

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Context-Aware Platform Design and Optimization

Permalink

<https://escholarship.org/uc/item/9h298051>

Author

Chan, Christine S.

Publication Date

2017

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Context-Aware Platform Design and Optimization

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Christine S. Chan

Committee in charge:

Professor Tajana Šimunić Rosing, Chair
Professor Sujit Dey
Professor Ryan Kastner
Professor Farinaz Koushanfar
Professor Steven Swanson

2017

Copyright
Christine S. Chan, 2017
All rights reserved.

The dissertation of Christine S. Chan is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2017

DEDICATION

For those who struggle on.

EPIGRAPH

For I resolved to know nothing while I was with you
except Jesus Christ and Him crucified.

– *1 Corinthians 2:2 (NIV)*

TABLE OF CONTENTS

Signature Page		iii
Dedication		iv
Epigraph		v
Table of Contents		vi
List of Figures		ix
List of Tables		xi
Acknowledgments		xii
Vita		xv
Abstract of the Dissertation		xvii
Chapter 1	Introduction	1
	1.1 Hierarchical context processing middleware	2
	1.2 Device context - optimizing a known, closed system . .	3
	1.3 Environmental context - adapting to an unknown world	4
	1.4 Thesis contributions	5
Chapter 2	The Context Engine	9
	2.1 Background and related work	10
	2.1.1 Use of machine learning	11
	2.1.2 Context Ontologies	11
	2.1.3 IoT Software Infrastructure and Middleware . .	13
	2.2 Hierarchical context modeling	14
	2.2.1 Generalized Data Transformation	17
	2.2.2 Integration with Ontologies	19
	2.3 Theoretical analysis	20
	2.3.1 Complexity	20
	2.3.2 Accuracy	22
	2.3.3 Scalability	24
	2.4 Case study: Fitness tracker	26
	2.4.1 Input sources	27
	2.4.2 Context engine organization	28
	2.4.3 Results	29
	2.5 Conclusion	32

Chapter 3	Device context: performance, power and thermal controls . . .	34
3.1	Background and related work	36
3.1.1	Database Workload Modeling	37
3.1.2	Mechanical considerations: shock and vibration	37
3.1.3	Power, Thermal and Cooling Management	38
3.2	Measuring cooling and performance interactions	40
3.2.1	Measurement methodology	40
3.2.2	Amplitude test with random frequencies	43
3.2.3	Frequency test with fixed amplitude	44
3.2.4	Fan sweep test	44
3.3	Integrated system model for performance, power, thermal and cooling	46
3.3.1	Workload representation	47
3.3.2	Power model	50
3.3.3	Thermal model	51
3.3.4	Cooling-vs-disk interaction model	52
3.3.5	Combining model dynamics	54
3.4	Energy-optimal cooling control	55
3.4.1	Convex optimal formulation	57
3.4.2	Optimal algorithm design	59
3.5	Results	61
3.5.1	Experimental Setup	61
3.5.2	Controller policy results comparison	64
3.5.3	Sensitivity to model inaccuracy	66
3.5.4	Overhead	69
3.6	Conclusion	71
Chapter 4	Environmental context: Air quality and smart health	73
4.1	Background and Related Work	74
4.2	A modular context-sensing platform	75
4.2.1	Sensors	76
4.2.2	Processing and communication	79
4.2.3	Power	81
4.3	Context-aware Smart Health application	83
4.3.1	Population health	84
4.3.2	Personal health	85
4.4	System evaluation and results	87
4.4.1	Smart Health Experimental Setup	87
4.4.2	Individual devices	88
4.4.3	Ecosystem utilization	88
4.5	Conclusion	94

Chapter 5	Summary and Future Work	96
5.1	Thesis Summary	96
5.1.1	Hierarchical context processing middleware . . .	96
5.1.2	Device context optimization	97
5.1.3	Environmental context in a hierarchy	97
5.2	Future Work	98
Bibliography	100

