nodes affects routing protocol selection. The larger the transmission range, the more immediate neighbors per sensor node. This is especially important to energy scavenging sensor networks, in which only a subset of neighbors are active at any time.

Figure 3.5 shows the packet delivery rate as a function of node transmission range for the four routing protocols. The transmission range of a sensor node depends on the radio device used in the sensor node, normally ranging from 100 m to 1 km. When the transmission range is large, simple protocols work as well as Ambient Energy Aware routing. When the transmission range is small, there are so few immediate neighbors per sensor node that selection of neighbors during routing is critical. In this case, Ambient Energy Aware routing protocol outperforms existing alternatives. This result holds for all packet delivery rate requirements.

3.5.3.3 Power requirement

The power required by a sensor node and an ambient power trace determine when the node will be active. Required power depends on the hardware platform and the software workload.



(a) Low delivery rate

(b) High delivery rate

Figure 3.6: Transmission latency comparison of four protocols when threshold wind speed changes. Arrows indicate that latency exceeded application latency constraint.

In our example application, ambient power is determined by wind speed. We define the minimum wind speed required by an application as *threshold wind speed*.

The impact of threshold wind speed on packet transmission latency in the sensor network is plotted in Figure 3.6. When the threshold wind speed is low, all protocols have short packet transmission latencies. For high threshold wind speeds, the Ambient Energy Aware protocol is superior. However, when threshold wind speed is very high, no protocol can finish transmitting enough packets during the active time periods to meet application requirements.

3.6 Protocol Selection

This section describes the relationship between protocol performance and both channel utilization and per-node packet delivery rate. It then explains a strategy for selecting an appropriate communication protocol.

3.6.1 Channel Utilization

Channel utilization is an important additional metric for evaluating our proposed protocols. The channel capacity is limited by the activity rates of nodes in an energy scavenging sensor network. We show the channel utilization for every node in a 500-node sensor network using

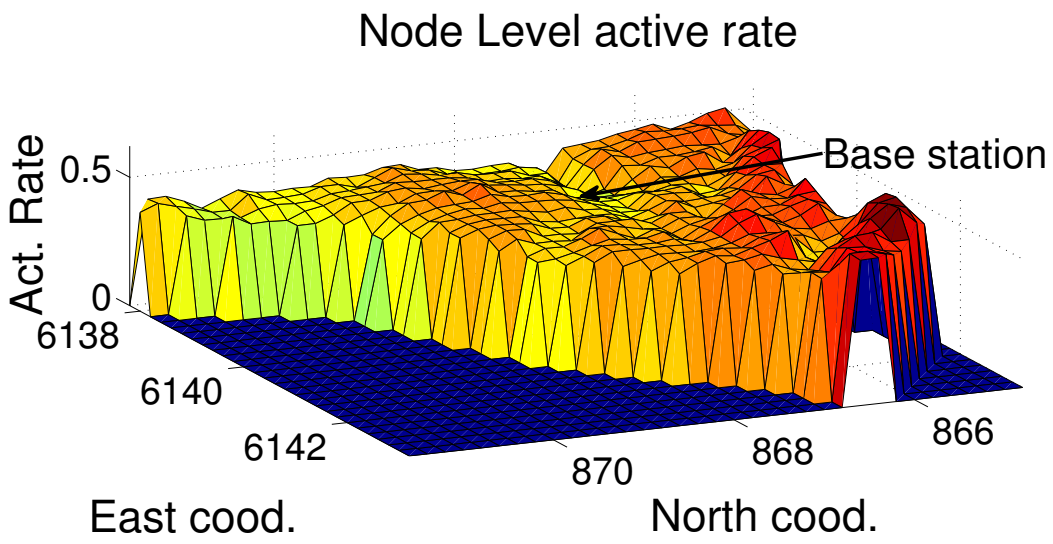


Figure 3.7: The node-level activity rate given by the statistical data.

four protocols. Geographic routing has high channel utilization around the base station, and has very low channel utilization at distant nodes. This increases collision rate near the base station, causing packets to be dropped. Ambient Energy Aware routing has less traffic concentrated at the base station (Figure 3.8), reducing the packet drop rate. Buffer size dependent routing and undirected routing have higher average channel utilizations compared to the previous two protocols, and their node-level channel utilization distributions follow the node activity rate (Figure 3.7). This is reasonable because nodes with higher activity rates are available more frequently and therefore may receive more packets. Whether or not this natural bias is helpful depends on the distribution of wind speed. If the wind speed is similar in most sensor locations, the natural bias will lead to spatially balanced channel use. Otherwise, channel traffic will concentrate on active nodes, and is likely to increase packet drop rates around those nodes. In this case, Ambient Energy Aware routing helps by distributing the channel capacity among nodes with higher probability in delivering packets to the base station.

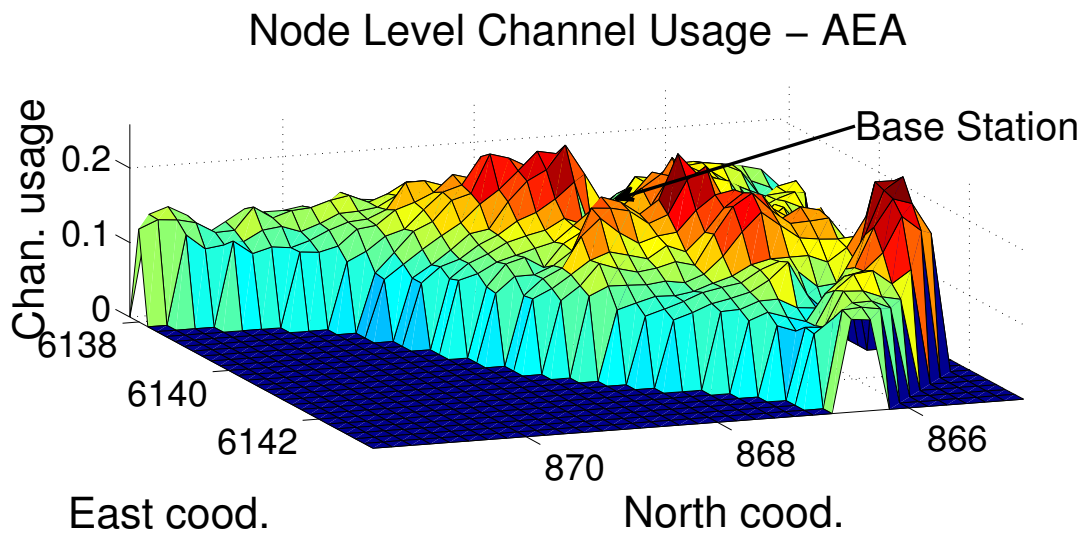


Figure 3.8: The node channel utilization for Ambient Energy Aware routing.

3.6.2 Per-Node Packet Delivery Rate

We now consider the *fairness* of the protocols under evaluation, i.e., the variation of packet delivery rates of all nodes in the network. We evaluate the fairness by plotting the location-dependent per-node packet delivery rate across the whole network.

Figure 3.9 shows the per-node packet delivery rate for a medium-scale sensor network. Ambient Energy Aware routing achieves good fairness among nodes, and results in higher average per-node packet delivery rate, while geographic, buffer size dependent, and undirected routing favor nodes that are closer to the base station. This result is consistent with the ranking function used in each protocol. Geographic routing relies on a ranking function that gives higher priority to nodes closer to the base station. Therefore, it is biased toward causing heavy communication on the nodes around the base station. Ambient Energy Aware routing considers the node distance to the base station as well as its activity rate, and therefore has less severe bias on nodes with different distances to the base station, resulting in better fairness.

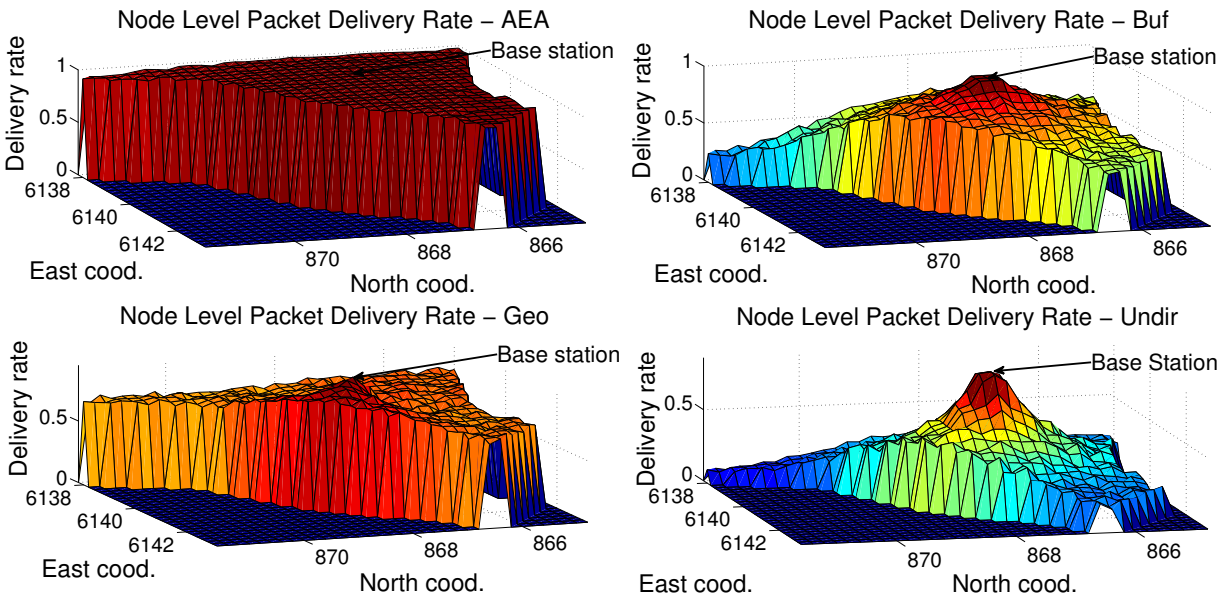


Figure 3.9: The node level packet delivery rates using four different routing protocols.

3.6.3 Protocol Selection Strategy

Using the results from this section and Section 3.5, we are able to provide protocol selection strategies for energy scavenging sensor network designers. We first give the conditions under which the four candidate protocols are appropriate. Undirected routing can only work efficiently in small-scale networks, and when ambient power is sufficient to power sensor nodes for most of the time. Buffer size dependent routing works best for small-to-medium size networks and medium transmission ranges. Geographic routing works best for small scale network and large transmission ranges. Ambient Energy Aware routing works well under most conditions, and can out-perform others in large-scale networks, even when ambient power is not sufficient to frequently wake up sensor nodes. It adapts better to extreme conditions than other protocols. Based on these working conditions, we provide the designer with several guidelines.

1. Determine required packet delivery rate and latency. The most appropriate protocol depends on these requirements.

2. Select values for sensitive parameters. The values for these parameters should be

determined first, since they are very likely to affect the result. These parameters include network scale and sensor node transmission range. The optimal protocol depends strongly on these parameters. On the one hand, Ambient Energy Aware routing outperforms others under "harsh" conditions: when network scale is large or sensor nodes have small transmission range. In addition, it performs well under looser requirements. Designers should choose this protocol when applications require a large number of sensor nodes, or when they are limited by sensor node communication hardware. On the other hand, in a small-scale network using nodes with long transmission ranges, users can instead use simpler protocols.

3. Select values for less sensitive parameters. These parameters either do not greatly affect packet transmission latency, or always result in the same optimal protocol when their values vary. These parameters include sample size and the maximum power consumption of the sensor node. Users are relatively free to select these parameter values; indeed, to improve the overall performance, we recommend selecting the most favorable parameter values for a given budget. For instance, using a larger energy scavenging device can provide more power to the sensor nodes under the same ambient energy conditions; this then allows users to use more powerful sensor nodes that boost the performance.

3.7 Discussion and Caveats

In this section, we address some of the simplifying assumptions made during our evaluation and discuss the impact that they might have on our reported results. We believe that our experiments capture the most important features of the environment and sensor network well enough to provide a reasonable evaluation of the routing protocols. However, our evaluation framework does not consider some secondary effects including long-term wind speed variation and adjusting node activity rates online. We now address these issues and

consider combining Ambient Energy Aware Design with more sophisticated protocols.

3.7.1 Long-term Wind Speed Variation

There can be long-term variations in wind speed distribution, on time scales ranging from three months to half a year [49] due to seasonal changes and long-term weather patterns. This means that the wind speed distribution will change multiple times during the lifetime of a long-term deployment. In our previous evaluation, we only use one fixed wind speed distribution at one location. This distribution is used in the ranking function. For a real system deployment, long-term wind speed variation should be considered.

The sensor network design can adapt to this variation by pre-storing multiple wind distributions. The variation of wind speed distribution in one location is periodic, repeating yearly. This periodic distribution is usually available from the wind atlas of local government websites [48, 59]. Therefore, even for long-term operations, only a limited number of wind speed distributions need to be stored in the nodes. We can then program nodes with multiple wind speed distributions and corresponding times at which the wind speed distribution changes. Sensor nodes will know to switch to a new activity rate by monitoring time stamps.

3.7.2 Online Adjustment Of Node Activity Rate

Sensor nodes can gather information on their activity rates after deployment. These activity rates are representative of the actual power source condition at the node's location. Therefore, using this value to adjust the pre-stored node activity rates makes them more accurate.

Sensor nodes use timers to gather activity rates. Due to power losses, a node's timer may not record the correct time. It can nonetheless record how long a sensor node has been active. Every time the node has an opportunity to synchronize its timer, it computes its activity rate by dividing the measured node active time by the total elapsed time since the last synchronization event and this value is used to update the pre-stored node activity rate.

3.7.3 Protocol Extensions

Although we mainly compare Ambient Energy Aware routing to geographic routing, other sophisticated geographic routing protocols, such as Greedy Perimeter Stateless Routing (GPSR) [47], also face similar problems when applied to energy scavenging sensor network. GPSR can prevent data from being stuck at the edges of holes in a sensor network. However, it still cannot avoid sending packets to an infrequently active node in an energy scavenging sensor network.

It would be possible to use the concepts described in Section 3.4 to design an ambient energy aware variation of GPSR by using a similar ranking function for each node. Such a protocol would avoid low activity rate nodes when searching for paths around holes.

3.8 Conclusion

We have described and evaluated a novel class of design techniques for indefinitely deployed sensor networks. To enable this increased lifespan, we proposed eliminating batteries from sensor nodes and introducing a new routing protocol that account for the resulting restrictions in node activity. This protocol can achieve high delivery rate, even when sensor nodes randomly lose their power sources. It uses stochastic models for ambient power sources and takes advantage of spatial and temporal correlation to make routing decisions. We evaluated the newly proposed protocol together with three existing approaches for four commonly used applications. Finally, we provided guidance on selection of routing protocols for specific applications.

CHAPTER 4

A Scheduler For Performance and Energy Optimization in Data Centers with Heterogeneous Tasks or Machines

4.1 Introduction

Both performance and energy consumption of data centers depend on the loading conditions of individual machines. Performance is best when resources don't needlessly sit idle and tasks are not hampered by resource contention; energy consumption is minimized when idle machines use little power. However, in real data centers, resources are often left idle but still consuming power, while others are over-loaded, even when the aggregate workload for the data center would permit more efficient use of resources, given the right task scheduling and assignment policies.

Suboptimal task assignment and scheduling causes resource contention on data center machines, reducing performance. Existing task schedulers achieve acceptable performance in data centers with moderate workloads that are mostly homogeneous, but suffer performance degradation when workloads vary and are heavy. They have three main drawbacks.

1. They do not consider machine loading during task assignment. The Hadoop default scheduler uses a constant constraint on the number of concurrent tasks. This number

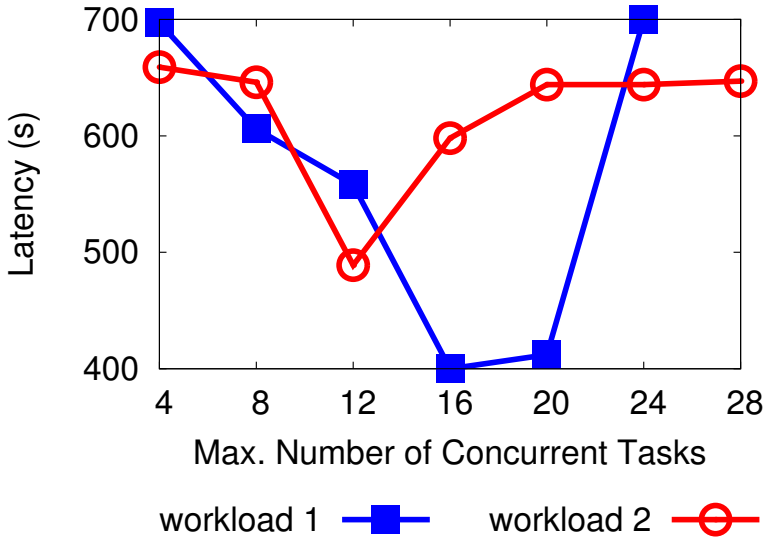


Figure 4.1: Dependence of latency on the maximum concurrent tasks constraint. The latency numbers are measured on one machine running two example workloads with different task compositions. The optimal latency value depends on the particular workload.

is generally suboptimal for task latency, as demonstrated in Figure 4.1, because the scheduler does not adapt to changes in loading conditions: machines are generally either overloaded or under-loaded. This increases average task latency.

2. They ignore resource sharing among concurrent tasks. Some existing works model concurrent loading effects by summing the utilizations of individual tasks running separately, instead of considering resource sharing, which increases resource contention and task latency [60–62].

3. They overlook task and machine heterogeneity, resulting in resource contention and therefore idle resources, increasing job finish time [63]. For example, consider a 6-node cluster running a heterogeneous workload. Figure 4.2 demonstrates the imbalanced resource utilization of each machine, which leads to a $6\times$ difference in workload finish times across different machines.

We have developed HAMS (*Heterogeneous Adaptive Modeling Scheduler*), a task scheduler that improves data center performance by addressing the above three problems.

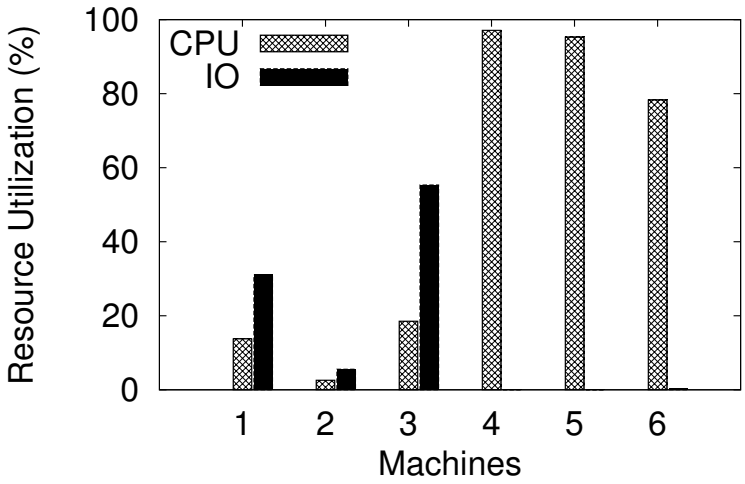


Figure 4.2: CPU and IO loading on a cluster of six machines running a group of workload. The workload consists of 40 CPU intensive tasks and 40 IO intensive tasks. The cluster consists of two types of computers: machine 1–3 and machine 4–6. The IO loadings for machine 4–6 are approximately zero.

HAMS initiates self-characterization of tasks on each type of machines in data centers, building two task-and-machine heterogeneity-aware models: a predictive task performance model and a background-loading-aware machine model. HAMS also monitors machine background loading. Using this loading information and the above two models, HAMS is predicts task execution latencies under many potential post-assignment workloads supporting scheduling decisions that optimize resource utilization and minimize job finish times.

HAMS can also be used to save data center energy when combined with a task concentration algorithm. Commercial data centers spend a lot of energy keeping idle or underloaded machines on. Diurnal and longer-term changes in workload permit substantial benefit from concentrating tasks and powering down idle machines when data center utilization is low. Such periods generally last several hours and the loading commonly varies from $<$10% to 80% in a single day [64, 65]. Idle machines consume 40% the power of loaded machines [2, 66, 67]. As a result, data centers with many idle machines waste a lot of energy. Energy efficiency can be improved by concentrating tasks on a subset of machines and shutting down the rest [4, 60, 68]. However, existing task concentration techniques

increase task latency greatly, which can result in a 3.6× increase in job finish time [4]. This limits their use, and increases energy consumption by causing workloads to take longer than necessary. In contrast, HAMS concentrates tasks to machines with loading conditions that minimize resource contentions, thereby maintaining task performance, and powers off underutilized machines to save energy consumption.