LIST OF FIGURES

Figure 2.1:	Traditional sensing application development serving multiple end-user applications	16
Figure 2.2:	A hierarchy of <i>context engines</i> sharing intermediate data processed from various sensors to serve multiple end-user applications	17
Figure 2.3:	Ontology specification for GPS data, with coordinates, source, and range.	19
Figure 2.4:	Breakdown of a single-step into lower-complexity equivalent reductions, with the minimum complexity occurring with maximum division (two-input engines on the right).	22
Figure 2.5:	Functionally equivalent organizations of the single-stage application (top) and the sequential (bottom).	23
Figure 2.6:	Scalability comparison between the single-stage approach and the context engine, comparing complexity with input size (left) and communication overhead over input size (right).	26
Figure 2.7:	Relevant variables filtered from the context space into an application’s context dimensions	28
Figure 2.8:	Bayesian Network to learn accurate step count from Fitbit and Moves data	30
Figure 2.9:	Bayesian Network to learn accurate step count from Fitbit and Moves data	31
Figure 3.1:	Server organization with (1) hard disks and (2) fan assembly directing airflow towards (3) the motherboard.	41
Figure 3.2:	Throughput dependence on vibrational amplitude, component frequencies ranging 20-800Hz	43
Figure 3.3:	Throughput dependence on vibrational amplitude, component frequencies ranging 20-2000Hz	44
Figure 3.4:	Throughput dependence on vibrational frequency with amplitude fixed at 0.17g	45
Figure 3.5:	Throughput dependence on fan speeds (no external vibrations)	46
Figure 3.6:	Subsystems and dependencies in the server model	47
Figure 3.7:	States as defined from CPU and IO bandwidth utilization vector clustering	49
Figure 3.8:	Model representation of query 2, with a chain of 6 states . . .	50
Figure 3.9:	Fan speed-disk throughput interaction fit to a sigmoid function	53
Figure 3.10:	Numerical functions of temperature and derivatives, with respect to fan speeds	55
Figure 3.11:	Simulation setup with sub-models.	62
Figure 3.12:	Final results of various algorithms on selected TPC-H queries	65
Figure 3.13:	Efficacy of the optimal solver with inaccurate power modeling	67

Figure 3.14: Efficacy of the optimal solver with inaccurate temperature prediction	68
Figure 3.15: Alternative disk delay models used by the optimal solver	69
Figure 3.16: Efficacy of the optimal solver with inaccurate delay models, compared against default “exponential” function	70
Figure 4.1: The AQ board with electrochemical sensors, supporting humidity and pressure sensors (left), and with VOC sensors, Bluetooth module and alternative processor (right).	76
Figure 4.2: A workflow for building our personalized health application based on population health data	84
Figure 4.3: For different configurations: Total execution time for anomaly detection and health risk model on different nodes. Note that y-axis is in log scale.	89
Figure 4.4: For different configurations/data sizes: Total energy consumption for anomaly detection and health risk model on different nodes. Note that y-axis is in log scale.	90
Figure 4.5: The total daily energy consumption of nodes in the ecosystem, for different task allocations across nodes, and different levels of variation in actual air quality sampled	92
Figure 4.6: Sensor node battery lifetime. Note that y-axis is in log scale.	93
Figure 4.7: Gateway (Raspberry Pi) battery lifetime. Systems 3 and 5 do not utilize Raspberry Pis and are omitted.	94
Figure 5.1: Proposed internal structure of a context engine. Current iteration does not provide privacy mechanisms.	98

LIST OF TABLES

Table 3.1:	Test server specifications	41
Table 3.2:	Disk drive models specifications	42
Table 3.3:	Overview of measured disk drive behavior	42
Table 3.4:	Comparison to related fan control strategies	63
Table 3.5:	Behavior of fan control strategies	66
Table 3.6:	Overhead of fan control strategies	70
Table 4.1:	Sensing elements of the air quality sensing platform, including built-in and optional extensions.	77
Table 4.2:	Power consumption in various configurations. “Env” refers to the set of basic environmental sensors (humidity, temperature, and pressure) and “AQI” refers to the pollutant sensors.	82
Table 4.3:	Three devices representing a range of capabilities in the cloud .	88
Table 4.4:	Task distributions to context engines residing on each node . .	91

ACKNOWLEDGMENTS

First, I would like to thank my advisor Tajana Šimunić Rosing, and all my committee members. They guided me in finding critical research directions, taught me how to communicate effectively and with integrity, and provided many opportunities to work with researchers from all walks of life.

My research was made possible by National Science Foundation (NSF) grant CNS-1446912, Semiconductors Research Corporation (SRC), Multi-Scale Systems Research Center (MuSyC), the Center for Networked Systems (CNS) at UCSD, the TerraSwarm Research Center, Oracle, Qualcomm, and Huawei. I am grateful for their generous support. I treasure the many collaborations and internships that gave me insight into visionary research in industry. I especially thank Kenny Gross and Kalyan Vaidynathan at Oracle, and Emily Shriver and Ankit More at Intel.

I have been fortunate to study alongside many talented and kind-hearted colleagues. Special thanks go to Jug Venkatesh, Baris Aksanli, Sinan Akyürek, and Pietro Mercati for challenging and supporting me throughout our time together. I also appreciate my friends who encouraged me from near and far, with perfect, excited understanding or a smile-and-nod: Eunice Lee, Yeesum Lo, Brandon Wouk, Mari Campbell, the Neuberts, and the Fifes.

I thank my parents, Stephen Chan and Cathy Luk, and my sisters Angeline and Eveline, for suffering my sporadic communications during many years apart. I have never not needed them.

Finally, I will never have enough words to thank Nate for his support, his patience, his sacrifices, and his unwavering faith. I started this for myself, but I am finishing because of us. Thank you.

Chapter 1 contains material from “An Ontology-Driven Context Engine for the Internet of Things” by Jagannathan Venkatesh, Christine Chan and Tajana Rosing, which appears in UC San Diego Technical Report CS2015-1009, 2015. The dissertation author was one of the primary investigators and the second author of this paper.

Chapter 1 contains material from “Context-Aware System Design” by Chris-

tine Chan, Michael H Ostertag, Alper Sinan Akyürek, and Tajana Šimunić Rosing, which appears in SPIE Defense + Security. International Society for Optics and Photonics 2017. The dissertation author was the primary investigator and the author of this paper.

Chapter 1 contains material from “Optimal Performance-Aware Cooling on Enterprise Servers” by Christine Chan, Alper Sinan Akyürek, Baris Aksanli, and Tajana Šimunić Rosing, which was submitted for consideration in IEEE Transactions on Computer-Aided Design. IEEE 2017. The dissertation author was the primary investigator and the author of this paper.

Chapter 2 contains material from “An Ontology-Driven Context Engine for the Internet of Things” by Jagannathan Venkatesh, Christine Chan and Tajana Rosing, which appears in UC San Diego Technical Report CS2015-1009, 2015. The dissertation author was one of the primary investigators and the second author of this paper.

Chapter 2 contains material from “A Modular Approach to Context-Aware IoT Applications” by Jagannathan Venkatesh, Christine Chan, Alper Sinan Akyürek and Tajana Šimunić Rosing, which appears in Proceedings of the International Conference on Internet-of-Things Design and Implementation (IoTDI), IEEE 2016. The dissertation author was one of the primary investigators and the second author of this paper.

Chapter 3 contains material from “Correcting vibration-induced performance degradation in enterprise servers” by Christine Chan, Boxiang Pan, Kenny Gross, Kalyan Vaidyanathan, and Tajana Šimunić Rosing, which appears in SIGMETRICS Performance Evaluation Review. ACM 2013. The dissertation author was the primary investigator and the author of this paper.

Chapter 3 contains material from “Optimal Performance-Aware Cooling on Enterprise Servers” by Christine Chan, Alper Sinan Akyürek, Baris Aksanli, and Tajana Šimunić Rosing, which was submitted for consideration in IEEE Transactions on Computer-Aided Design. IEEE 2017. The dissertation author was the primary investigator and the author of this paper.

Chapter 4 contains material from “Context-Aware System Design” by Chris-

tine Chan, Michael H Ostertag, Alper Sinan Akyürek, and Tajana Šimunić Rosing, which appears in SPIE Defense + Security. International Society for Optics and Photonics 2017. The dissertation author was the primary investigator and the author of this paper.

VITA

- 2011 Computer Engineering B.S.
University of Illinois Urbana-Champaign
- 2013 Computer Engineering M.S.
University of California, San Diego
- 2017 Computer Engineering Ph.D
University of California, San Diego

PUBLICATIONS

Christine Chan, Alper Sinan Akyürek, Baris Aksanli, and Tajana Šimunić Rosing. “Optimal Performance-Aware Cooling on Enterprise Servers.” (*Submitted for consideration in IEEE Transactions on Computer-Aided Design*) 2017.