We have implemented our modeling and task scheduling techniques for the MapReduce programming model, which is used for processing large datasets. MapReduce is widely used in distributed systems and cloud computing. We chose it for evaluation because this paradigm is common in modern data centers. However, the modeling and scheduling ideas described in this chapter can be extended to other computing paradigms. We base HAMS on the Hadoop framework, an open-source platform that is widely used in industry and academia.

4.1.1 Contributions

This chapter presents a novel task scheduler that optimizes data center performance and is suitable for task concentration. It makes three contributions.

- We describe a data center modeling, assignment, and scheduling technique that adapts to task and machine heterogeneity. Our task modeling technique specifically addresses task resource contention.

- We describe how to predict the impact of data center assignment decisions on already-running tasks, making task migration (with its associated overhead) unnecessary.

- We provide an approach that reduces data center energy consumption via task concentration with little impact on latency. This approach does not require frequent activation or deactivation of machines.

4.2 Problem Definition

We aim to improve data center performance through intelligent task scheduling. The task scheduler should adapt to machine loading conditions and impose little overhead. We face two major challenges. First, the number of possible multi-task assignments grows exponentially with the number of pending jobs. Second, to avoid task migration overhead, task scheduling decisions are made before actually executing the tasks. This requires accurate prediction of the performance implication of task assignment decisions before they are made, which requires sophisticated performance models that capture task and resource heterogeneity. This section formalizes these problems and describes the insights that lead us to solutions.

4.2.1 Performance Metrics

We use an assignment-dependent performance predictor to evaluate candidate task assignments. Each data center machine contains multiple CPUs, a shared memory system, and a dedicated hard drive. This configuration is used in research and commercial data centers, e.g., Amazon EC2. Tasks running on the same machine share its resources, and do not interfere with tasks running on other machines (network effects are discussed in Subsection 4.3.3). To estimate the rate of progress for tasks assigned to particular resources, it is necessary to consider the total load imposed on the resources.

We first define some terms that we use when describing machine throughput and job deadline prediction.

- A *task* is the unit of MapReduce execution. One task processes a known amount of input data.
- A *job* is a combination of several Map or Reduce tasks.
- *Work* is the time spent by a task to process a fixed amount of data. It can be described

as a function of the input data size and the CPU time for task execution (exe_time) on an unloaded machine. This is a per-task characteristic. The work for task i is $work_i$.

- A *resource utilization vector* ($U(CPU, memory, disk)$) consists of the utilization percentages for several resources that influence task performance or machine power consumption. CPU, memory, and disk utilization are the elements of the resource utilization vector.

- *Performance degradation ratio* is the ratio of the execution time of a task running on a loaded machine to the execution time of the same task running on an unloaded machine, i.e., for task i

$$ratio_i = \frac{exe\_time_{i,\,loaded}(U)}{exe\_time_{i,\,unloaded}}. \tag{4.1}$$

- The *finish time* ($T_{finish,\,j}$) of a job is the time when the jth job finishes execution.

We define our goal and the corresponding cost function as follows.

Aggregate throughput: Our goal is to optimize the aggregate throughput of all tasks in the entire data center, i.e.,

$$Throughput = \sum_{i=1}^{all\_tasks} ratio_i \times work_i. \tag{4.2}$$

In order to maximize aggregate throughput, we optimize the execution time and machine assignments of tasks.

Fairness: We use the following function of the amount of time that a job takes to finish after its deadline to describe the cost of deadline violation of all N jobs:

$$Unfairness = \sum_{j=1}^{N} \left(\frac{\max((T_{finish,\,j} - deadline_j),\, 0\,\mathrm{s})}{deadline_j} \right)^{\gamma}. \tag{4.3}$$

γ will generally be ≥ 1. We try to meet all job deadlines. If a deadline cannot be met due to

resource constraints, we minimize the summed violations of job deadlines.

Formal problem definition: *Given the resource utilization vectors of machines in a cluster and resource utilization functions for tasks, determine which tasks should be assigned to each machine to achieve the best throughput and meet job deadlines.* We will later extend this definition to optimize energy consumption. The subsequent sections describe these models and our task assignment algorithm.

4.2.2 Energy Optimization

Our task scheduling algorithms can be extended to optimize energy consumption as well as performance. Tasks can be concentrated on a subset of machines, reducing data center power consumption. When a group of jobs runs on a cluster with multiple machines, the scheduler detects the resource utilization of each machine, and shuts off those on which all resources are under-utilized for a certain period of time, e.g., 80 seconds, a common task execution time [69]. Machines are turned back on when the average job latency increases beyond a user-specified threshold percentage. Therefore, the energy saved by concentration follows:

$$\begin{aligned} E_{saved} &= E_{normal} - E_{con} \\ &= P_{normal} \cdot L_{normal} - P_{con} \cdot L_{con}, \end{aligned} \tag{4.4}$$

in which E_{normal} and E_{con} are the total energy consumptions before and after task concentration, and P and L are the total power consumption and task latency. The latency of turning on/off machines is usually tens of seconds [4], which is ignorable because the number of powered down machines need change only every few hours.

The above approach, however, can increase energy consumption if the task execution latency (L_{con}) increases. After task concentration, the workloads on active machines increase, increasing the probability of resource contention, leading to performance degradation, there-

fore reducing possible energy savings. We consider resource sharing and task heterogeneity to optimize energy consumption, and explicitly model the impact of task execution time on energy. This approach makes use of the same task scheduling infrastructure used for performance optimization.

4.3 Design and Implementation

This section describes the design and implementation of HAMS. We consider the effect of resource contention on CPU, memory, and disks shared by concurrently running tasks. Our scheduler consists of two models and a predictor. It estimates future task resource utilization vectors based on the resource utilization vectors of already-running tasks, and predicts the resulting execution times of candidate and already-running tasks. These models allow prediction of the aggregated throughput of each machine in the cluster, which is used to guide task assignment and scheduling.

HAMS has two main components that are used in an iterative prediction process.

1. Task models are used to estimate task properties. They take background resource utilization as inputs. We build the resource utilization and execution time models via pre-characterization. Each type of task is executed with resource loading conditions spanning the range encountered during normal execution.

2. Machine throughput prediction estimates machine properties using information of its loaded tasks. Before assigning candidate tasks, the scheduler uses task models to predict the future throughput of a machine, and makes the final assignment decision by selecting the task combination with maximal throughput.

Figure 4.3 illustrates the design of HAMS. Incoming tasks $t_1, t_2, ..., t_n$ are first assigned to the cluster for pre-characterization. Each type of task is characterized on different types of machines $M_1, M_2, ..., M_m$ to build their machine type dependent models. The task scheduler

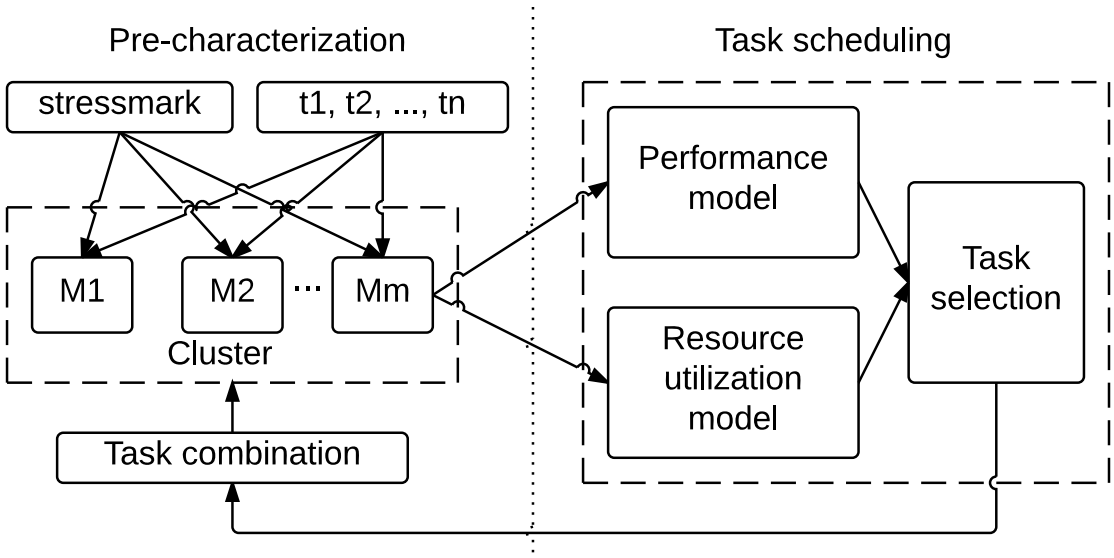


Figure 4.3: System architecture of HAMS.

then starts using task performance models to optimize task combinations. The rest of this section explains this process in detail.

4.3.1 Task Pre-Characterization and Performance Modeling

We use a resource utilization dependent task performance model to predict task execution times and machine throughputs. The model is built during the task characterization process, which explicitly considers task heterogeneity and resource contention.

4.3.1.1 Model Description

An accurate model for task execution time is central to making high-quality assignment decisions. This task model must capture the effects of task heterogeneity. Some previous work [69, 70] uses task execution time alone to characterize tasks. There are two main drawbacks to using task execution time. First, this time does not capture differences in resource utilization among tasks. Tasks with the same execution time can have very different resource utilization patterns. Second, this time is usually measured on unloaded machines, and does not change linearly with the machine loading.

We aim to provide task models that capture the heterogeneity of individual tasks, and capture the influence of resource contention with other tasks on performance. Task heterogeneity can be represented by a set of resource utilization vectors and their corresponding execution times. Task performance changes imposed by contention with other tasks can be captured by identifying how a task's resource utilization vector is affected by concurrently running tasks. More precisely, we determine the impact of heterogeneous machine background loading on each task type, and also determine how each task influences the total loading of the machine. We first describe a model based on per-task resource utilization vectors. We then describe a model that estimates task execution times based on these vectors.

Our task resource utilization and performance model consists of two functions.

1. A model of task resource utilization vector (CPU, I/O, memory) as a function of background loading.

2. A model of task execution time as a function of its resource utilization vector.

We express these two models as functions of resource utilization vectors for the candidate task t_i and the machine:

$$U_i = R(U_{background}) \textit{ and} \tag{4.5}$$

$$D_i = P(U_i), \tag{4.6}$$

in which functions $R(U)$ and $P(U)$ represent the relationship between the resource utilization vector and performance, $U_{background}$ is the resource utilization vector of the machine background loadings (i.e., the CPU, memory, and disk vector resulting from all existing tasks running on the associated machine), U_i is the predicted resource utilization vector of the candidate task t_i, and D_i is the predicted execution time of that task.

Model functions R and P come from interpolation of the relationships among task performance, resource utilization, and background loadings. Figure 4.4 shows an example of model function, P. The task resource utilization relationship cannot be captured by a simple,

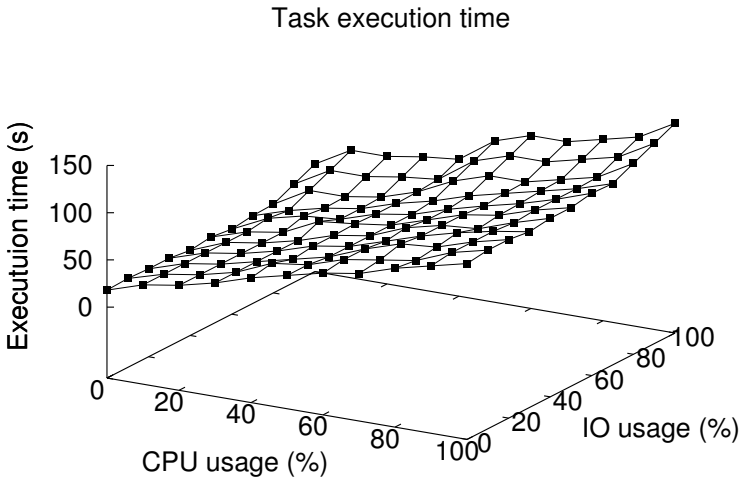


Figure 4.4: Performance model for one Map task. Task latency is dependent on resource utilizations.

linear model; using a non-linear multi-dimensional surface is necessary. When using the model to compute task resource utilization under arbitrary loadings, we use multi-dimension interpolation from the model matrix. The surface can be sampled with sufficient density to keep interpolation error low.

Cache effect is omitted in the resource utilization vector presented above because MapReduce tasks are typically not very sensitive to changes in cache miss rate resulting from inter-task cache contention. We used memory-intensive background loading tasks to vary the degree of cache contention experienced by each type of task in our benchmarks, thereby varying the cache miss rates measured by processor performance counters from nearly 0% to 60%. The resulting change in task execution time was only 3%. Note that although it is unnecessary to consider the impact of cache contention on task execution time, our measurements indicate that memory contention has a significant impact on task execution time, and is therefore explicitly modeled.

4.3.1.2 Model Building

We determine the resource utilization and execution time functions for each task by characterizing it on a machine with a controlled range of loadings. During characterization, tasks are run in the presence of concurrent threads that apply varying and controlled loads to CPU, memory, and disk. For each loading condition, the resulting execution times and resource utilization vectors are noted. We sweep the background loading of each resource from unloaded to nearly fully loaded, and iteratively increase the number of samples for each resource until adding more points does not significant change the surfaces (per-task resource utilization and task execution time).

In real data centers, some types of tasks are typically executed many times. It is possible to avoid pre-characterization of a particular type of task until it has been encountered numerous times, or to gradually increase modeling resolution as execution count increases. This amortizes characterization overhead. The time spent on pre-characterizing one task should be small compared to the total runtime of it being repeatedly run on a data center. Therefore, we recommend only characterizing tasks that contribute to jobs that lasts over 10 minutes and repeats over 300 times, i.e., the tasks that account for more than 10% of the total data center workload [70]. We list the pre-characterization overhead of an example group of such tasks in Table 4.1. The average overhead is 2.5% of the total workload runtime for pre-characterizing 50 and 100 samples across different types of tasks. This overhead is taken into account when discussing our performance improvement result.

4.3.2 Throughput Prediction

Task performance models are used to calculate the throughputs of machines running particular sets of tasks. This allows the scheduler to determine whether additional tasks should be assigned and, if so, which machines they should be assigned to. By using the models to estimate the impact of different tentative assignment decisions, it is possible to optimize performance and energy consumption.

Table 4.1: Task Pre-Characterization Overhead

task	avg. runtime (s)	50 samples	100 samples
JavaSort1	48.0	1.8%	2.7%
WordCount0	43.0	1.2%	2.4%
WordCount1	80.0	2.5%	4.4%
HtmlIndexing	65.0	2.0%	3.6%

Predicting the implications of a particular assignment decision is challenging, because it is not practical to solve for task resource utilization and execution time directly from Equations 4.5 and 4.6. The non-linearity of these functions prevents derivation of a closed-form analytical solution. Therefore, we use the two-stage iterative process shown in Figure 4.5 to solve for per-task resource utilizations and execution times.

The scheduler predicts the resource utilization of the candidate task using Equation 4.5. The input to that equation, background loading, only accounts for already-assigned tasks on the target machine, without considering the new candidate. Therefore, this value is inaccurate; however, it serves as a good starting point for iterative estimation. Next, this estimated resource utilization vector is used to recompute the resource utilization vectors of already-running tasks using Equation 3. We cannot simply add up and renormalize the resource utilization vectors of each individual task for this value, since tasks can share resources. However, we can start by assuming a linear relationship between individual task resource utilization vector U_i and the aggregate background loadings and iterate until convergence using the following equation:

$$U_{background} = L(U_1, U_2, ..., U_M) = \sum_{i=1}^{M} U_i \cdot \alpha_i, \tag{4.7}$$

in which M is the number of concurrent tasks on the same machine, $L(U_1, U_2, ..., U_M)$ describes the linear relationship, and α_i's are constants.

We use the predicted resource utilization vector value for each task, including the already-running tasks and the candidate task, in the execution time model to compute the predicted task execution times of all concurrent tasks. This value is later used in the throughput calculation in Equation 4.2.

After carrying out these steps for every candidate task, the scheduler selects the task–machine pair resulting in the highest predicted throughput. If the maximum predicted throughput is lower than the current throughput, the scheduler pauses until a task completes execution before reconsidering the assignment of additional tasks. *This policy is important; it acts to optimize flow control for maximal throughput.*

During evaluation, we built task models on physical machines. However, the same method can be applied to virtual machines, if the scheduler is made aware of resource sharing. Resource utilization will be changed if the same resource is shared among two different virtual machines. It is possible to determine which physical node is hosting a virtual machine, enabling measurement of shared resource utilization by different virtual machines. During task pre-characterization and task execution, the resource utilization of a task and background loadings can be measured from virtual machines that share the same resource as the candidate tasks. Therefore, the scheduler is able to obtain sufficient information to perform task modeling and scheduling.

4.3.3 Model Accuracy

We now discuss the two main limitations on the accuracy of our task performance models.

1. The resource utilization vector of already-running tasks may change during task execution due to the completion, assignment, and dynamic changes of resource utilization by other tasks. However, prediction is done before assigning a task. It would be possible to compensate for such changes by periodically re-estimating task performance and migrating tasks to other resources. Unfortunately, this solution is complicated and increases estimation overhead. Fortunately, there is another way to compensate for

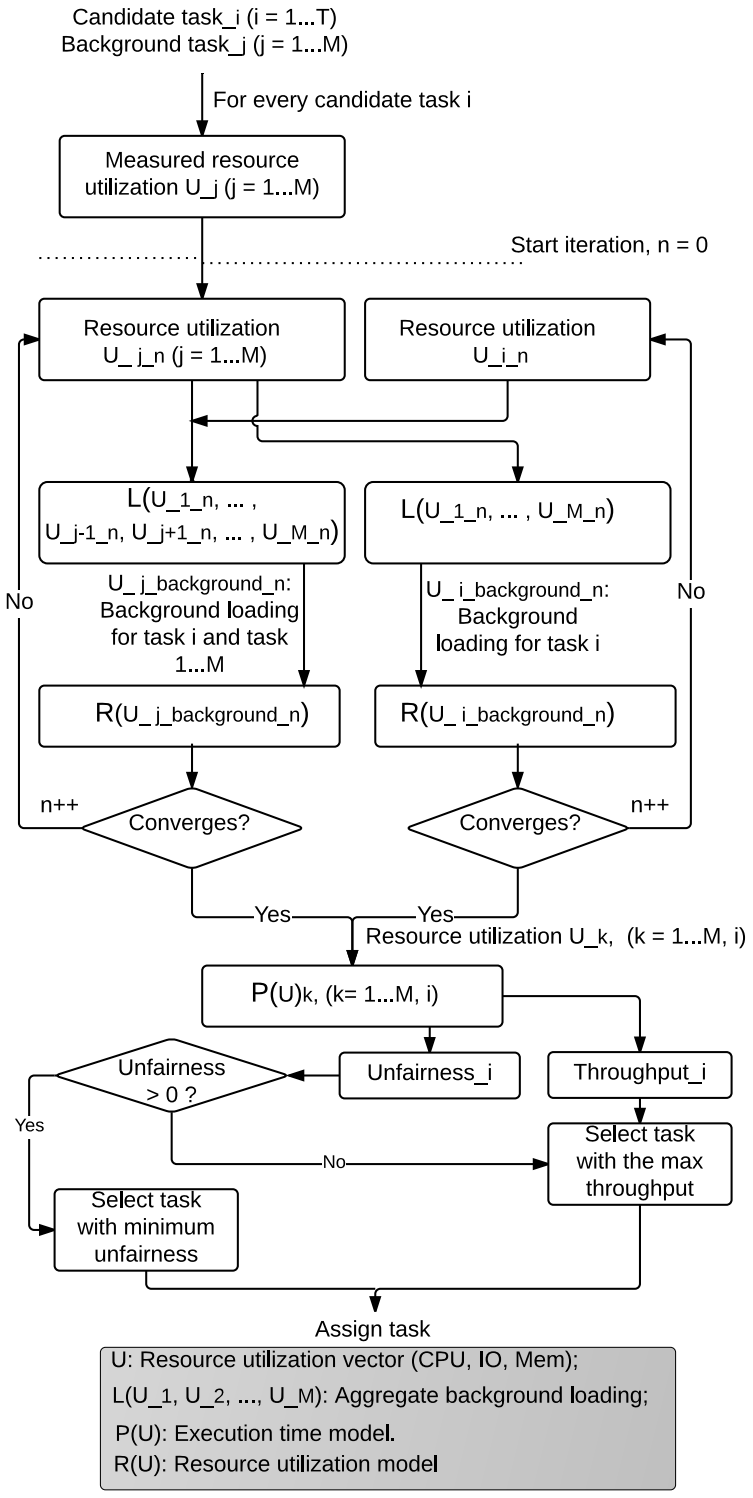


Figure 4.5: Task throughput prediction procedure.

post-assignment changes in resource utilization. Task assignment decisions occurring after a resource utilization change consider the impact of that change. They also consider the impact of the new assignment decision on the completion times of all currently running tasks. This form of adaptation achieves most of the benefit of periodic task migration without its complexity as long as each machine concurrently runs more than a couple of tasks, which is generally the case in data centers.