Jagannathan Venkatesh, Baris Aksanli, Christine Chan, Alper Sinan Akyürek, Tajana Šimunić Rosing. “Modular and Personalized Smart Health Application Design in a Smart City Environment.” IEEE Internet of Things Journal Special Issue on Internet of Things for Smart Cities (*Accepted for publication in November 2017*)

Christine Chan, Michael H Ostertag, Alper Sinan Akyürek, Tajana Šimunić Rosing. “Context-Aware System Design.” SPIE Defense + Security. International Society for Optics and Photonics 2017 (*Invited paper*)

Baris Aksanli, Jagannathan Venkatesh, Christine Chan, Alper Sinan Akyürek, Tajana Šimunić Rosing. “Context-Aware and User-Centric Residential Energy Management.” International Workshop on Mobile and Pervasive Internet of Things (PerIoT). IEEE 2017

Jagannathan Venkatesh, Baris Aksanli, Christine Chan, Alper Sinan Akyürek, Tajana Šimunić Rosing. “Scalable Application Design for the IoT.” IEEE Software, Special Issue on Software Engineering for the Internet of Things. IEEE 2017

Jagannathan Venkatesh, Christine Chan, Alper Sinan Akyürek, Tajana Šimunić Rosing. “A Modular Approach to Context-Aware IoT Applications.” International Conference on Internet-of-Things Design and Implementation (IoTDI). IEEE 2016

Jagannathan Venkatesh, Christine Chan and Tajana Šimunić Rosing. “An Ontology-Driven Context Engine for the Internet of Things.” UC San Diego Technical Report CS2015-1009. 2015

Christine Chan, Boxiang Pan, Kenny Gross, Kalyan Vaidyanathan, and Tajana Šimunić Rosing. “Correcting vibration-induced performance degradation in enterprise servers.” SIGMETRICS Performance Evaluation Review. ACM 2013 (*Best Student Paper Award at GreenMetrics Workshop 2013*)

Christine Chan, Yanqin Jin, Yen-kuan Wu, Kenny Gross, Kalyan Vaidyanathan, and Tajana Šimunić Rosing. “Fan-speed-aware scheduling of data intensive jobs.” International Symposium on Low Power Electronics and Design (ISLPED). ACM 2013

ABSTRACT OF THE DISSERTATION

Context-Aware Platform Design and Optimization

by

Christine S. Chan

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2017

Professor Tajana Šimunić Rosing, Chair

The Internet of Things (IoT) envisions a web-connected infrastructure comprising billions of sensors and actuation devices that collect contextual data about the environment and respond accordingly. Machine and environmental context-aware computing opens up opportunities to learn new abstract information from sensed data, and is key for both standalone and interconnected applications to run efficiently. Since the volume of such data is growing exponentially, it is essential to develop automated techniques to derive useful data for design of scalable systems. We present a middleware framework along with analytical proofs that reduce the computing complexity of vast context collection and communication of abstract data among distributed nodes.

Effective and scalable distributed systems rely on accurate and flexible modeling of the devices and their interactions. In this dissertation, we study the use of context extraction across a hierarchy of devices, including resource-constrained embedded platforms as well as more powerful edge servers. We show that relevant context for optimizing performance and energy consumption, even for a single server executing database workloads, goes well beyond traditional variables like performance counters. In fact, we are the first to show that the speed of cooling fans can strongly affect the energy-delay product of a system. We leverage this additional contextual information as input to convex optimization that reduces the energy consumption of a server by 65% compared to basic PID controllers, or by 19% in comparison to advanced hardware management techniques proposed in literature. At an ecosystem level, we compare our distributed infrastructure to current monolithic implementations with single-point communications between sensor nodes and the cloud servers, demonstrating a reduction in combined system energy by 22-45%, and increasing the battery lifetime of power-constrained devices by at least 22x.

Chapter 1

Introduction

What happens if the Internet of Things (IoT) is fully realized? Market researchers speculate that there will be anywhere between 25 billion [1] to 50 billion devices [2] globally by 2020, producing about 2.3 ZB of fresh data annually [2]! As the number of devices and the volume of data explode, resources that we take for granted today will become highly contended - radio spectrum, communication bandwidth, even the processing capabilities of the cloud [3]. The current strategies that aggregate all data to the cloud for processing will not scale. An infrastructure needs to be in place to support these diverse devices. As more platforms join or age out of the system, the system infrastructure needs to scale and adapt to new data types, resources, or device capabilities. Based on their operating context, applications should anticipate user needs and preferences, and proactively respond. This requires an understanding of both innate and contextual abilities of the devices and systems.