2. We recommend neglecting some secondary factors that influence prediction accuracy by only 2%, e.g., impacts of network and disk seek latencies. First, modern data centers use network switches of sufficient speed, e.g., 40 G links [71], that network transfer delay can be neglected. Furthermore, network transfer for large datasets is extremely rare because data locality is commonly maintained, as described in the following subsection. Second, modern hard drive seek times are small enough to safely neglect the variation in disk access latency. We repeatedly ran tasks with fixed resource utilization and found that the parameters neglected by the model only introduced 2% error in performance prediction.

4.3.4 Impact on Data Locality

Our task assignment and concentration techniques have little impact on data locality. This is because Hadoop distributes multiple (by default 3) replicas of data for a task across different levels of network hierarchy, allowing most tasks to access their data on local or adjacent nodes with low network latency. If data center designers choose to use a network architecture in which network delays are large and highly variable, it would be necessary to extend the model in HAMS to explicitly consider data locality. We discuss this impact of data locality on HAMS in Section 4.6.

4.3.5 Other Design Considerations

The use of model-based task assignment naturally entails additional changes relative to the Hadoop default scheduler. The following changes result in additional performance improvement and fairness preservation.

HAMS makes task assignment decisions dynamically when new tasks become available for execution and when tasks finish execution. The original Hadoop scheduler is invoked periodically at a fixed interval, which produces unnecessary overhead (if too frequent) or suboptimal results (if too infrequent). Furthermore, the Hadoop original scheduler waits until all tasks in a pending group finish before admitting new tasks, wasting resources near the end of a group's execution. HAMS assigns tasks until there is no throughput improvement; it then stop assigning new tasks until there is status change, e.g., task completion or new job arrival. As a result, HAMS rapidly responds to changes in loading and avoids leaving resources idle.

HAMS adaptively sets the number of concurrently running tasks on a machine. Existing Hadoop schedulers use a fixed, user-specified limit on the maximum number of concurrent tasks. This limit is a coarse way of preventing detrimental resource contention, but it ignores task heterogeneity. HAMS stops assigning tasks to a machine when its resource utilization vector becomes so unbalanced that assigning additional tasks would reduce aggregate throughput. In addition, HAMS stops assigning additional tasks when a machine's physical memory is 70% full to avoid swapping to disk. Swapping could have been handled implicitly by task performance models. However, it is so destructive that explicitly preventing it results in correct behavior and enables simpler task performance models.

HAMS also preserves fairness among jobs by respecting deadlines, which may be specified by users or automatically assigned. HAMS minimizes the aggregate deadline violation penalty for assignment decisions (Equation 4.3). Candidate task assignment decisions that will not result in deadline violations are prioritized over those resulting in deadline violations. Using the result from task performance prediction, we can estimate job finish time by

assuming that job progress rate will remain the same in the near future. This progress rate is calculated using the recent task progress during a short time period, and is updated every time the loading condition of a machine changes, to increase estimation accuracy. Although HAMS considers fairness, space constraints force us to focus on performance and energy comparison in our experimental evaluation (see Section Section 4.5).

4.4 Power Model

This section describes the power model HAMS uses for energy optimization. Server machine power consumption depends on resource utilization. We do not claim that this model is novel; it is instead a means of evaluating the impact of our contribution (heterogeneous task modeling and scheduling) on energy consumption.

Idle power consumption can contribute up to 40% of the total machine power consumption, especially in machines with many hard drives and a lot of memory. CPU and memory power management only save a limited amount of machine power. Ongoing work on power proportional computing is attempting to remedy this situation [4,60,64], but it is unclear that idle power consumption will be reduced to nearly zero in the foreseeable future. Turning off or sleeping an entire machine reduces its power consumption to nearly zero, but imposes a large time penalty when it is again required. We evaluate using HAMS to reduce the performance penalty of concentrating tasks onto a limited number of machines, making it safe to power off some machines.

We now describe the power model used for evaluation. It has been found [66, 67, 72] that the active power of a machine mainly depends on variation in CPU utilization, and that variation in memory and disk utilization has limited effect (approximately 4%) on the total power consumption. Therefore, we omitted the power effects of varying the use of these

two resources. Machine power consumption is modeled as follows:

$$P_{total} = P_{idle} + P_{active} \cdot CPU\%. \tag{4.8}$$

We break the machine total power consumption into idle power and active power, which is linear in the CPU utilization.

We compare the energy consumptions of the normal operating mode and task concentration using the following power model. Under normal operation, all m machines in a cluster are active. During task concentration, only a subset (numbering r) remain active. Let L_{norm} and L_{con} be the total execution times during normal operating mode and after task concentration. The relative increase in aggregate workload latency as a result of task concentration can be described using a factor $c = L_{con}/L_{norm}$. Thus, the energy saved by task concentration follows:

$$\begin{aligned} E_{saved} &= P_{norm} \cdot L_{norm} - P_{con} \cdot L_{con} \\ &= L_{norm}[P_{idle}(m - c \cdot r) + \\ &\quad P_{active}(m \cdot CPU_{norm}\% - c \cdot r \cdot CPU_{con}\%)]. \end{aligned} \tag{4.9}$$

The above energy savings depend on the relationship between P_{idle} and P_{active}, and also the difference in performance and the number of machines that are still active after task concentration. If (1) idle power is significant, (2) the scheduler permits task concentration with little performance penalty, and (3) the number of active machines is significantly reduced, then significant energy savings are possible. The first condition is determined by the properties of the machines in the data center. The second and third conditions depend on the task scheduler. We will later show that HAMS is able to reduce r and L_{con} relative to other Hadoop schedulers resulting in significant energy savings.

Power consumption data are available from server datasheets. We gather power consumption values for two types of machines: HP and Dell servers (two Intel Xeon E5540

Table 4.2: Server Power Consumption Breakdown

Machine	P_{max} (W)	P_{idle} (W)	P_{total} (W)
Dell M610	255	116	$(0.44 + 0.56 \times CPU\%) \times 255$
HP BL640c	257	118	$(0.46 + 0.54 \times CPU\%) \times 257$

processors, 6 RAMs, and 2 HDDs), which are similar to those used in our evaluation. The machine idle power (P_{idle}) and maximum power (P_{max}, when the CPU is fully utilized) are shown in Table 4.2 [73]. The last column in this table demonstrates the linear power model for these two servers. These two models have similar parameters. We will later use these models for evaluating the impact of scheduling policy on energy consumption.

4.5 Evaluation of the Task Scheduler

This section presents our experimental evaluation of HAMS. Our task scheduling policy is based on the facts that (1) balanced resource utilization leads to higher throughput and (2) for most tasks, increasing the per task resource utilization improves performance. Both machine and task resource utilization are affected by the resource sharing conditions implied by scheduling decisions. We report the impact of scheduling policy on data centers running Hadoop for several benchmarks containing varying mixes of CPU-, memory-, and disk-intensive tasks. HAMS outperforms existing scheduling policies for both homogeneous and heterogeneous task combinations. We also report the impact of using HAMS to optimize energy consumption.

4.5.1 Experimental Setup

We perform our evaluation on Emulab clusters [74] using two types of machines. Type 1 has two 64-bit Intel Quad Core Xeon E5530 processors, 8 GB memory, and two 250 GB, 7200 RPM hard drives. Type 2 has a 3.0 GHz 64-bit dual-core Xeon processor, 2 GB memory, and two 146 GB 10,000 RPM hard drives. These machines use tree-structured network connections; the Hadoop name node is at the root and data nodes are at the leafs. We use two clusters: a *single-node cluster* containing only one data node on a type 1 machine, and a *30-node cluster* with 15 type 1 machines and 15 type 2 machines. While running the workloads, we measure the resource utilization of each machine using Linux /proc pseudo-files.

We used two sets of experiments to determine the aggregate throughput of a cluster when using HAMS and existing scheduling policies.

1. *Evaluating task modeling and throughput prediction on the single-node cluster.* This is included primarily to simplify displaying and explaining machine resource utilization and concurrent running tasks information. In a cluster, our task assignment policy assigns every task to the machine that results in the highest data center throughput. The assignment process can be divided into two steps: (1) estimate the throughput of every machine for the candidate task considering the impact on other tasks and (2) assign to the machine resulting in the maximum overall throughput. The second step is straightforward; therefore, our evaluation focuses on the first step, which can be reduced to evaluating the scheduling policies on a loaded machine after scaling down the workload from one appropriate for a many-machine cluster to one appropriate for a single machine. We use the default data replication factor (level 3) provided by Hadoop. Therefore, little network traffic occurs in multi-machine clusters, reducing the overhead of the second step. As a result, it is possible to get a lot of information about the performance of different scheduling policies with this set of experiments.

2. *Evaluating scalability and energy savings on the 30-node, 90-core cluster.* These experiments are necessary to determine how the scheduling policies perform on a cluster. We start from the benchmarks used in the first set of experiments but increase the number of jobs in proportion to the number of machines.

4.5.2 Benchmarks

To evaluate the impact of loading conditions on scheduler performance, benchmarks should cover a range loading conditions. Our benchmarks exercise three different resources (CPU, disk, and memory) with different intensities by mixing tasks that heavily use particular resources. They are based on three types of tasks: sorting, word counting, and HTML indexing. We pick these tasks because they are commonly used in data centers and in evaluating related modeling and optimization techniques [69, 70]. We prepared several variants within each task type to produce different resource utilization levels, as described in Table 4.3. The table also demonstrates nine benchmarks with varying task mixtures. BM1–5 are heavily-loaded, while BM6–9 are lightly loaded.

The above workloads can be divided into two categories: (1) *naturally balanced workloads*, which contain tasks that are nearly balanced in their resource utilization and (2) *unbalanced workloads*, which contain tasks that heavily use one of the resources (CPU, disk, or memory), resulting in unbalanced resource utilization when running one task alone, but have a potential for balanced resource utilization when multiple tasks are run concurrently, if appropriate scheduling decisions are made. A well-provisioned data center will have an aggregate resource balance appropriate to the tasks it will run. Otherwise, the cost of the data center could be reduced with little or no performance penalty by decreasing the over-provisioned resource(s).

Table 4.3: Description of Tasks and Benchmarks

Task Description		Task Composition for Benchmarks								
		BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8	BM9
JavaSort1	Heavy util. on disk and memory	80	40	80	40	40	10	10	20	0
WordCount0	Moderate utli. on CPU and disk	0	2	0	40	40	0	0	0	80
WordCount1	Heavy util. on CPU, moderate util. on disk	0	0	0	0	0	0	1	0	0
WordCount2	Very heavy util. on CPU	0	0	40	0	10	1	0	0	0
HtmlIndexing	Heavy util. on disk, moderate util. on CPU	0	4	0	4	4	0	0	4	0
Balanced workload?										✓
Heterogeneous workload?			✓	✓	✓	✓	✓	✓	✓	

4.5.3 Performance Evaluation For All Benchmarks

We compare the scheduling policy used by HAMS with three other policies from Hadoop and other research projects. The original Hadoop scheduler is called *ori*; a scheduler that treats tasks as homogeneous [63] is called *simple*; and a scheduler that models heterogeneous tasks by considering execution time when run on an otherwise-unloaded machine [69, 70] is called *hetero*.

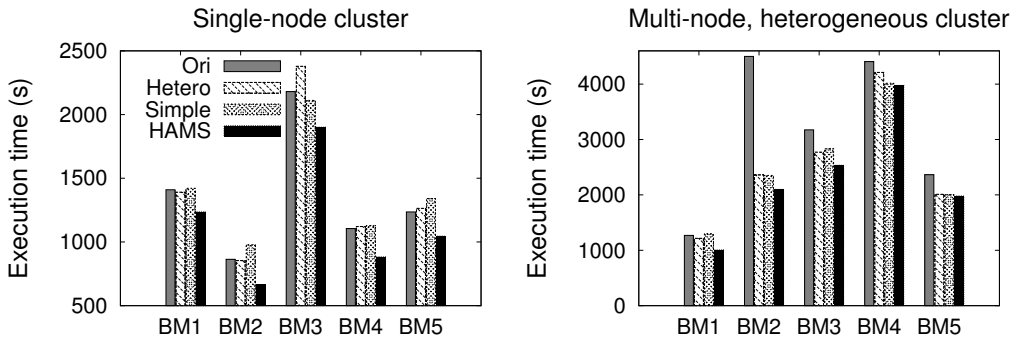
We perform two sets of experiments by running the four schedulers on the single-node cluster and the multi-node cluster. Improvements already account for task pre-characterization overhead.

- Figure 4.6a compares the results produced by the four scheduling policies on the single-node cluster. HAMS improves finish time by 13% compared to *ori* and by 11% compared to the other two schedulers. The improvement ranges from none to 24%, depending on the types of tasks in the benchmark.

- Figure 4.6b shows the performance comparison of four schedulers on the 30-node cluster. HAMS has an average performance improvement of 15% over *ori* and 10% over the other two schedulers. The performance improvements for unbalanced, heterogeneous benchmarks (BM2–BM5) are larger than for the single-node case.

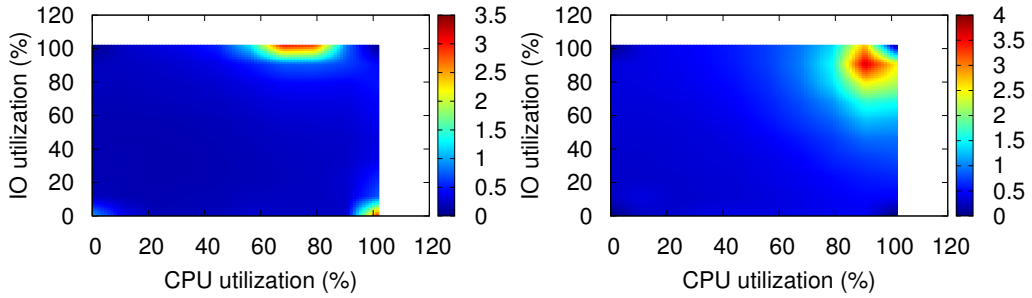
4.5.4 Balancing Machine Resource Utilization

The reduction in benchmark execution time results from improvement in machine resource utilization by better balancing utilization. HAMS has multiple features that might have brought this improvement. We now evaluate the impact of each. The *simple* scheduling policy ignores task heterogeneity and resource contention and the *hetero* policy considers task heterogeneity but ignores its change due to resource contention. As a result, they make unwise assignments by either issuing sub-optimal task combinations or by overloading machines. HAMS avoids these problems.



(a) Single-node cluster (b) Multi-mode, heterogeneous cluster

Figure 4.6: Workload execution times for HAMS and existing Hadoop schedulers on two clusters.



(a) *ori* (b) HAMS

Figure 4.7: Distribution of resource utilization vectors when running one benchmark using *ori* and HAMS. We omitted the results for the other schedulers because *simple* is similar to *ori* and *hetero* is similar to HAMS for this example.

Avoiding sub-optimal task combinations helps for unbalanced, heterogeneous tasks. Figure 4.7 shows the distribution of resource utilization vectors during execution of an unbalanced workload. We omit memory loading for figure, because it doesn't vary much. Both *ori* and *simple* produce unbalanced resource use. HAMS and *hetero* balance resource use.

Preventing over-utilization or under-utilization of machines helps when the workload over-utilizes some resources because a fixed number of concurrent tasks are run on each machine, as is the case for *ori*. This strategy improves throughput for both heterogeneous

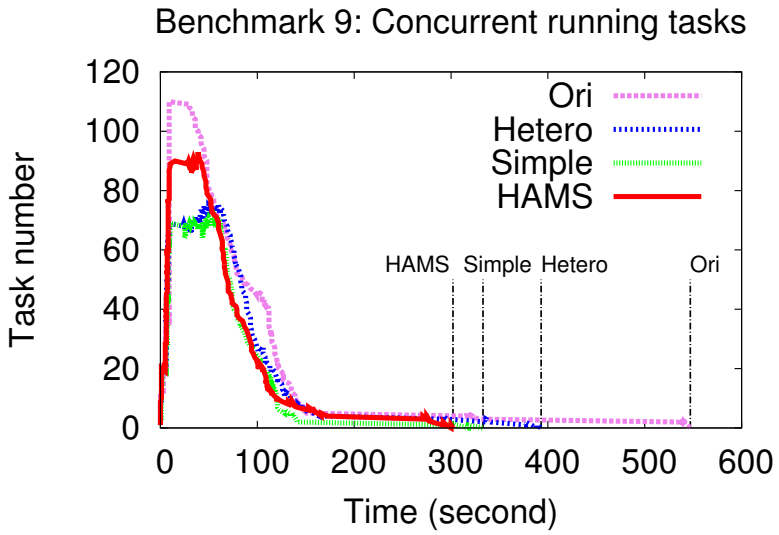


Figure 4.8: The number of concurrently running tasks on the cluster. The labels mark the job finish times for different schedulers.

and homogeneous workloads. Even when assigning a homogeneous tasks, *ori*, which always assigns a fixed number of tasks, may generate sub-optimal loading. When assigning a heterogeneous workload, task schedulers that are not aware of task heterogeneity (*simple*), or use a task model that neglects task interference (*hetero*), also result in inappropriate concurrency. Figure 4.8 shows an example of this. *Ori* loads the cluster with too many tasks, while *simple* and *hetero* load the cluster with too few.

To summarize, HAMS achieves performance improvement for both heterogeneous and homogeneous workloads. The biggest benefits occur for two types of workloads. (1) Unbalanced, heterogeneous workloads (BM2–BM5). This is due to the more comprehensive task and resource utilization modeling in HAMS. These workloads also benefit more from HAMS when run on heterogeneous clusters than when run on homogeneous ones. (2) Workloads that can overload machines (BM1). This is due to adapting the number of concurrently running tasks based on task resource use and machine resource availability.

All four task schedulers perform similarly for lightly loaded benchmarks, e.g., BM6–BM9 (Figure 4.9). This is due to the lack of resource contention or overloading, which makes

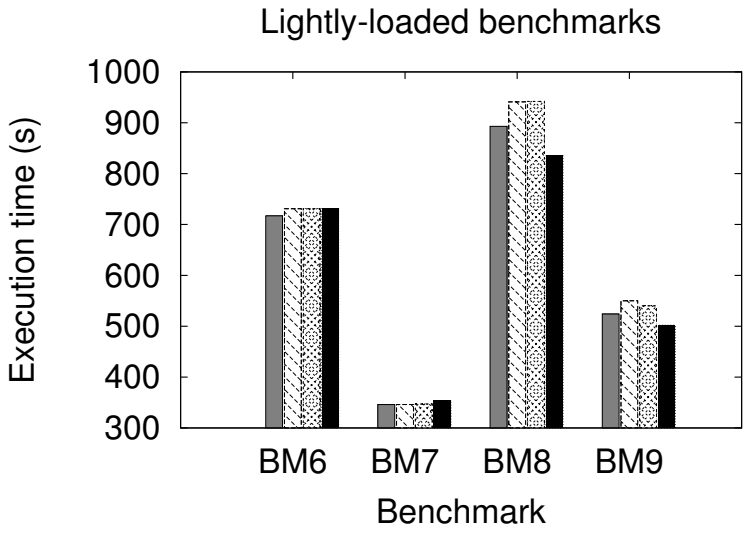


Figure 4.9: Execution time comparison of schedulers for lightly-loaded benchmarks on a 30-node cluster.

task assignment policies unimportant. Such loads often occur at night. While we cannot improve the performance of such workloads, we can improve the energy consumption.

4.5.5 Energy Consumption Reduction

This section explains the impact of scheduling policy on energy consumption for clusters using task concentration.

We run the same groups of benchmarks described in Subsection 4.5.2 in the multi-node cluster and measure the resource utilization of every machine during execution. These data are used to calculate machine power consumption using the model described in Table 4.2. The energy consumption is then computed using Equation 4.9 with the parameter values from the power model and measured resource utilization. We require benchmarks to finish in the same amount of time as they require during the most heavily loaded period in the diurnal cycle [65].

Figure 4.10 shows energy savings for *ori* and HAMS for five benchmarks taken from Subsection 4.5.2. *Simple* and *hetero* perform only slightly better than *ori*, therefore we focus

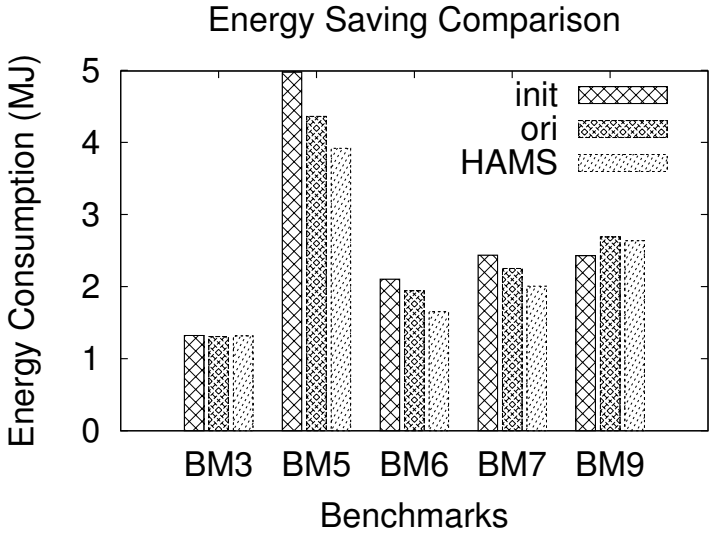


Figure 4.10: Energy savings with task concentration for *ori* and HAMS. *init* does not use task concentration.

.

on *ori* and HAMS. *Ori* achieves 10% average reduction in energy consumption given the constraint that night-time performance must be at least as good as daytime performance. HAMS achieves 23% reduction in energy consumption because it optimizes resource sharing among concentrated tasks, allowing more machines to be powered down while honoring performance constraints (see Figure 4.11). To summarize, HAMS reduces energy consumption compared to *ori* under the same latency constraint.

The amount of energy saved by HAMS depends on workloads. In benchmark BM5, BM6, and BM7, task concentration reduces energy consumption for both HAMS and *ori*, although HAMS does better. Note that BM6 and BM7 are lightly-loaded benchmarks for which task assignment policy has little impact on performance, but are suitable for energy reduction because they leave rooms for task concentration. BM3 and BM9 are not suitable for task concentration because they very heavily load the data center, leaving few idle machines to power down. If a data center is almost always heavily loaded, task concentration doesn't help.

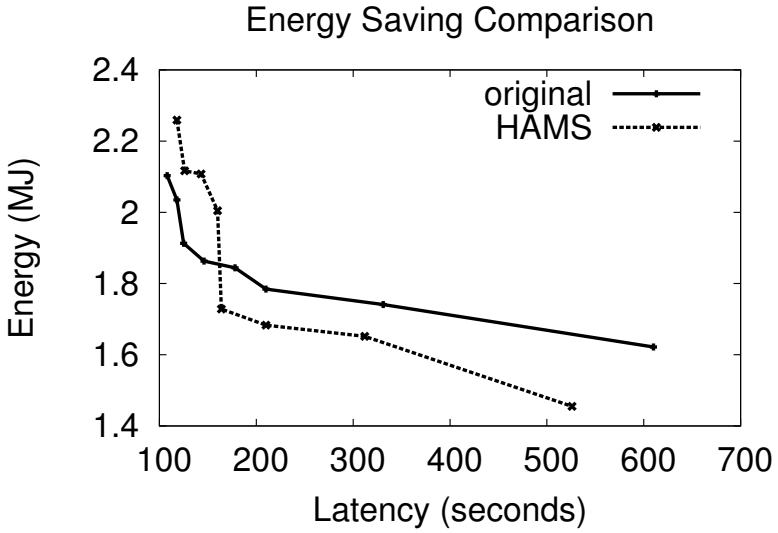


Figure 4.11: Energy consumption–latency relationship for Benchmark 6.

4.6 Additional Feature Considering Data Locality

Our previous discussion on HAMS has focused on task and machine resource sharing only. Data locality can also significantly impact data center task execution. Assigning a task remotely, i.e., on a machine that does not contain the input data replication of the task, requires the input data of the task to be transferred through the network, increasing task execution latency.

One common solution to this problem is to always assign a task to the machine containing its input data, e.g., to keep data locality. This solution reduces the overall job execution latency because transferring workloads (tasks) has smaller overhead than transferring data. Some works compute the probability of scheduling a task based on their data replications, and select the one with the highest locality [75, 76]. Ibrahim et al. design a key partitioning method to partition keys of Map tasks according to their input data location. Some other approaches improve data locality through initial data placement and redistribution. These works distribute data in proportion to the computational abilities of machines, i.e., hardware setup of servers [77–79].

However, the above assumption does not always hold. Keeping all tasks on their data-

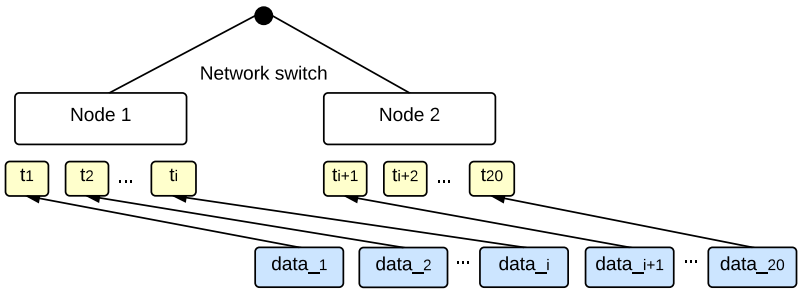


Figure 4.12: Experiment setup for the motivating example. The input data of all tasks reside in node 1. We test the assignment of these 20 tasks starting from running all tasks on node 1 (all local) to running all tasks on node 2 (all remote), increase the number of remote task by 1 each time.

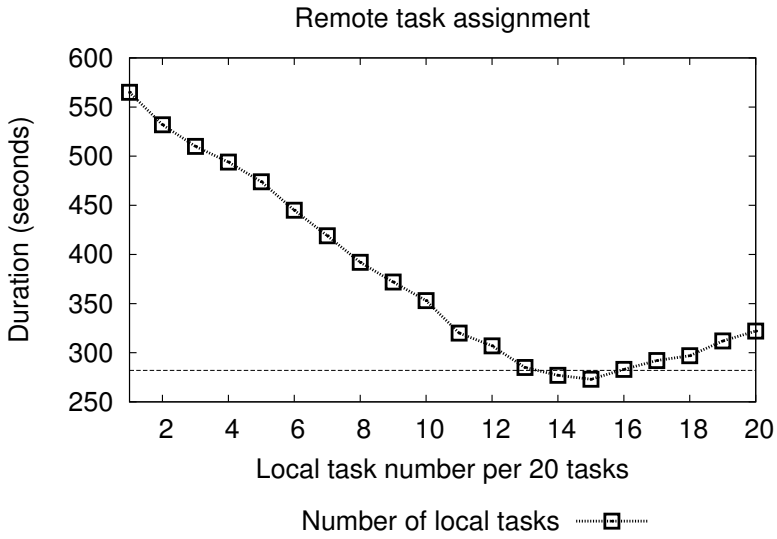


Figure 4.13: Change of workload durations as the number of remote tasks changes.

local machines can result in sub-optimal resource sharing, if these local tasks on the same machine happen to compete on the same resource. Severe resource contention can sometimes result in higher task execution time compared to the delay induced by remote data transfer. On the contrary, remote tasks can have shorter execution time if they are assigned to remote, but resource-contention-free machines. Existing works that consider resource contention on remote machines use the CPU loading as one single indicator of the total loadings on a machine, ignoring contention on multiple resources [80].

A task scheduler that considers resource utilization and remote assignment together

needs to balance between these two effects. Only considering resource balancing or targeting best locality can both lead to sub-optimal assignments. We use one motivating example to demonstrate the idea. We perform a test with 20 tasks running on a cluster of two nodes (Figure 4.12). The execution time of these 20 assignments are show in Figure 4.13. The lowest execution time is not achieved by "balanced assignment", which is 10 tasks in each node, nor "local assignment", which has no remote tasks. Instead, optimal assignment contains slightly lower number of remote tasks (5 tasks) and higher number of local tasks (15 tasks). This is because the balanced assignment fails to consider the task input data transfer delay, therefore slows down the machine with remote tasks; The local assignment overloads machine resources with too many concurrent tasks, slowing down task execution by resource contention.

In summary, an optimal task assignment scheme depends on the overhead of two effects: resource contentions caused by local tasks, and the network transfer latency of remote tasks. To capture the impacts of these two effects during the task assignment process, detailed models on task execution and network transfer are required.

4.6.1 Models

We propose to use three models to describe the performance impact of local and remote tasks.

1. *Local task resource utilization and execution time model:* It describes the change in resource utilization, and the corresponding slow down in task execution time when assigning a new task on the same machine. This model is already described in our previous task scheduler design, *HAMS*, in Chapter 4.

2. *Network transfer model:* It calculates the delay of transferring data through networks.

3. *Remote task resource utilization model:* It models the change in resource utilization of the source machine, the machine that provides data for a remote task.

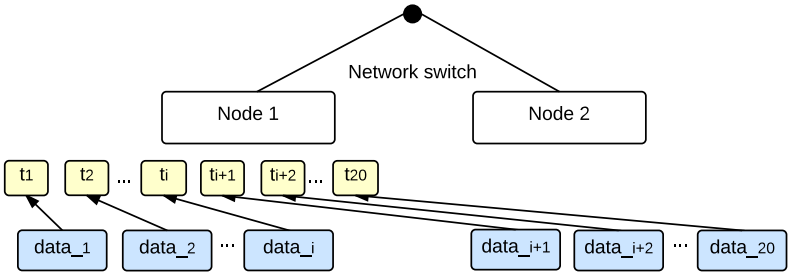


Figure 4.14: Experiment setup for building the network transfer model.

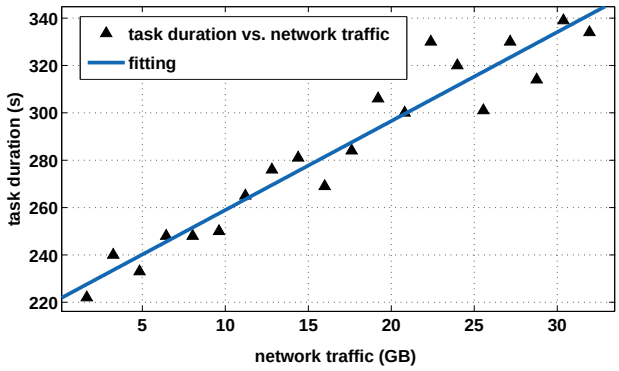


Figure 4.15: Network transfer model for one task.

The first model has already been constructed in our previous work; we describe the format and construction process of the remaining two models in the following sections.

4.6.1.1 Network Transfer Model

The network transfer model describes the delay on a remote task execution caused by network data transfer. This network delay in a data center is mainly caused by insufficient capacity of the network switches. It is a function of the network background loading, and is independent of the task type. Therefore, we can characterize the network transfer separately.

The network background loading and the corresponding network delay can be directly measured. I design an experiment to gather these data and construct the model. The experiment uses two nodes (node 1 and node 2). All 20 tasks are initially running on node 1. As demonstrated in Figure 4.14, there are i sets of data on node 1 and 20 - i sets of data on

node 2. Varying i from 0 to 20 changes the number of remote tasks from 20 to 0. This allows us to rule out other effects (e.g., resource contentions from local tasks) and only consider the delay change due to network transfer. For each execution, we measure the finish time of the task set and also the network traffic, and perform a curve fit to extract model parameters.

The resulting network delay model is linear, as shown in Figure 4.15. It takes the following format:

$$execution\_time(second) = p1 \times network\_data(GB) + p2. \tag{4.10}$$

$p1$ is the fitting parameter, and $p2$ is close to the task execution time on a data local, unloaded machine. While $p1$ is independent of tasks and can be used across different task types, $p2$ can be taken from the previously built task execution time model from HAMS. In our measurement, we obtain the value of $p1$ and $p2$ as 3.8 and 221.4.

The above characterization process assumes that the measured delay increase is completely caused by the network transfer latency. However, part of this delay can also be a result of the increase in disk IO when remote data are written to local disk. We measure the disk IO increase on the destination machine, and compare it with that of a local task. The additional disk IO ranges from 2% to 9%, which is not sufficient to significantly change the task execution time. Therefore, we ignore this effect in our characterization process and treat the measured delay as purely caused by network data transfer.

4.6.1.2 Remote Task Resource Utilization Model

The remote task resource utilization model captures the resource utilization changes on the source machine due to data transfer. More specifically, it captures the increase in disk IO utilization.

Measurement shows that the increase in disk IO is proportional to the remote data size.

Therefore, we describe the increase in disk utilization below:

$$\frac{num\_of\_remote\_task \cdot \alpha \cdot remote\_input\_size}{num\_of\_local\_task \cdot local\_input\_size}, \tag{4.11}$$

in which α is a scaling factor.

4.6.2 Scheduler Design

We design a task scheduler that integrates the above three models together, and determines weather to assign a local or remote task on a machine when called. If assigning a remote task, the scheduler also determines the source machine that the task should read its input data from.

The assignment process computes task execution overhead for all candidate tasks, both local and remote, and select the assignment that results in the highest overall throughput in both the local and remote machines. It considers resource contention generated by the candidate tasks. This is done by computing the impact of this candidate assignment on already-running tasks residing on the destination machines. It also considers the increase of disk IO on all candidate source machines. These impacts can be captured using the network transfer model and remote task resource utilization model. The process of scheduling one task is demonstrated in Figure 4.16. Task $t1$ is assigned to machine 1 as a remote task. It has two impacts: the network delay $n1$, which is calculated using the network transfer model, and the increase in the disk IO of the source node (machine 0), which is calculated using the remote task resource utilization model, and is represented in the disk utilization of task $t0$.

The above process is repeated every time when an assignment decision is made, resulting in repeated computation of the network delay and resource utilization. This computation is redundant if the loading conditions of the source machines stay unchanged or only change slightly. We use a global bookkeeping data structure to prevent these unnecessary overhead, and only update this information on-demand, as demonstrated in the table in Figure 4.16. It

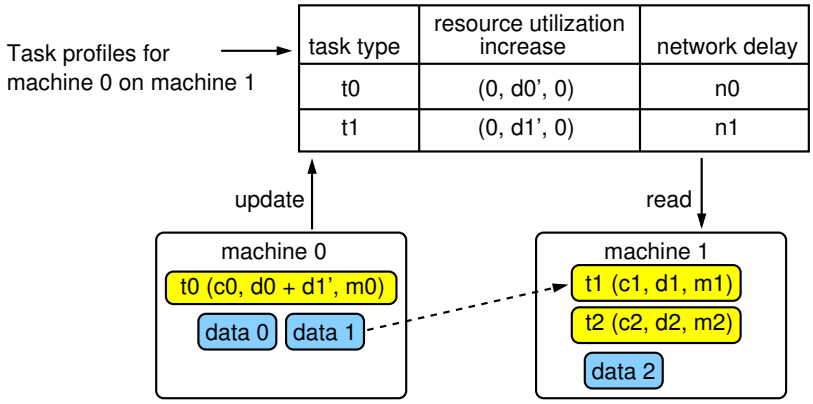


Figure 4.16: Overview of the task scheduler.

records the network transfer delay of a unit size data trunk between two machines, and also records the modeled task resource utilization and the corresponding execution time on every candidate remote machines. Both this network delay and remote task execution time are affected by the background loading on the source and the destination machines. As a result, they are only updated when the network loadings or the loadings on the remote machine change.

4.6.3 Experimental Results

We integrated the network delay model into HAMS and implemented the above task assignment algorithm. Tasks running on machines containing their input data are called *local tasks*. Tasks getting their input data from machines within the same rack, or from a different rack, are called *remote tasks*. The data locality rate is the percentage of local tasks out of all tasks. We use this data locality rate to describe how much locality a task scheduling algorithm should have to achieve best performance. We evaluate two aspect of the improved scheduler: First, we measure the data locality rate that results in optimal job finish time. Second, we compare the overall task execution time of this remote task aware algorithm with HAMS. We compare the improvement in overall benchmark execution time when explicitly considering data locality and remote tasks.

Table 4.4: Average Data Locality And Job Finish Time

Scheduling Algorithms	Original	HAMS	Local	NHAMS
Data locality rate	12%	39%	100%	45%
Normalized job finish time	1.0	0.62	0.84	0.51

We measured the average data locality rates and average improvement in job finish times for several task scheduling algorithms:

- *The hadoop original scheduling algorithm.* This algorithm completely ignores data locality.
- *The HAMS without the network delay model.* This scheduler targets optimal resource utilization among tasks, while does not guarantee data locality.
- *The local scheduling algorithm,* which runs every task on machines containing its input data.
- *NHAMS: The HAMS with network delay model and locality aware scheduling.* This algorithm integrates network transfer model together with task resource and performance models.

We deploy the above schedulers on a 16-node cluster. This cluster has two racks of machines connecting via one network switch. Each rack contains 8 machines. One rack has tpye 1 machines, and the other rack has type 2 machines. We run three benchmarks using the four scheduling algorithms, and measure and calculate the average data locality rate and performance change.

The results are shown in Table 4.4. NHAMS achieves shortest job finish time, while its average data locality rate is only 45%. This result in a data locality rate lower than the local scheduling algorithm, and higher than HAMS. This result shows that neither running all tasks locally, nor only considering task resource sharing but ignoring data locality can

minimize job finish time. Combining the two models together achieves the best performance in task execution. This result is consistent with our example shown in Figure 4.13.

4.7 Related Work

This section summarizes related work in the areas of task performance modeling, data center load-balancing, and energy optimization.

Resource utilization based task modeling is common in data center task scheduling. It optimizes task performance by doing load-balancing on machines. There are two variants of this approach. (1) Some researchers formulate this problem as multi-dimensional vector packing. Mastroianni et al. use a probability function based on CPU and RAM utilization to make assignment decisions [61]. Borgetto et al. use a metric called *yield* to describe the ratio of required to maximum possible resource utilization [62]. These methods model tasks individually. They do not consider resource contention among tasks or machine heterogeneity. (2) Other researchers construct detailed models focusing on sharing one specific resource. Tang et al. evaluated the impact of memory resource sharing on co-located tasks and designed a thread-to-core mapping algorithm that is suitable for such sharing [81]. We take a more general approach and consider memory, CPU, and IO.

Task and machine heterogeneity have been modeled in prior work. Ganapathi et al. developed statistical models to predict execution times for MapReduce jobs [82]. Their model is based on features peculiar to MapReduce jobs, while our modeling approach is based on measured responses to heterogeneous resource utilization, and can therefore be readily extended to other cloud computing tasks. Delimitrou et al. designed Paragon [83], a server heterogeneity aware scheduler that selects the most suitable server for workloads based on server configuration and interface. Zaharia et al. develop LATE [84], a scheduler that improves speculative task assignment based on estimated task finish time. They explicitly consider machine heterogeneity when computing the expected finish time. These models

do not consider the impact of resource utilization on task performance, while we provide a more accurate model that captures both heterogeneity and resource contention.

There are several methods of optimizing data center energy use by task concentration and powering down sub-group of machines. Meisner et al. developed PowerNap [2], a system to minimize machine idle power and transition time. Their application requires hardware redesign. Our approach does not require hardware changes and makes high-latency power state transitions less harmful. Pinheiro et al. migrated tasks into sub-groups of machines [60]. They determine when to migrate tasks and disable/enable nodes to minimize energy consumption. Their model for data center throughput neglects load-dependent resource usage. Lang et al. proposed to use all machines in a cluster to finish the workload and power down the whole cluster [4]. This approach can reduce energy consumption when there are significant time gaps between job arrivals. However, it does not work when jobs frequently arrive, leaving little or no gap for powering down machines. HAMS can save energy in the presence of frequently or infrequently arriving jobs.