Consider a most primitive representation of a signal processor: it receives some signal data as an input, applies mathematical function to that data, and then provides an output. Naively, it would seem that a digital signal is the only data needed for the processor to complete its functions. In reality, external forces, like network congestion, may affect the expected speed or frequency of input arrival, peak power constraints may limit data processing speed, and battery lifetime remaining may dictate the use of output communication. Throughout this dissertation, we define *context* to be any dynamically varying information about the

operating environment of a given platform or a network of platforms. This could include the ambient temperature, availability of computing resources, battery levels, or power constraints, in addition to the primary input source that the applications were originally intended to process. Machine and environmental context-aware computing opens up opportunities to learn new abstract information from sensed data, and is essential for both standalone and interconnected applications to run efficiently. We propose novel methods to represent and leverage this context data to control platforms and systems.

Considering that IoT applications are executed across a hierarchy of devices, we use multiple scopes when defining the context variables that influence a system or application. The backbone of computing in the cloud is supported by datacenter servers. In these machines, programmers traditionally use performance counters to estimate system behavior at a microarchitecture level. These traces produce contextual data about the state of the system, allows hardware management to mitigate how it might affect performance (e.g. hardware cooling vs. application performance). Extending to applications at the edge of the cloud, devices interact with each other, gateways, the environment, and humans. Each computing system produces context that can be observed by the outside world, as well as consumes the same contextual data for its own sustenance - to ensure correct and efficient operation.

1.1 Hierarchical context processing middleware

Sensor networks and ubiquitous sensing are evolving into a new concept the Internet of Things (IoT): the collection of sensing and actuation backed by the existing and growing Internet infrastructure [4]. The emerging implementation of the IoT has multiple distinct systems communicating with their own web-based backends, exposing distinct APIs for interaction and data retrieval. These heterogeneous devices are added, removed, or updated independently of each other by different manufacturers with different goals and release cycles, and the choices among them are entirely in the hands of the user. The wearable fitness tracker

market is a current example: Fitbit, Garmin, Moves, Microsoft, and Apple, the most prominent among several, have developed trackers and independent backends. Dozens of applications exist that independently scour the data stores for different pieces of relevant data to aggregate metrics for a users goals and progress. Extension of applications to new devices and APIs is a manual process requiring a redeployment of the user-facing application itself [5]. If the data sources and types change, the backend of the application might require reimplementations as well. Existing sensing and context infrastructures [5, 6] implement strong ontologies but lack support for such a changing system, as applications need to constantly adapt to the environment and the constituent devices.

1.2 Device context - optimizing a known, closed system

In a well-controlled datacenter environment, we can consider a single server as a closed system, facing little unforeseen external disturbance, and where the range of workloads and hardware states is stochastically predictable. Enterprise servers generate profit for their operators by delivering data and computing for a wide range of concurrent applications at high performance. They serve simultaneous requests from multiple clients (e.g. in big data and cloud computing services) while guaranteeing a quality of service (QoS) for each one. The QoS metric for different applications may vary - e.g. data throughput for online transaction processing (OLTP) database operations [7], or response time for interactive web applications, but as developments in processing power have advanced far ahead of storage and communication, many high-performance services are now I/O bound.

Their datasets are primarily stored on traditional disk media, and only partially cached in solid state drives and physical memory [8]. Datacenters rely on many drive types to maintain these different tiers of storage. Tier 2 comprises the largest proportion of current data that needs to be readily accessible (i.e. non-archival), and is commonly fulfilled by either Serial Attached SCSI (SAS) or Serial ATA (SATA) drives. They differ greatly in manufacturing tradeoffs in

terms of mechanics, materials and electronics [9] - the more robust SAS drives are reserved for more expensive deployments. There are emerging storage strategies that bypass hard drive performance issues to recover high access speeds, such as RAMCloud [10], which divides and distributes datasets across physical memory in multiple machines. Cost-sensitive datacenters still primarily deploy commodity SATA drives.