4.8 Conclusion And Future Work

This chapter has describe HAMS, a task scheduler for the Hadoop MapReduce framework that accurately models the relationships among task resource utilization, performance, and power consumption in order to optimize performance and energy consumption. HAMS predicts task execution time and optimizes task assignment decisions by explicitly considering task and machine resource heterogeneity using a background-loading-aware task performance model. This brings an average performance improvement of 13% over the Hadoop original scheduler and 11% over schedulers that overlook task and machine heterogeneity or machine background loading. When used together with task concentration, HAMS reduces energy consumption by 23% while meeting task latency requirements. Existing scheduling policies reduce energy consumption by only 10% under the same conditions. We

also consider the impact of locality when data replication is insufficient to provide local input data to tasks. Adding a network transfer model to HAMS achieved an additional 9% performance improvement. Finally, we discussed the data locality rate favorable for task execution time.

CHAPTER 5

Reliability-Aware Cooling Energy Saving Through Task Assignment in Heterogeneous Data Centers

5.1 Introduction

Data center energy consumption has become a significant problem. Commercial data centers consumes millions of dollars per year of electricity; the computational capacities of data centers are typically constrained by their energy availabilities.

Data center energy consumption can be reduced by reducing cooling energy. Cooling energy accounts for a large portion of data center total energy consumption. The cooling efficiency of a data center can be quantized by Power Usage Effectiveness (PUE), which is the ratio of the computational energy to the total energy. The PUE of a traditional data center ranges from 1.5 to 2 [85, 86]. Therefore, reducing cooling energy can result in 30% to 50% reduction in data center energy consumption.

People have proposed energy saving solutions for data centers. However, the focus has been on server energy reduction, while cooling energy is often ignored. Even when people consider cooling energy, it is optimized independently from server energy [87, 88]. Cooling energy consumption in a data center consists of server fan energy and cooling system energy. The energy consumption of server fans is a function of server processor temperature, and

therefore is dependent on server utilization. The energy consumption of data center cooling system strongly depends on the number of active servers and the temperatures of server heat sinks. Therefore, cooling energy is strongly dependent on server operating states. This observation leads us to consider the effects of concurrently optimizing server energy and data center cooling energy.

It is not straightforward to optimize both server and cooling energy consumption, as there is a trade-off between the two. Cooling energy decreases when the operating temperatures of servers increase. Nevertheless, higher temperature setpoints also increase energy consumptions in server processors. In 45 nm or newer technology, a large portion of server power is due to leakage, which increases exponentially with temperature. Figure 5.1 illustrates the temperature dependence of the data center cooling, leakage, and total energy consumption. The optimal overall energy consumption is achieved when servers operate at 50 °C or higher.

These temperature setpoints of server processors are higher than the current operating temperature range used in data centers (e.g., 30 °C–40 °C). These operating temperature standards are used to avoid reliability problems. The reliability of a server processor is affected by temperature-dependent fault mechanisms such as Time-Dependent-Dielectric-Breakdown (TDDB), Electromigration (EM), Thermal Cycling (TC), and Negative-Bias Temperature Instability (NBTI) effects. These effects are more severe at high temperatures. Running servers at higher temperature will certainly shorten the lifetimes of server processors. However, it may not much affect the overall system lifetime. The system lifetime of a server is constrained by the least reliable components, which in most cases are hard drives and RAM rather than the processors. The hard drives are usually the first to fail, with average lifetimes of half a year, followed by the RAM, which typically fails within one year [89]. The reliabilities of these two components are not strongly temperature-dependent; therefore, it is safe to push the processor to run at a higher temperature until its lifetime reduces to that of the hard drives or any user-specified lifetime constraints, while optimal energy saving is achieved.

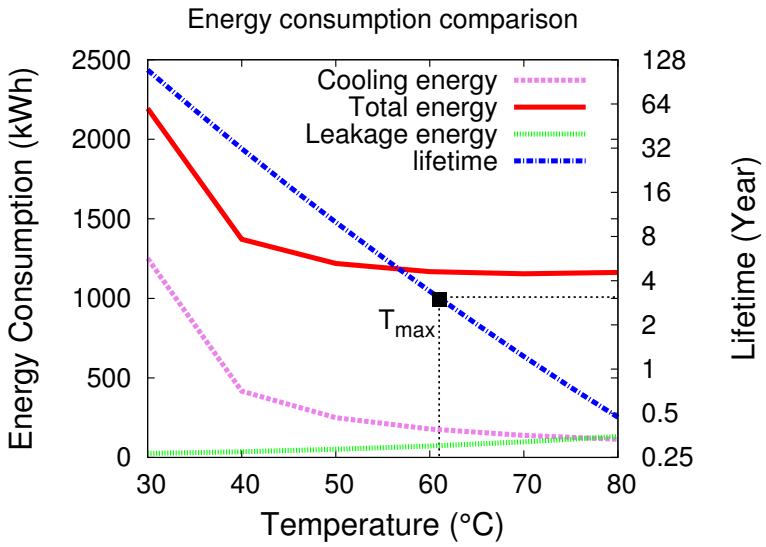


Figure 5.1: Relationship between data center energy consumption, temperature, and processor lifetime.

The temperature setpoints of processor cores are affected by two factors: the total energy consumption of the data center and the reliabilities of the processor cores. There are two temperature-dependent terms in the data center total energy consumption: the cooling energy and leakage energy. Cooling energy dominates data center total energy [1]. When processors are set to run at a higher temperature, less heat is required to be removed from the server room to reach the desired processor temperature setpoints, so the cooling energy is lower. Therefore, the constraint on total energy consumption sets the minimum processor temperature (T_{min} in Figure 5.1) that will result in acceptable total energy consumption. The reliabilities of processor cores can be described by the mean-time-to-failures (MTTFs) of the cores, which decrease with rising temperature. The lower bound of core MTTF sets the maximum value of the processor operating temperature (T_{max} in Figure 5.1).

In data centers using multi-core servers, it is suboptimal to set one single temperature setpoint for all server processors. This is due to heterogeneity of server processor cores. This heterogeneity has three sources: First, the server processors of different machines are at different wear states, since they are either of different ages, or have different utilization

histories. Second, cores have different loadings. Third, cooling conditions vary among servers because the temperature distribution and air flow are not exactly the same for every machine. As a result, the optimal temperature setpoints for each processor have to be calculated separately. On the other hand, different cores have different requirement for cooling air temperature. This air temperature is usually determined by the output air temperature of the cooling equipment–usually chillers–in data centers. However, in practice, this temperature is set to one single value for a whole rack of servers. The operating temperature setpoint of each server depends on the operating state of all servers. This requires detailed thermal modeling of the data center, and the complexity scales up with the number of servers.

People have proposed data center energy optimization techniques under reliability constraints. These works focus on reliability-aware DVFS techniques [90] or workload scheduling [91]. Existing methods have two disadvantages. First, they only consider energy saving on a single server, without considering the impact on data center cooling system energy consumption. Second, they do not have a way of measuring the reliability influencing wear states of individual processor cores and their changes over time. In contrast, we optimize data center energy consumption as processor wear states change. Furthermore, we propose a method to measure the processor reliability.

The reliability constraint shown in Figure 5.1 is determined by the processor MTTF and its wear state. While the required core MTTF is pre-determined and is age-independent, the processor wear state, which describes how much the core wears out due to different fault mechanisms, depends on the operating states of the processor, and changes over time as processor cores are stressed. The wear-out of processors increases with the processor stress time, and in most cases is also temperature dependent. However, it is not straight-forward to directly measure the processor wear state, before the core fails.

We developed a method of measuring processor wear state that focuses on one type of faults–the timing error. Timing errors are detectable during normal processor operating

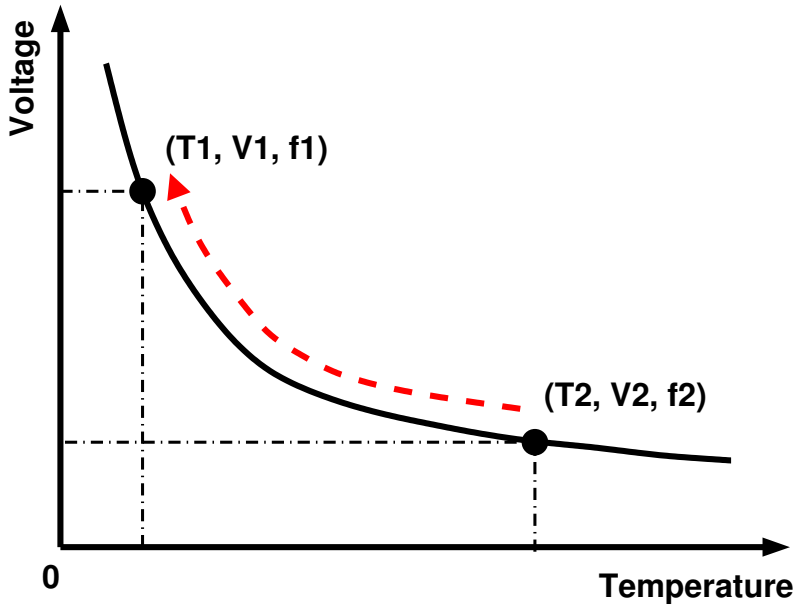


Figure 5.2: This figure shows a voltage–temperature dependency curve of a processor. This can be retrieved through circuit modeling of the critical path of that processor. Using this curve, one can determine the crashing frequency f_1 under normal operating voltage and temperature (T_1, V_1) by tracing back from the extreme operating condition (T_2, V_2, f_2) at which there is a processor core crash due to timing error.

and are still an outcome of several major fault mechanisms that lead to processor lifetime degradation. A timing error can be observed directly when the system fails to operate. This happens when the timing slack in the critical path of the processor reduces to zero or negative as the processor wears out. Therefore, a measurement of the timing slack tells people how far the processor is from a timing error. At each fixed wear state, the timing slack reduces as the processor runs at extreme operating conditions (e.g., high operating temperature or low core supply voltage). Therefore, one could stress the processor to extreme operating conditions, and record the temperature-voltage-frequency points at which timing errors occur. Using this information, one can deduce the remaining timing slack under normal operating condition (lower temperature and higher core supply voltage) by modeling the temperature-dependent timing change on the critical path, as shown in Figure 5.2. Performing this measurement at every stage of processor life allows us to measure wear progression and build a map of the time-dependent processor wear state.

The above wear state measurement requires setting the processor at different Dynamic Voltage and Frequency Scaling (DVFS) states, and restarting the server when a timing error happens. This test can be performed without physical access to the servers by using a remote-controlled switch to detect the occurrence of timing errors, and restart the machine.

5.2 Related Work

Existing works have used different approaches to minimize data center energy consumption. Some work has considered only one stand-alone servers and other work has considered a whole data center. First, when considering only one stand-alone server, people optimize the socket-level workload distribution to reduce server fan energy consumption. Ayoub et al. proposed *cool and save* [87], with the goal of reducing the overall energy consumption of servers by doing socket-level task assignment. This technique saves cooling energy by reducing CPU fan speed. Coskun et al. developed a temperature aware task scheduling method to balance workloads in MPSoCs (Multiprocessor system-on-chip) [92]. They used a method called Adaptive-Random to calculate the task assignment probability of each core based on its temperature difference with the threshold temperature. Huang et al. designed TAPO [88], a technique that selects the temperature setpoints of server machines and data center ventilation systems. However, these techniques consider the data center cooling system and servers separately. We descirbe a method to co-optimize the cooling system and computation system, and will show that this can achieve better overall energy savings in later parts of this chapter. Second, when considering a data center, people propose to save energy through optimizing energy distribution among different servers. Das et al. provide a utility function based solution to power down under-utilized servers to save energy [93]. Chen et al. use a predictive model to find the temperature setpoints for server inlets [94].

Some works characterize and model the cooling system to optimize the cooling control flow. Heath et al. designed Mercury, a system that models the heat and air flow to emulate

temperature distribution in the data center [95]. Breen et al. designed a system model of server heat sink temperature as a function of data center air-flow and server temperature setpoints [96]. In later part of this work, we will integrate some of these models into our data center cooling system model.

People have also done research on energy optimization under reliability constraints. Basoglu et al. proposed NBTI-aware DVFS, a voltage scaling scheme that assigns optimal operating voltages to processor cores [90]. Sun et al. designed a NBTI-aware workload balacing scheme for multi-core systems, with the goal of reducing the variation in lifetime degradation of multi-core systems [91]. These works focus on a multi-core system in a stand-alone server. We focus on data center clusters.

5.3 Problem Definition

The goal of this work is to optimize the total energy of a data center. The total energy consists of two major parts: the server computational energy $E_{machine}$, and the cooling energy $E_{cooling}$. These two terms are directly affected by tasks running on the servers. Furthermore, cooling energy is also a function of several user-controlled variables in the data center setting. Therefore, we use dynamic task assignment that changes the above variables to achieve optimal energy consumption. In addition, we impose a reliability constraint on our solution.

1. **Objective:** minimize data center total energy consumption, including computation energy and cooling energy, while trying to meet all task deadlines. The total energy consumption of the data center can be expressed as follows:

$$E_{total} = \sum_{i=0}^{n-1} E_{machine,\, i} + E_{cooling}, \tag{5.1}$$

in which n is the number of servers.

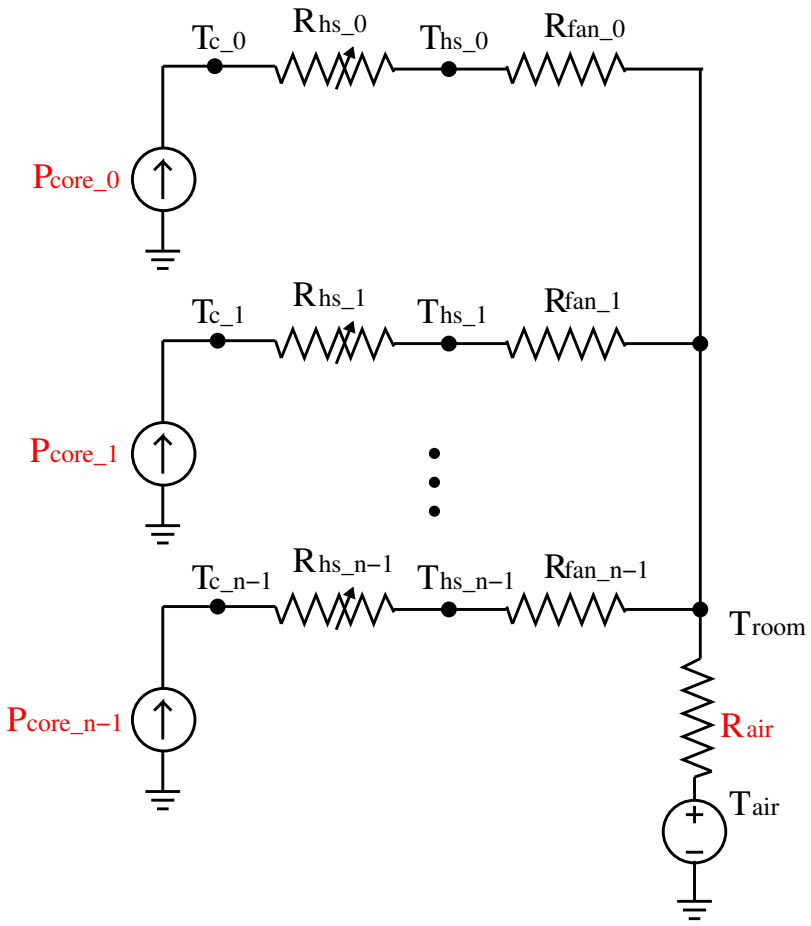


Figure 5.3: Thermal transfer network model for an air-cooled data center. Each core is modeled as a power source P_{core_i}. Server fans and the chiller are modeled using their thermal resistances R_{fan_i} and R_{air}. T_{c_i}, T_{hs_i}, T_{room} and T_{air} are the core temperature and heat sink temperature of core i, server room temperature, and outside air temperature.

The cost function can be expressed as the total energy plus the deadline cost function.

$$f = E_{total} + \sum_{i=0}^{n-1} (d_i - d_{i0})^{\alpha},$$

in which α is a parameter larger than 1. Therefore, assignments that result in little or no deadline violation for tasks are favorable. Both the energy term and task deadlines are affected by task assignment decisions.

2. **Input:** A pool of tasks. At the time when an assignment decision is made, we assume that the scheduler always knows the type and number of all candidate tasks. A new

group of tasks can arrive later and will of course affect the scheduling decision; however, the time interval between the arrivals of two groups of tasks is generally longer than the average execution time of one task. Therefore, the above assumption about the task scheduler still holds under most circumstances.

3. **Controllable Variables:**

   - task assignment (when and where to assign a task),
   - processor operating voltage (V_c),
   - processor frequency (f_c), and
   - chiller air flow rate (F_{air}).

4. **Constraints:** MTTF of each core. It is a function of the failure rate of each core:

$$\textit{Processor mean-time-to-failure: } \textit{MTTF}(T_c, f_c, V_c, t_i) < \textit{MTTF}_{\textit{constraint}}.$$

Calculating the above objective function and constraint requires detailed modeling of data center energy consumption and the processor lifetime. We discuss our models in the following sections.

5.3.1 System Overview

We present the overview of our experiment and task scheduling design. This design consists of two parts: a wear state crash test and a reliability aware task scheduler. The system diagram is shown in Figure 5.4.

The wear state crash test is performed on each individual core. When running this test, the core is loaded with a group of CPU stressmarks. They exercise the core to cover most DVFS states until the core crashes. The crash states are used to compute processor temperature setpoints using two models: the core reliability model, and the circuit model of the core critical path. The temperature setpoint of each core is used in the task scheduler.

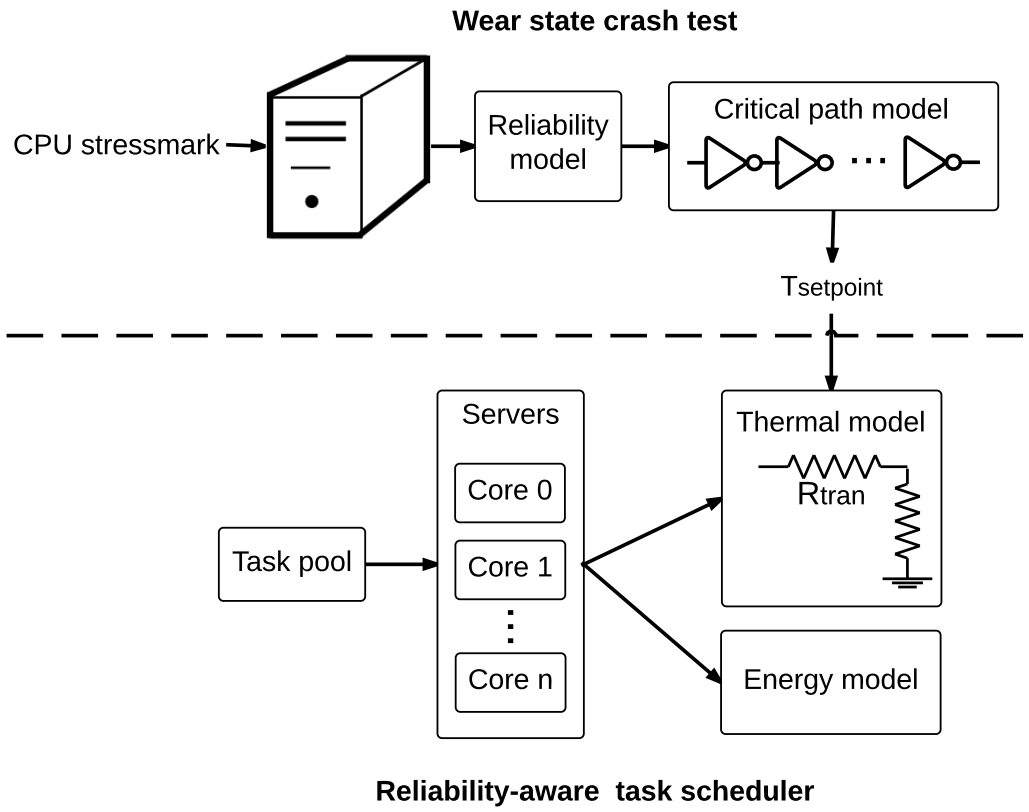


Figure 5.4: System diagram of wear state test and reliability aware task scheduler.

The reliability aware task scheduler minimizes data center energy consumption through task assignment. The scheduler selects tasks from the input task pool and assign them to cores, targeting at their temperature setpoints. We use a data center thermal model to calculate core temperature when assigning candidate tasks. Also, the energy consumption of each assignment is calculated using a data center energy model. The scheduler then selects the assignment with minimum energy consumption under the core temperature constraint.

We discuss the above models and two parts of our design in the following sections.

5.4 Modeling

In this section, we describe several models that will be used in our design: the energy and thermal models of servers, and reliability models. These models are used in our wear state

measurement and task scheduler design.

5.4.1 Energy and Thermal Modeling

The total energy consumption of the data center consists of server energy consumption and cooling energy consumption. The server energy consumption can be easily modeled using the CPU utilization. The cooling energy is dependent on both the core power consumption and its operating temperature, and therefore requires using a thermal transfer network.

The total energy consumption of server processors is the sum of individual core energy consumptions. The core energy consumption consists of two parts: the idle energy, which is mostly independent of the workload; and the utilization-dependent active energy:

$$E_{server} = \sum_{i=0}^{n-1} \left(E_{idle} + u_i \cdot E_{active} \right), \tag{5.2}$$

in which n is the number of servers, E_{idle} and E_{active} are the idle and active energy consumptions of the server, u_i is the resource utilization vector of machine i.

Chiller energy is the main contributor to cooling energy consumption. We assume that the chiller uses air cooling only. It pumps in ambient air and uses it to cool the data center. Therefore, the chiller energy consumption is a linear function of its air flow rate F_{air}:

$$E_{chiller} = A_{chiller} F_{air}. \tag{5.3}$$

$A_{chiller}$ is a constant.

In order to calculate the cooling energy consumption, we need to find the value of F_{air}, which is determined by the processor power consumption and the ambient air temperature outside the data center. This relationship can be described using the thermal transfer network shown in Figure 5.3, which models the heat distribution among each core, and shows the heat flow from processor fans to the central cooling system [86]. The thermal relationship in

one individual core and its fan can be modeled as follows:

$$P_{core\_i}(R_{hs} + R_{fan\_i}) = T_{c\_i} - T_{room}. \tag{5.4}$$

P_{core_i} is the power consumption of the ith core, R_{hs} and R_{fan_i} are the thermal resistance of the heat sink and the server fan, and T_{c_i} and T_{room} are the core and room temperature.

The thermal relationship between the server room and the chiller can be described using the following equation:

$$R_{air} \sum_{i=0}^{n-1} P_{core\_i} = T_{room} - T_{air\_out}. \tag{5.5}$$

Thermal resistance of the air-cooled fan system (R_{air}) can be modeled using its air flow rate: F_{air}. This air flow rate F_{air} must be sufficient to cool the servers. That is, the amount of heat taken away by the chiller outlet should be equal to the heat generated by servers [87, 97]. For each server, the following equations hold:

$$R_{air} = 1/(F_{air} C_{air}) \; and \tag{5.6}$$

$$R_{fan\_i} = 1/(F_{fan\_i} C_{air}). \tag{5.7}$$

Considering mass conservation in air flow, we have the following relationships among air flow rates:

$$\sum_{j=0}^{n-1} F_{fan\_j} = F_{air}. \tag{5.8}$$

Solving F_{fan_i} from Equation 5.4 and Equation 5.5 yields

$$F_{fan\_i} = \frac{P_{core\_i}}{T_{c\_i} - T_{air\_out} - \frac{1}{F_{air}} \sum_{j=0}^{n-1} P_{core\_j}}. \tag{5.9}$$

Substituting Equation 5.9 into Equation 5.7 yields an expression of F_{fan} using P_{core_i} and

T_{c_i}:

$$F_{air} = \sum_{i=0}^{n-1} \frac{P_{core\_i}}{T_{c\_i} - T_{air\_out} - \frac{1}{F_{air}} \sum_{j=0}^{n-1} P_{core\_j}}. \tag{5.10}$$

In the above expression, processor power consumption P_{core_i} is a function of the workload on the core, which is determined by the task assignment algorithm. The core temperature T_{c_i} is constrained by the reliability requirement of the processor, which can be solved for using the reliability constraint.

5.4.2 Reliability Constraints

The system reliability constraint sets a lower bound on the required MTTF of the system. This required MTTF of the system is the expected time at which the processor will first encounter a failure. In the rest of our discussion, we focus on failure introduced by NBTI effects. NBTI is one of the dominating mechanism for circuit failure [98]. It causes circuit failure by slowing down transistors and leads to a timing error, which can be caught by external measurements. NBTI affects circuit timing due to increasing threshold voltage, thus slowing down the circuit. Therefore, the NBTI effect can be modeled by considering time-dependent change in the threshold voltage [98, 99]:

$$\Delta V_{th}(t) = \Delta V_{th\_max} \cdot (1 - e^{-(t/\tau)^{\beta}}) \ \ and \tag{5.11}$$

$$\tau = \left(\frac{N_i \cdot e^{E_H/kT_c} \cdot D_{00}}{kT_c \cdot \beta} \right)^{-1/\beta} \cdot {E_{ox}}^{-1/\beta}. \tag{5.12}$$

The time constant τ indicates when ΔV_{th} reaches 63% of V_{th_max}. τ is solely dependent on the operating temperature T_c of the circuit, while N_i, E_H, k, D_{00}, β, and E_{ox} are all process-dependent constants.

We calculate the system MTTF as the time when the change in the threshold voltage

$\Delta V_{th}(t)$ reaches a certain portion of the maximum voltage ΔV_{th_max}:

$$\Delta V_{th}(MTTF) = D \cdot \Delta V_{th\_max}, \tag{5.13}$$

in which D is a constant specified by the user.

The system MTTF is affected by both the core stress time t and core wear state. The wear state describes the wear-out of processors, i.e., the reduction in processor MTTF caused by fault mechanisms. It can be measured using our proposed experiment. The analysis yields the current threshold voltage V_{th_c}. The remaining lifetime after stress should still satisfy the reliability requirement:

$$\Delta V_{th}(t_{remaining}) + (V_{th_c} - V_{th0}) = D \cdot \Delta V_{th\_max}, \tag{5.14}$$

in which V_{th0} is the starting threshold voltage before stress.

Using the above equation together with Equation 5.12, we can calculate the remaining lifetime of the circuit under stress:

$$t_{remaining} = \tau \left[-1 \cdot ln \left(1 - D + \frac{V_{thc} - V_{th0}}{\Delta V_{th\_max}} \right) \right]^{1/\beta}. \tag{5.15}$$

This $t_{remaining}$, plus the already stressed time, has to be longer than the required system lifetime:

$$t_{remaining} + t_{stressed} \geq MTTF_{constraint}. \tag{5.16}$$

Using Equation 5.15 and Equation 5.16, we can solve for the lower bound on the

temperature-dependent time constant τ_{min}:

$$\tau_{min} = (MTTF_{constraint} - t_{stressed}) \cdot \left[-1 \cdot ln \left(1 - D + \frac{V_{thc} - V_{th0}}{\Delta V_{th\_max}} \right) \right]^{-1/\beta}. \quad (5.17)$$

Combining this equation with Equation 5.12, we can solve for the operating range of core temperature T_c. Therefore, the reliability constraint can be transfered into a temperature constraint of each processor core. The range of T_c can be used in solving F_{air} from Equation 5.10, which directly gives us the cooling energy consumption.

5.4.3 Wear State Measurement

The processor wear state can be described using the increase of threshold voltage of transistors. This increase affects processor delay d:

$$d = A \cdot (V_{GS} - V_{th})^{\alpha}, \quad (5.18)$$

in which A and α are constants, and V_{GS} is the gate-to-source voltage. Therefore, the maximum increase in processor threshold voltage ΔV_{th_max} is a function of the maximum allowed critical path delay of the processor, d_{max}, and the current increase in threshold voltage ΔV_{thc} is a function of the current critical path delay d_c. Using Equation 5.18, we derive the expression of ΔV_{th_max}:

$$\frac{V_{GS} - V_{th0} - \Delta V_{th\_max}}{V_{GS} - V_{th0}} = \left(\frac{d_{max}}{d_0} \right)^{1/\alpha} \quad and \quad (5.19)$$

$$\Delta V_{th\_max} = (V_{GS} - V_{th0}) \left(1 - \left(\frac{d_{max}}{d_0} \right)^{1/\alpha} \right). \quad (5.20)$$

Similarly, the expression of ΔV_{thc} is

$$\Delta V_{thc} = (V_{GS} - V_{th0}) \left(1 - \left(\frac{d_c}{d_0}\right)^{1/\alpha}\right). \tag{5.21}$$

Therefore, the ratio $\frac{\Delta V_{thc}}{\Delta V_{th\_max}}$ is described as follows:

$$\frac{\Delta V_{thc}}{\Delta V_{th\_max}} = \frac{1 - \left(\frac{d_c}{d_0}\right)^{1/\alpha}}{1 - \left(\frac{d_{max}}{d_0}\right)^{1/\alpha}}. \tag{5.22}$$

The maximum circuit critical path delay d_{max} that would not result in a timing error is one over the rated frequency of the processor. The critical path delay d_0 can be measured at the beginning of the processor life, and d_c can be measured at the current operation state. Both of these delays can be obtained through processor crash tests, which will be described in the following sections.

Substituting Equation 5.22 into Equation 5.17, we can solve for the lower bound of the time constant of MTTF aging:

$$\begin{aligned}\tau_{min} =& (MTTF_{constraint} - t_{stressed}) \\ & \cdot \left[-1 \cdot ln\left(1 - D + \frac{1 - \left(\frac{d_c}{d_0}\right)^{1/\alpha}}{1 - \left(\frac{d_{max}}{d_0}\right)^{1/\alpha}}\right)\right]^{-1/\beta}.\end{aligned} \tag{5.23}$$

This τ_{min}, which is later translated into the processor temperature setpoint, can be computed using user-measurable parameters. Therefore, we have derived a method of calculating processor temperature setpoint from measurement.

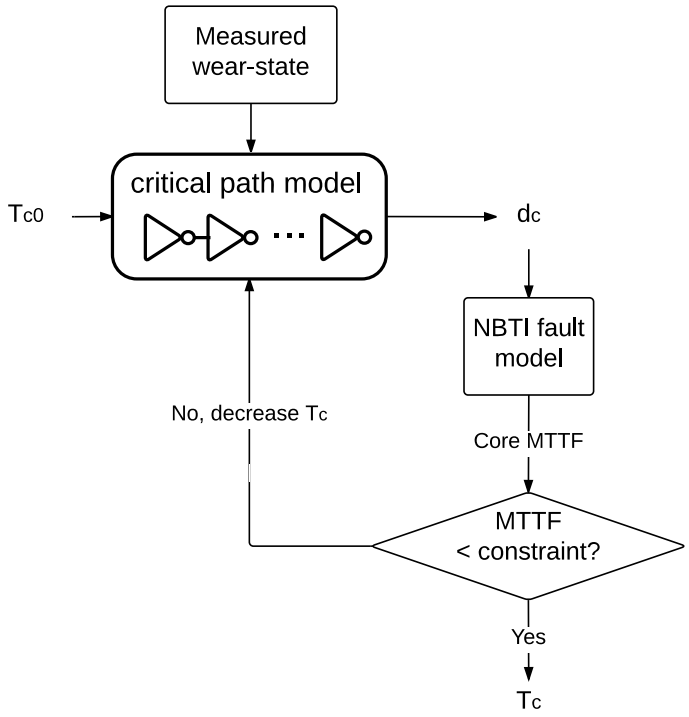


Figure 5.5: Using measured wear states and the circuit critical path model to determine the processor temperature setpoint.

5.5 Methodology and Design

In this section, we describe the design of our system for reliability aware task assignment. The system consists of two parts: automatic wear state characterization and reliability-aware task scheduling to minimize energy consumption. We first describe online measurement of processor wear state through crash testing. The wear states are used to describe the lifetime degradation of processors, and serve as guidances for reliability-aware task scheduling. We then describe the reliability-aware task scheduler.

5.5.1 Core Wear-State Test

Processor wear due to NBTI is a result of voltage increase. This increase can be measured by capturing processor timing errors. NBTI increases the transistor threshold voltage during long-term stress, slowing processors down, thereby reducing processor timing slacks. Timing

slack is the difference between processor critical path delay and clock period. If timing slack becomes zero, the processor may fail and crash, which can be observed externally. Figure 5.5 illustrates the test process. We describe the test in detail in this subsection.

A crash happens when the processor encounters a timing error. A worn-out processor crashes under normal operating conditions, while a healthy processor only crashes under extreme conditions, e.g., voltage-temperature-frequency points at which it is not rated to operate. To cause a crash for healthy processors, we stress it with a CPU-intensive workloads and run it under different frequency-voltage-temperature conditions until it crashes. The frequency-voltage-temperature points at which the processor crashes are recorded as the crash state.

The frequency at the crash state is one over the critical path delay of the processor under that specific, extreme voltage and temperature condition. Under normal conditions with lower temperature and higher voltage, the critical path delay decreases, leaving some timing slack to the processor clock cycle. This timing slack can be used to calculate the current transistor threshold voltage, which can be used to compute the processor remaining lifetime using Equation 5.17.

To compute this normal operating condition frequency from the crash condition frequency, one can use the voltage-temperature-frequency curve of the critical path of a processor. We generate this curve through SPICE simulation on a serial inverter train modeling the processor critical path, and adjust transistor sizing to match the measured crash state frequency under crash voltage and temperature condition. Therefore, the simulated frequency at normal operating voltage and temperature should represents the actual normal operating frequency. We discuss the process of building this critical path model in Subsection 5.6.2.

In a real data center, this test can be repeated periodically, e.g., every six months. We perform automatic execution of the stressmarks and data logger. The machine will shut down after a crash; it can be restarted using a remotely controlled power strip. The whole

process can be performed automatically.

5.5.2 Task Scheduler Design

We use dynamic task scheduling to achieve optimization in both computation and cooling energy consumption in data centers. The task scheduler consists of three models: the task performance and utilization model, the thermal and energy model, and the core reliability model. The details of the three models are described in Section 5.4. Using these three models, we are able to develop a dynamic task assigning procedure.

We incorporate task scheduling with reliability constraints to minimize data center energy. As briefly introduced before, the key concept of reliability-aware task assignment is to have processors run at the highest temperature that would not violate system lifetime requirements. Increasing processor operating temperatures allows the chiller to run at a higher output temperature or reduced air flow rate, thereby reducing the cooling energy consumption. The upper bounds of core temperatures are determined by core reliability constraint, as shown in Equation 5.17. This desired operating temperature for every core is achieved through dynamically scheduling tasks and adjusting chiller output temperature. The operating temperature for every core is computed separately: we consider heterogeneity caused by process variation and different wear states across different cores, and accommodate different temperature setpoints for cores in the same data center.

The task scheduler performs dynamic task assignment, i.e., assigns tasks as they arrive and adjusts assignment decisions based on previous assignments and current conditions. Assignment decisions aim to minimize total energy consumption of the data center without violating processor reliability constraints. Both the data center energy consumption and processor reliability are affected by processor temperature, which is a function of the utilization of each core. Therefore, the task scheduler should predict CPU utilizations of assigning candidate tasks, and use these values to estimate core temperatures, machine and cooling energy consumptions, and core reliabilities. Such estimations are done through task

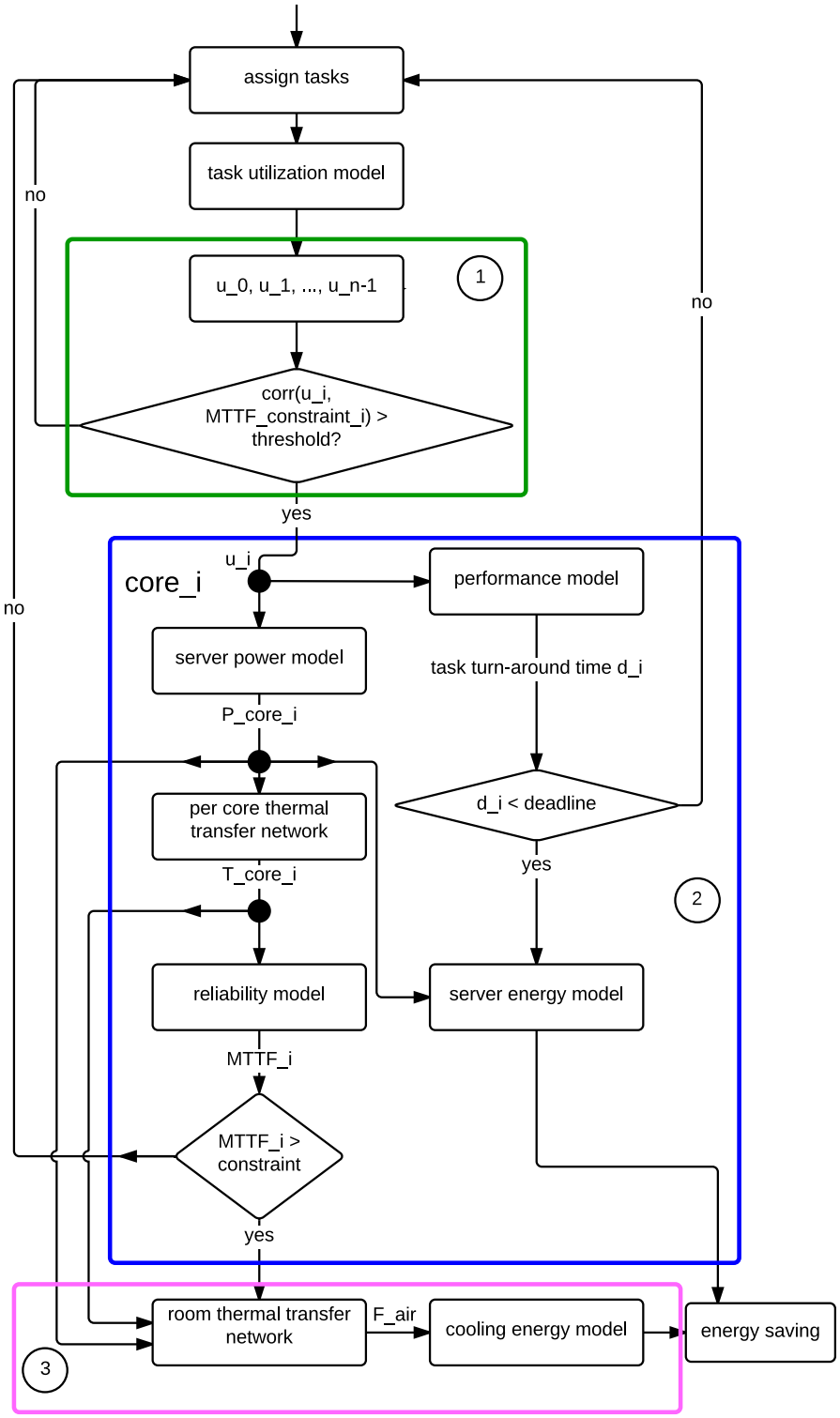


Figure 5.6: The flow chart of the task assignment algorithm. It consists of three main parts: 1. Task utilization and response time prediction ; 2. Per-core temperature, power consumption, reliability, and task response time calculation; and 3. Cooling energy calculation.

and machine modeling. Figure 5.6 demonstrates the three-step assignment process:

1. *Task pruning.* This is done by correlating the (predicted) processor resource utilization (u_0, u_1, ..., u_{n-1}) with the core MTTF requirement. We select the task assignment that results in high correlation between its CPU utilization and MTTF distribution of cores. The resource utilization of cores running existing tasks are reported by the operating system, while that of the core that will host the candidate task can be estimated using the task model. If the correlation between core resource utilization and MTTF is low, the candidate task is not suitable for assignment and will be postponed for later consideration.

2. *Temperature, power, reliability, and performance modeling.* The resource utilizations u_i calculated from the previous step are used to compute core temperature T_{core_i} and power consumption P_{core_i} using the corresponding models. The resulting temperature and power consumption are used in two models: the reliability model and the performance model. The MTTF value calculated from the reliability model should be larger than the MTTF constraint, and the predicted task response time d_i from the performance model should be shorter than the task deadline. Violating either of these constraints will result in dropping the candidate task.

3. Cooling energy modeling. As described in Equation 5.10, cooling energy consumption is a function of the temperatures and power consumptions of all cores. These values are output from the previous step and used in the cooling energy consumption model to determine chiller air flow rate F_{air}. This yields the total energy consumption of assigning the candidate task. The task resulting in the minimum total energy consumption is assigned.

5.6 Experiments

This section describe the process of processor wear state measurement and deployment of the adaptive temperature tasks scheduler.

5.6.1 Wear State Measurement

Wear state measurement captures a series of crash states of a processor. This measurement is performed under extreme conditions to cause processors to crash. The resulting crash states are used to build a circuit critical path model, which can be used to derive circuit timing slacks under normal operating conditions. This timing slack can be used to calculate the processor temperature setpoint under this wear state.

We perform the wear state measurement on a server with one 3.16 GHz, quad-core Xeon 5460 processor and 8 GB RAM. The server fans are removed from the system to allow processor temperature to increase. Installing more advanced fan control system can allow users to disable fans from the operating system, avoiding removal of processor fans. At high temperatures, processors are more likely to crash. The circuit delay increases with increasing operating temperature. Therefore, this high operating temperature increases the probability of processor timing errors, shortening the stress time required for the processor to crash.

Before knowing the current processor wear state, we don't know the DVFS state and operating temperature at which the processor will crash. Therefore, the wear state test has to iterate through several extreme operating state of the processor. The test stresses processors under different DVFS states and operating temperatures. These DVFS states can be set by programing the model-specific registers (msr) of the processor. The processor operating temperature can be adjusted by running CPU stress marks that change the power consumption of the processor. We change the combination of instructions with different power consumption (nop, memory access, branch, and computational intensive instructions)

Table 5.1: Voltage = 1.47 V, frequency = 3.16 GHz

Temperature (°C)	Crash number	Total number of measurements
95	1	6
97	6	6
99	10	10

to change the power consumption of these stress marks, resulting in varying equilibrium temperatures for the server processor.

At each DVFS state, the processor is stressed for a fixed time interval. Ideally, for a given voltage-frequency state, the core should consistently crash when its operating temperature reaches a threshold. However, noise in the system randomly changes the circuit delay, resulting in a distribution of this crash state temperature, instead of one single value. To rule out this noise from our measurement result, we assume that the circuit delay due to random noise follows Gaussian distribution. Instead of only stressing the processor at one fixed temperature point, we sweep a range of operating temperatures for the processor, and repeat the stress test for several times. We record the number of crashes under each temperature point. This crash number on different temperatures should follow Gaussian distribution as well. As a result, the crash temperature is the temperature at the 50% probability of this distribution.

We have successfully measured two different crash states on the testing server processor. These two crash states are shown in Tables 5.1 and 5.2. The temperatures at the 50% probability of these two states are 97 °C and 99 °C. Therefore, the two measured crash states are (1.47 V, 3.16 GHz, 97 °C) and (1.41 V, 2.67 GHz, 99 °C).

5.6.2 Circuit Critical Path Model

Using the measured crash states, we build a circuit model that fits the measured data and can later be used to calculate the processor temperature setpoints. The crash states indicate

Table 5.2: Voltage = 1.41 V, frequency = 2.67 GHz

Temperature (°C)	Crash number	Total number of measurements
95	0	6
97	3	6
99	10	10

the processor critical path delays under extreme conditions, e.g., high temperature and low voltage. Using this information, we calculate the processor critical path delay under normal operating conditions, and further calculating the temperature setpoint of this processor. This is done by building a processor critical path model.

We model the processor critical path with a chain of combinational logic gates. The propagation delay of this chain should equal to the measured crash delay $(1/f)$ at the processor crash temperature and operating voltage. This delay is affected by several parameters, including the logic depth, the gate type, and the sizing of each stage. Therefore, finding one single model based on the limited number of measured crash states is difficult, as there can be a large number of circuit structures that fit the measurement data. Increasing the number of measured crash states can increase the model accuracy. However, the number of crash states that can be measured by users is limited, because users can only set the processor to run at several fixed DVFS operating states, and some of these states will not result in timing errors.

Therefore, instead of building one critical path model, we select a group of circuit chains, all of which are consistent with the measured crash states, to give simulation result of the range of processor delay, and calculate the processor temperature setpoint accordingly. We use Monte-Carlo simulation to determine this group of circuit models. The simulation considers three parameters: the gate type, the number of stages, and the sizing of each gate. We increase the number of runs until adding more runs does not change the range of processor delay by more than 5%.

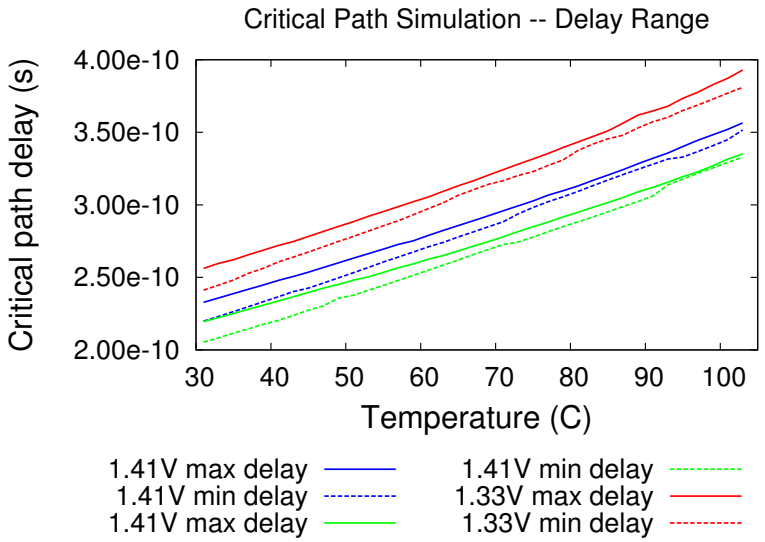


Figure 5.7: The temperature-delay relationship from the critical path model simulation.

The resulting circuit critical path model after 500 runs is shown in Figure 5.7. This model covers three operating voltage level and the temperature range that we are interested in. The processor critical path delay at each temperature and operating voltage can be determined using this model. This delay can be used in the process described in Figure 5.5 to calculate the processor temperature setpoint given the MTTF constraint.

5.6.3 Task Scheduler Deployment

We integrate the reliability model and core temperature setpoints into the reliability aware task scheduler, and deploy the scheduler on a cluster of servers to evaluate its impact on data center energy consumption.

The task scheduler is deployed on a cluster of servers. The cluster contains two racks, with eight servers on each rack. We assume that the server fan can adjust its speed continuously, i.e., it can always achieve the air flow rate required by the scheduler.

We run a group of benchmarks with different CPU intensity levels. These benchmarks consume different amounts of power and can result in different temperature distributions among server processors.

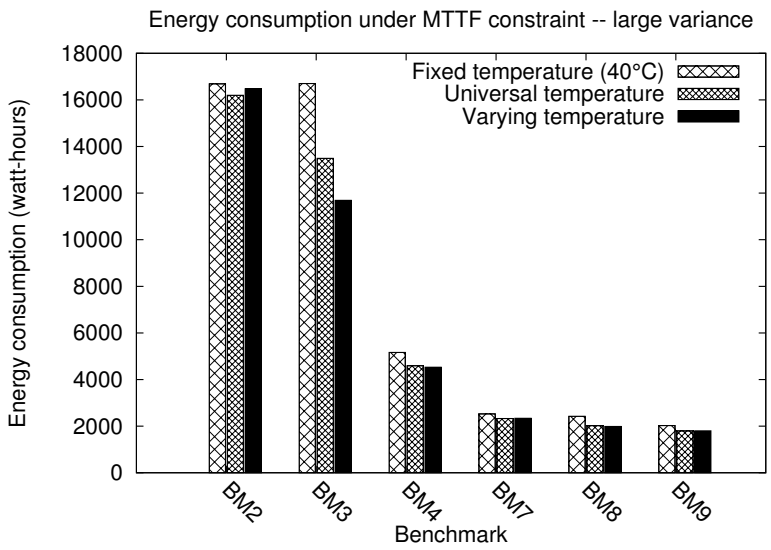


Figure 5.8: Total energy consumptions for three algorithms.

In this cluster, the wear states of processors are represented as their initial MTTF values. These MTTF values are the processor lifetimes under a certain operating temperature and voltage. They are used to describe the initial difference in failure rates of processors. The difference of these MTTF values are due to process variation. We generate these MTTF values with a Gaussian distribution with 6 year mean and 1.5 year variance. This distribution is used as initial process variation of processors [100]. As processors wear out, the mean of their distribution becomes larger and the variance becomes smaller [98, 101]. Therefore, we select the mean value after two years of usage and the initial variance as the initial distribution parameters. This distribution captures the possible MTTF values that processors can have after two years of wear. As processors further wear out, the mean and variance of this distribution can change. We evaluate these changes in our result discussion.

We compare our proposed solution with existing energy saving approaches for servers. People have designed temperature management techniques for servers, but their focus has always been setting different cores at the same target temperature setpoint [87, 92]. The difference among these existing methods lies in how close the cores can approach the target setpoints. Here we use this approach as our baseline for comparison, and assume

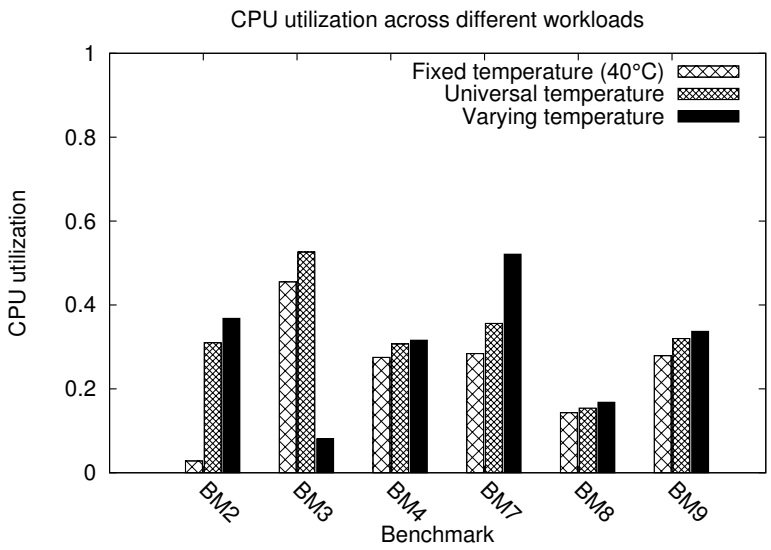


Figure 5.9: CPU utilization of different benchmarks.

that processors can reach their temperature setpoint with little performance and energy overhead. In the deployed experiment, we compare among three different methods of using the processor temperature setpoints.

- *The Fixed Temperature method,* in which every core targets the same fixed temperature setpoint. This setpoint is specified by the data center manager, and therefore does not require the wear state test. This method does not guarantee the MTTF constraint to be satisfied in every core. This is the simplest algorithm and is the one currently used in most commercial data centers.

- *The Universal Temperature method,* in which every core targets the lowest temperature setpoint of all cores. This method requires a wear state test of every core, but does not require the iterative method of computing th suitable fan speed for every core. This method guarantees the MTTF constraint would be satisfied by all cores, but has the potential of wasting cooling energy on the most reliable cores.

- *The Varying Temperature method,* which sets every core to run at its own temperature setpoint based on its individual wear state measurement result. This method satisfies

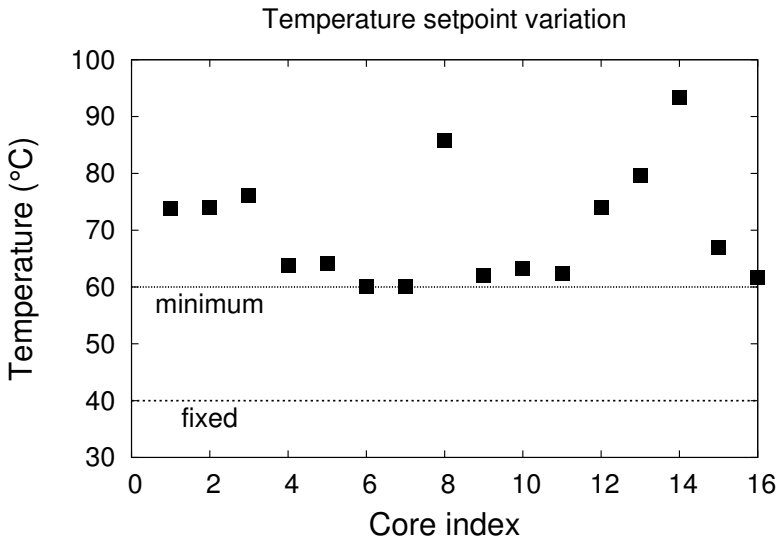


Figure 5.10: The temperature setpoints of the three algorithms. The dots are temperature setpoints of different cores in the Varying Temperature method. The "minimum" line is the temperature setpoint of the Universal Temperature method, and the "fixed" line is the temperature setpoint of the Fixed Temperature method.

the reliability constraint for all cores while reducing cooling energy waste.

We use the total data center energy consumption as a metric to evaluate the above three algorithms. Figure 5.8 demonstrates the energy consumption of the three algorithms across a group of benchmarks. Comparing to the baseline (the Fixed Temperature method), the Universal Temperature setpoint algorithm saves 11.5% energy on average, and the Varying Temperature setpoint method saves 13.3% energy. Different benchmarks having different energy savings because these benchmarks vary in CPU utilizations, as demonstrated in Figure 5.9. Some benchmarks cause CPUs to run at hither loadings to reach their core temperature setpoints. Some other benchmarks constrain their CPU utilization to lower level to prevent resource contention, so that workloads can finish in shorter times without increasing in server energy consumption. The Varying Temperature method captures both properties and therefore saves more energy than the other two algorithms.

The Varying Temperature method saves more energy than the other two methods because it does not run processors at unnecessarily low temperature setpoints. The temperature

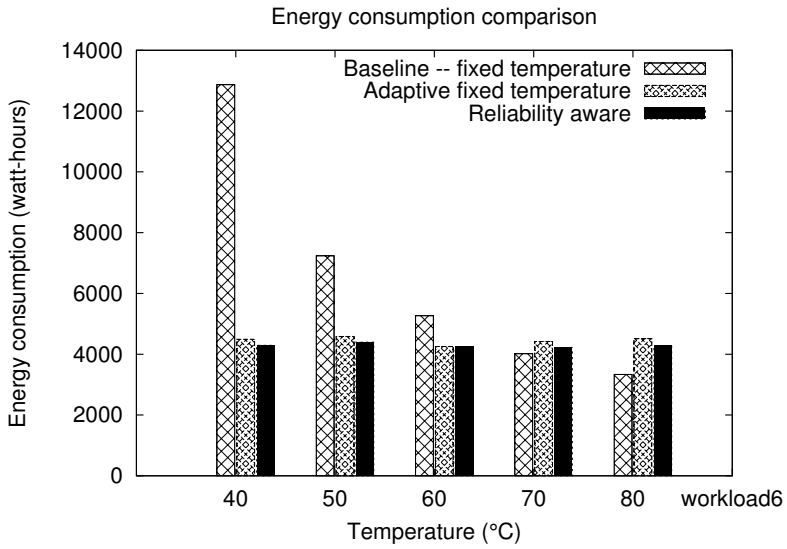


Figure 5.11: Energy consumption comparison of three algorithms when the fixed operating temperature changes. This affects the energy consumption of the Fixed temperature method, but has little influence on the other two.

setpoints of the above three algorithms are shown in Figure 5.10. The varying temperature reduces cooling energy waste in the data center. On the contrary, the Universal Temperature method sets every core to run at the lowest temperature setpoint of all cores, over-cooling some more reliable cores which should have been able to run at higher temperatures. The Fixed Temperature method runs all cores at an unnecessarily low temperature, burning cooling energy.

In the above evaluation, the temperature setpoint of the Fixed Temperature method is set to 40 °C. As described earlier, data center total energy consumption decreases with processor temperature setpoint. Therefore, increasing the temperature setpoint can potentially benefit the Fixed Temperature method. The Fixed Temperature method can achieve reasonable energy saving if its temperature setpoint is close to the average of the temperature setpoints of all cores. However, this can result in MTTF constraint violations for some less reliable cores. Figure 5.11 demonstrates the change in energy consumption of the three algorithms when the fixed operating temperature increases from 40 °C to 80 °C. At low temperatures, the energy consumption of the Fixed Temperature method is higher than the other two. At

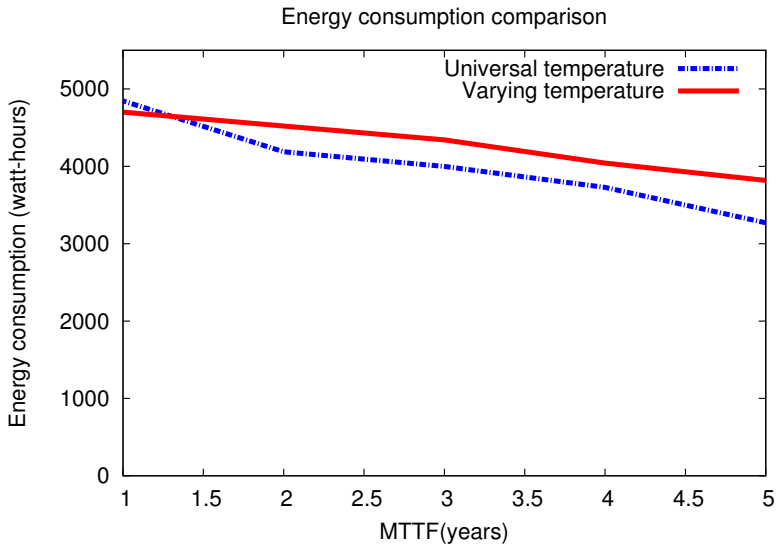


Figure 5.12: Total energy consumption of the cluster when the mean of core MTTF changes.

higher temperatures, the energy consumption of the Fixed temperature method decreases, saving more energy compared to the other two methods. However, this high operating temperature has already violate the MTTF constraint for some cores.

5.6.4 Result Discussions

Above experiments assume that wear states of processors follows the same distribution. This is under the assumption that processors are of the same type, and the difference in wear state is mainly caused by process variation and difference in processor aging. However, this distribution can change in real data centers. A data center can have both new and old processors; it can also have processors from different types or makers. This different distributions of MTTF among cores can impact energy saving results. We discuss this impact and provide guidance for data center managers to reduce energy and maintenance cost of servers. We start with the Gaussian distribution and evaluate the impact of changing the mean and variance of the distribution.

First, Figure 5.12 demonstrates the relationship between the mean of core MTTF distri-

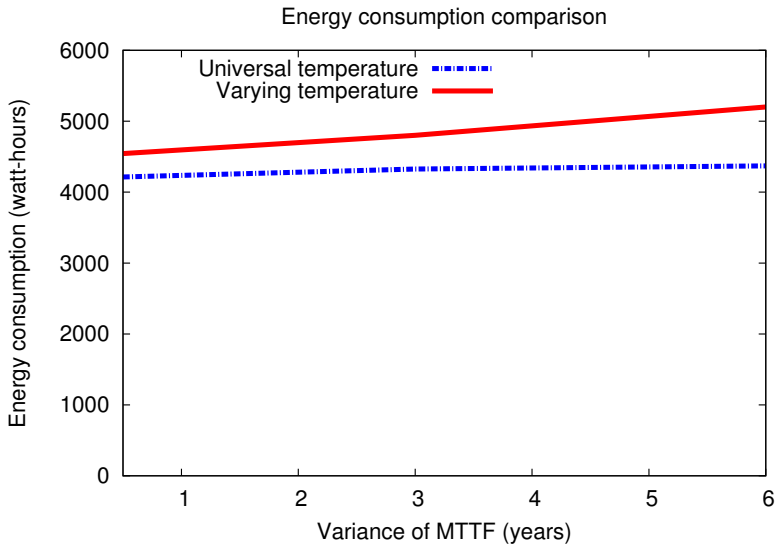


Figure 5.13: Total energy consumption of the cluster when the variance of core MTTF changes.

bution and server energy consumption. The mean increases from one year to five years. The energy consumption of both the Universal Temperature and Fixed Temperature methods decreases with the mean of core MTTF, i.e., when cores becomes more reliable. The reason is straightforward: higher core MTTF means that cores are more reliable, allowing the temperature setpoint to be reached with the given MTTF constraint, reducing cooling energy consumption.

Second, Figure 5.13 shows the impact of increasing the variance of core MTTF distribution on server energy consumption. When the variance increases from 0.5 year to 6 years while the mean stays unchanged, the energy consumption of the Universal Temperature method increases accordingly. This is because the temperature setpoint in this method is constrained by the least reliable core in the system. Therefore, larger variance results in a smaller MTTF for the least reliable core and a lower universal temperature setpoint, increasing the cooling energy waste in other more reliable cores. As a contrary, the energy consumption of the Varying Temperature method increases only slightly with increasing variance of core MTTF distribution, because this method guarantees that every core runs at

its optimal temperature setpoint. These setpoints are calculated separately based on the wear state of each core. Therefore the overall energy consumption, which is the sum of cooling energies for all processors, is affected little by the variance of the core MTTF distributions.

As a result, when the variance of core MTTF distribution of the cores is small, the above two methods result in little difference in energy usage. Each consumes higher energy when the mean of core MTTF distribution increases. When the variance of core MTTFs is large, the Varying Temperature method saves more energy.

Finally, we present an alternative way for the Universal Temperature method to achieve decent energy savings when the core MTTF variance is large. This method can save additional energy by replacing one or several of the least reliable cores with new ones. As shown in Figure 5.10, two unreliable cores lower the temperature setpoint of the Universal Temperature algorithms. Replacing these cores with reliable ones can increase the temperature setpoint, which reduces cooling energy consumption. Therefore, the total cost of Universal Temperature algorithm is the above energy savings, plus the cost of core replacement. We perform Monte-Carlo simulation of 100 runs on core wear states following a distribution of 4 years of mean and 1.5 years of variance, and calculated the energy saving produced by replacing the least reliable core(s). The average energy savings is 5.4% and the maximum is 16%. As a result, the Universal Temperature algorithm avoids frequent crash state test and can save a lot of energy, but may require core replacement.

5.7 Conclusion

We have described a reliability aware task scheduling algorithm that reduces cooling energy consumptions of servers. This method reduces cooling energy consumption by running each core at its optimal temperature setpoint. We develop two components for this scheduler: an automatic processor crash test method, which measures core wear state and calculates the corresponding temperature setpoint of processor, and a reliability aware task scheduling

algorithm, which assign tasks to processors to reach their desired temperature setpoints. The scheduler reduces energy consumption by 13% on average, compared with a technique using reliability unaware temperature setpoints. We also evaluated the scheduler in the presence of varying core wear state distributions and pointed out the most appropriate assignment algorithm in each case.

CHAPTER 6

Conclusion and Future Works

This thesis has discussed several problems and solutions related to matching resources and workloads. Our designs allow systems to remain functional given limited energy sources, or improve system throughput and reduce energy use given user-specified reliability constraint. This thesis focuses on two types of systems.

- **Embedded systems.** We have developed two methods to explore designs for systems using restricted or unreliable energy sources for embedded applications. First, we developed a power deregulation method that allows an embedded system to operate on unregulated batteries. We then designed a process for application-based battery and processor selection, to extend system lifespan in deregulated systems. Second, we developed the AEA (Ambient Energy Aware) routing protocol for batteryless sensor networks operating on scavenged energy sources. The AEA protocol matches spatially- and temporally-correlated energy sources with workloads in the system to provide successful data transmission.

- **Data centers.** We have also optimized workload and resource matching in data centers, with the goal of improving performance and saving energy. We have developed HAMS (Heterogeneous Adaptive Modeling Scheduler), a task scheduler that optimizes workload allocation based on available computational resources. To optimize scheduling results, we developed task performance models that consider machine resource utilization and heterogeneity of machine and tasks. These models

predict task resource utilizations and execution times. HAMS improves overall data center throughput and reduces computational energy consumption via task concentration. In addition, we reduced the data center cooling energy by determining the optimal processor temperature setpoints. To calculate these setpoints, we developed an automatic crash test method to measure the wear states of processors. We also provided a reliability model and an energy model for data cetners. We incorporated the processor temperature setpoint, the reliability model, and the energy model into HAMS to minimize data center computational and cooling energy.

This thesis has focused on the software side of system design. Fine-grained modeling of the physical system has the potential to further improve the reported results. Researchers who are interested in this area should consider the following directions.

- **Building more sophisticated and customized thermal models for data centers.** Currently we use an analytical thermal model for the server room cooling system. However, this model only considers a single air inlet in the server room. Developing a more accurate data center thermal model can improve the energy consumption of reliability-aware task scheduling. Researchers can use Computational Fluid Dynamics (CFD) tools to simulate air flows in the server room and derive the relationship among heat transfer, air flow rate, and temperature.

- **Improving the processor critical path model.** An accurate critical path model should select the correct number of combinational logic stages, the types of gates, and the optimal gate sizings. In this study, we used Monte Carlo simulation to identify the group of models that fit the measurement data. However, more thorough analysis should be conducted to select more reasonable parameter values, and further increase the accuracy of the modeled server temperature setpoint results.

BIBLIOGRAPHY

[1] "America's data centers consuming and wasting growing amounts of energy."

[2] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Mar. 2009, pp. 205–216.

[3] "Health concerns with batteries," http://batteryuniversity.com/learn/article/health_concerns.

[4] W. Lang and J. M. Patel, "Energy management for MapReduce clusters," *Proc. Very Large Database Endowment*, vol. 3, no. 1-2, pp. 129–139, Sept. 2010.

[5] B. Arbetter, R. Erickson, and D. Maksimovic, "DC-DC converter design for battery-operated systems," in *Proc. Conf. Power Electronics Specialists*, vol. 1, June 1995, pp. 103–109.

[6] "LM78XX/LM78XXA 3-terminal 1 A positive voltage regulator," Fairchild Semiconductor, https://www.fairchildsemi.com/datasheets/LM/LM7805.pdf.

[7] "iPod Nano 6th generation teardown," iFixit, https://www.ifixit.com/Teardown/iPod+Nano+6th+Generation+Teardown/3563.

[8] D. Linden, *Handbook of Batteries*, 3rd ed. New York: McGraw-Hill, 2002.

[9] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proc. Int. Conf. Information Processing in Sensor Networks*, Apr. 2005.

[10] S. Kim, R. P. Dick, and R. Joseph, "Power deregulation: eliminating off-chip voltage regulation circuitry from embedded systems," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2007, pp. 105–110.

[11] J. Cao and A. Emadi, "A new battery/UltraCapacitor hybrid energy storage system for electric, hybrid, and plug-in hybrid electric vehicles," *IEEE Trans. Power Electronics*, vol. 27, no. 1, pp. 122–132, Jan. 2012.

[12] Y. Cho, Y. Kim, Y. Joo, K. Lee, and N. Chang, "Simultaneous optimization of battery-aware voltage regulator scheduling with dynamic voltage and frequency scaling," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2008, pp. 309–314.

[13] Y. Choi, N. Chang, and T. Kim, "DC-DC converter-aware power management for low-power embedded systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 8, pp. 1367–1381, Aug. 2007.

[14] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *Proc. Design, Automation & Test in Europe Conf.*, 1999, pp. 861–866.

[15] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE J. Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, Apr. 1990.

[16] M. Chen and G. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and I-V performance," *IEEE Trans. Energy Conversion*, vol. 21, no. 2, pp. 504–511, June 2006.

[17] V. Rao, G. Singhal, A. Kumar, and N. Navet, "Battery model for embedded systems," in *Proc. Int. Conf. VLSI Design*, Jan. 2005, pp. 105–110.

[18] R. Lawrence, S. M. Manager, Battery, Megger, G. Esmet, P. Merl, and J. Heynecke, "The virtues of impedance testing of batteries," in *Proc. Int. Conf. Stationary Battery*, 2003.

[19] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. VLSI Systems*, vol. 8, no. 3, pp. 299–316, June 2000.

[20] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "A discrete-time battery model for high-level power estimation," in *Proc. Design, Automation & Test in Europe Conf.*, 2000, pp. 35–41.

[21] I. Buchmann, "Does internal resistance reveal battery capacity? A study on rapid-test methods for stationary and automotive batteries," Cadex Electronics Inc., June 2004.

[22] E. Bonizzoni, F. Borghetti, P. Malcovati, F. Maloberti, and B. Niessen, "A 200 mA 93% peak efficiency single-inductor dual-output DC-DC buck converter," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2007, pp. 526–619.

[23] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the internet of things: A survey," in *Proc. Software, Telecommunications and Computer Networks*, sep 2011, pp. 1–6.

[24] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921–960, 2007.

[25] "Linear regulators in portable applications," Maxim, July 2002, http://pdfserv.maxim-ic.com/en/an/AN751.pdf.

[26] *DC-DC Converters: A Primer*, Jaycar Electronics, 2001, http://www.jaycar.com.au/images_uploaded/dcdcconv.pdf.

[27] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2010, pp. 105–114.

[28] J. Blochwitz-Nimoth, M. Pfeiffer, X. Zhou, J. Huang, G. He, and K. Leo, "Highly efficient and low-operating-voltage OLEDs for active and passive matrix displays," *Organic Light-Emitting Materials and Devices VII*, vol. 5214, no. 1, pp. 172–179, 2004.

[29] H. Wu, et al., "Stable cycling of double-walled silicon nanotube battery anodes through solid-electrolyte interphase control," *Nature nanotechnology*, vol. 7, no. 5, pp. 310–315, 2012.

[30] A. R. Armstrong, C. Lyness, P. M. Panchmatia, M. S. Islam, and P. G. Bruce, "The lithium intercalation process in the low-voltage lithium battery anode $Li_{1+x}V_{1--x}O_2$," *Nature Materials*, vol. 10, no. 3, pp. 223–229, 2011.

[31] D. Berndt., *Maintenance-free batteries : lead-acid, nickel/cadmium, nickel/metal hydride : a handbook of battery technology*, 2nd ed. Somerset, England : Research Studies Press, 1997.

[32] M. Minami, T. Morito, and H. Morikawa, "Solar biscuit: a battery-less wireless sensor network system for environmental monitoring applications," in *Proc. 9th Int. Conf. on Networked Sensing Systems*, 2005.

[33] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. Int. Wkshp. Wireless Sensor Networks and Applications*, Sept. 2002, pp. 88–97.

[34] E.-H. El Brouji, O. Briat, J.-M. Vinassa, N. Bertrand, and E. Woirgard, "Impact of calendar life and cycling ageing on supercapacitor performance," *IEEE Trans. Vehicular Technology*, vol. 58, no. 8, pp. 3917–3929, Oct. 2009.

[35] "Using a small solar cell and a supercapacitor in a wireless sensor," http://www.sensorsmag.com.

[36] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Proc. Conf. on Information Processing in Sensor Networks*, 2005.

[37] J. Taneja, J. Jeong, and D. Culler, "Design, modeling, and capacity planning for micro-solar power sensor networks," in *Proc. Int. Conf. Information Processing in Sensor Networks*, Apr. 2008, pp. 407–418.

[38] T. Voigt, H. Ritter, and J. Schiller, "Utilizing solar power in wireless sensor networks," in *Proc. Int. Conf. Local Computer Networks*, Oct. 2003, pp. 416–422.

[39] "SMART modular technologies," http://www.smartm.com.

[40] J. Schiffer, D. Linzen, and D. U. Sauer, "Heat generation in double layer capacitors," *J. Power Sources*, vol. 160, no. 1, pp. 765–772, 2006.

[41] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proc. Int. Conf. Information Processing in Sensor Networks*, Apr. 2005, pp. 463–468.

[42] M. Philipose, J. Smith, B. Jiang, A. Mamishev, S. Roy, and K. Sundara-Rajan, "Battery-free wireless identification and sensing," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 37–45, Jan. 2005.

[43] R. Vyas, V. Lakafosis, Z. Konstas, and M. Tentzeris, "Design and characterization of a novel battery-less, solar powered wireless tag for enhanced-range remote tracking applications," in *Microwave Conference, European*, Oct. 2009, pp. 169–172.

[44] A. Patel, M. Vaghela, H. Bajwa, and P. Patra, "Power harvesting for low power wireless sensor network," in *Antennas Propagation Conference, 2009. LAPC 2009. Loughborough*, Nov. 2009, pp. 633–636.

[45] J. W. Ng, B. P. Lo, O. Wells, M. Sloman, N. Peters, A. Darzi, C. Toumazou, and G.-Z. Yang, "Ubiquitous monitoring environment for wearable and implantable sensors (ubimon)," in *Proc. 6th Int. Conf. on Ubiquitous Computing*, Sept. 2004.

[46] R. Morais, S. G. Matos, M. A. Fernandes, A. L. G. Valente, S. F. S. P. Soares, P. J. S. G. Ferreira, and M. J. C. S. Reis, "Sun, wind and water flow as energy supply for small stationary data acquisition platforms," *Computers and Electronics in Agriculture*, vol. 64, pp. 120–132, Dec. 2008.

[47] F. Zhao and L. J. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, 2004.

[48] "World wind maps – wind atlases of the world," http://www.mywindpowersystem.com.

[49] C. L. Archer and M. Z. Jacobson, "The spatial and temporal distribution of U.S. winds and windpower at 80 m derived from measurements," *American Geophysical Union*, p. C82, Dec. 2002.

[50] S. Chan, D. Powell, M. Yoshimura, and D. Curtice, "Operations requirements of utilites with wind power generation," *IEEE Trans. Power Apparatus and Systems*, vol. PAS-102, no. 9, pp. 2850–2860, Sept. 1983.

[51] I. Damousis, M. Alexiadis, J. Theocharis, and P. Dokopoulos, "A fuzzy model for wind speed prediction and power generation in wind parks using spatial correlation," *IEEE Trans. Energy Conversion*, vol. 19, no. 2, pp. 352–361, June 2004.

[52] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, Mar. 2006.

[53] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proc. Int. Symp. Operating Systems Design and Implementation*, Nov. 2006, pp. 381–396.

[54] K. Martinez, R. Ong, and J. Hart, "Glacsweb: a sensor network for hostile environments," in *Proc. Conf. Sensor and Ad Hoc Communications and Networks*, Oct. 2004, pp. 81–87.

[55] J. Lundquist, D. Cayan, and M. Dettinger, "Meteorology and hydrology in Yosemite national park: A sensor network application," in *Proc. Int. Conf. Information Processing in Sensor Networks*, Apr. 2003, pp. 553–553.

[56] S. N. Simic and S. Sastry, "Distributed environmental monitoring using random sensor networks," in *Proc. Int. Conf. Information Processing in Sensor Networks*, Apr. 2003, pp. 582–592.

[57] "Center of costal margin observation and prediction," http://www.ccalmr.ogi.edu/CORIE/.

[58] M. Shinozuka, P. H. Chou, S. Kim, H. R. Kim, E. Yoon, H. Mustafa, D. Karmakar, and S. Pul, "Nondestructive monitoring of a pipe network using a MEMS-based wireless network," in *Proc. SPIE*, Mar. 2010, p. 7649.

[59] "Wind energy resource atlas of the United States," http://rredc.nrel.gov/wind/pubs/atlas/atlas_index.html.

[60] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Proc. Wkshp. on Compilers and Operating Systems for Low Power*, Sept. 2001.

[61] C. Mastroianni, M. Meo, and G. Papuzzo, "Multi-resource workload consolidation in cloud data centers," in *Proc. Int. Conf. Utility and Cloud Computing*, 2013, pp. 297–298.

[62] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel & Distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.

[63] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proc. USENIX Conf. Networked Systems Design and Implementation*, Mar. 2011, pp. 22–22.

[64] L. Barroso and U. Holzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[65] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proc. Int. Symp. Computer Architecture*, June 2011, pp. 319–330.

[66] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proc. USENIX Conf.*, June 2011, pp. 12–12.

[67] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proc. Wkshp. Power Aware Computing and Systems*, Dec. 2008.

[68] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of Hadoop clusters," *Proc. Special Interest Group on Operating Systems European Wkshp.*, vol. 44, no. 1, pp. 61–65, Mar. 2010.

[69] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. European Conf. on Computer Systems*, Apr. 2010, pp. 265–278.

[70] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. Symp. Operating Systems Principles*, Nov. 2009, pp. 261–276.

[71] "Introducing data center fabric, the next-generation facebook data center network," code.facebook.com/posts/360346274145943/ introducing-data-center-fabric-the-next-generation-facebook-data-center-network.

[72] C. K. D. Economou, S. Rivoire and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Wkshp. on Modeling Benchmarking and Simulation*, June 2006.

[73] "M610 vs. HP BL460c: Full enclosure SPECpower_ssj2008 testing," http://www.dell.com/downloads/global/products/pedge/en/poweredge-m610-power-competitive-whitpaper.pdf.

[74] "Emulab network testbed," www.emulab.net.

[75] X. Zhang, Z. Zhong, S. Feng, B. Tu, and J. Fan, "Improving data locality of mapreduce by scheduling in homogeneous computing environments," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, 2011, pp. 120–126.

[76] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for mapreduce," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 2012, pp. 435–442.

[77] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010, pp. 1–9.

[78] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 2011, pp. 1199–1208.

[79] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 17–24.

[80] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, p. 58.

[81] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *Proc. Int. Symp. Computer Architecture*, June 2011, pp. 283–294.

[82] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Proc. Int. Conf. Data Engineering Wkshps.*, Mar. 2010, pp. 87–92.

[83] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Mar. 2013, pp. 77–88.

[84] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. Int. Symp. Operating Systems Design and Implementation*, Dec. 2008, pp. 29–42.

[85] "Synjet: Low-power "green" cooling," Nuventix, Inc., =http://www.digikey.pt/Web%20Export/Supplier%20Content/nuventix-1061/pdf/nuventix-white-paper-synjet-green-cooling.pdf.

[86] M. Patterson, "The effect of data center temperature on energy efficiency," in *Proc. Intersociety Conference Thermal and Thermomechanical Phenomena in Electronic Systems*, May 2008, pp. 1167–1174.

[87] R. Ayoub and T. Rosing, "Cool and save: Cooling aware dynamic workload scheduling in multi-socket cpu systems," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2010, pp. 891–896.

[88] W. Huang, M. Allen-Ware, J. Carter, E. Elnozahy, H. Hamann, T. Keller, C. Lefurgy, J. Li, K. Rajamani, and J. Rubio, "TAPO: Thermal-aware power optimization techniques for servers and data centers," in *Proc. Green Computing Conf. and Wkshp. (IGCC), 2011 Int.*, July 2011, pp. 1–8.

[89] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 193–204.

[90] M. Basoglu, M. Orshansky, and M. Erez, "NBTI-aware dvfs: A new approach to saving energy and increasing processor lifetime," in *Proc. Int. Symp. Low Power Electronics & Design*, 2010, pp. 253–258.

[91] J. Sun, A. Kodi, A. Louri, and J. Wang, "NBTI aware workload balancing in multi-core systems," in *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, Mar. 2009, pp. 833–838.

[92] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. Design, Automation & Test in Europe Conf.*, Apr. 2007, pp. 1659–1664.

[93] R. Das, J. O. Kephart, J. Lenchner, and H. Hamann, "Utility-function-driven energy-efficient cooling in data centers," in *Proc. of the 7th Int. Conf. on Autonomic Computing*, 2010, pp. 61–70.

[94] J. Chen, R. Tan, G. Xing, and X. Wang, "PTEC: A system for predictive thermal and energy control in data centers," in *Proc. Real-Time Systems Symp.*, Dec. 2014, pp. 218–227.

[95] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini, "Mercury and freon: Temperature emulation and management for server systems," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 106–116.

[96] T. Breen, E. Walsh, J. Punch, A. Shah, and C. Bash, "From chip to cooling tower data center modeling: Part i influence of server inlet temperature and temperature rise across cabinet," in *Proc. Intersociety Conference Thermal and Thermomechanical Phenomena in Electronic Systems*, June 2010, pp. 1–10.

[97] E. J. Walsh, T. J. Breen, J. Punch, A. J. Shah, and C. E. Bash, "From chip to cooling tower data center modeling: Part ii influence of chip temperature control philosophy," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, 2010, pp. 1–7.

[98] W. Wang, V. Reddy, B. Yang, V. Balakrishnan, S. Krishnan, and Y. Cao, "Statistical prediction of circuit aging under process variations," in *Proc. Custom Integrated Circuits Conf.*, 2008, pp. 13–16.

[99] S. Zafar, "Statistical mechanics based model for negative bias temperature instability induced degradation," *J. Applied Physics*, vol. 97, no. 10, p. 3709, 2005.

[100] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Design Automation Conf.*, 2003, pp. 338–342.

[101] K. Kang, S. P. Park, K. Roy, M. Alam, et al., “Estimation of statistical variation in temporal nbti degradation and its impact on lifetime circuit performance,” in *Proc. Int. Conf. Computer-Aided Design*, 2007, pp. 730–734.