An excellent example of context data that strongly influences server performance and energy consumption, but hasn't been considered so far, is the cooling system. It generates mechanical disturbances that cause temporary crashes or misses in spinning storage systems, which in turn inflate workload execution times and server uptime electric bills [11]. These transient problems can be very difficult to diagnose in deployment. They are also neglected by existing thermal models and management policies. While current hardware management strategies leverage the tradeoffs between processor and cooling power consumption, and core execution speed, they neglect the relationship between cooling and data performance. Thus, their performance metrics are skewed in favor of CPU-intensive workloads and perform poorly for data-intensive ones.

1.3 Environmental context - adapting to an unknown world

In the cyber-physical world, a system is no longer closed. Beyond datacenter workloads, Internet of Things (IoT) application demands are unpredictable. The application responds to changing environmental context by design, so the devices should also respond by leveraging context to function efficiently. Experimental context should inform how the larger application behaves, including actuating device sleep modes, sampling intervals, or communication intervals.

In contrast to yesterday's systems, where sensor nodes collect data and blindly send all raw data to aggregation in the cloud, data analysis now occurs at each layer in the system. Increasingly, intelligence is appearing at every layer of device hierarchy. Smart sensors not only sense the physical world around them, but

also provide data management and analysis capabilities (e.g. detecting anomalies). IoT gateway devices gather the data from relevant sensors in their surroundings and derive context for ever changing applications.

As an example, the DELPHI project is a large-scale integration of data sources at various levels of hierarchy, including electronic health records (EHR), environmental data, medication usage and physical activity monitors [12]. As sensor networks and devices become increasingly flexible and personalized, much effort has been made toward empowering citizens to monitor and analyze their own environment. For example, healthcare or fitness applications in the IoT require hardware and software systems for crowd sourced air quality monitoring. There are many ideas and prototypes for crowd sensing, but there is a need for connections to higher-level actionable health information.

1.4 Thesis contributions

This dissertation explores various ways to leverage “context” for sensing and control, always mindful of delay and energy costs. Based on our understanding of current IoT development principles and needs, from both data and machine perspectives, we propose a middleware framework that efficiently divides and conquers the many automated tasks required to support context aware computing at scale. We also present methodologies and results for investigating relevant context-aware control of single and multiple nodes in cooperation. The rest of this document will expand on the following contributions:

- We propose a modular approach to designing incoming IoT applications, breaking them into an equivalent set of functional units (context engines) whose input/output transformations are driven by general-purpose machine learning. This organization improves compute redundancy and computational complexity with a minimal impact on accuracy. In conjunction with formal data specifications, or ontologies, we can replace application-specific implementations with a composition of context engines that use common statistical learning to generate output, thus improving context reuse. We

provide formal proofs for several characteristics of the context engine organization to motivate adoption: for any application with non-linear complexity, dividing the input space into multiple context engines will provably reduce total computation complexity. In fact, the complexity of a context engine system is minimized when each individual engine only takes two inputs. Moreover, context engines scale linearly with increases in input data load. In a sample application using two unreliable sources of overlapping data, we use a Bayesian network to automatically learn when to trust and when to discard particular data streams based on the context of that data collection. As a result, we can produce an output stream with an average 60x accuracy improvement over either of the individual sensor streams alone.

- One potential application area for the context engine framework is in a data-center, since it houses many computing and communication nodes organized in a hierarchy of rows, racks and individual servers, where each level produces context information. As an example of extracting high-level context from low-level data streams, we focus on an in-depth study into leveraging context to optimize operation of a complex enterprise server platform. First, we identify a previously-neglected link between the cooling system and application performance. We characterize commodity drive sensitivity to fans up to 88% lower performance in realistic cooling situations. Since current enclosure, cabinet and raised-floor room designs are insufficient in eliminating all vibrations [13], we turn to software-based detection and control. We develop and integrate interdependent analytical models for performance, power, thermal, cooling. Fortunately, though datacenter workloads can be highly demanding of the hardware platform, and increasingly complex in terms of software optimization, they are also fairly well-known and predictable at a large scale [14]. By modeling the workloads in terms of their resource consumption, we can predict the application’s needs and manipulate the operating conditions such as temperature and core availability to improve performance. Notably, we approach the server efficiency problem by targeting the cooling-performance relationship. Server context is represented using a

combination of models derived from online measurements. Our method summarizes the cost of execution into a single cost metric, either energy or delay, for formal optimization. Our results are based on real physical telemetry of a late-model multi-threaded, multi-core server processor running a standard database benchmark suite (TPC-H [15]). This multi-model cost function can be solved to find optimally low-cost fan speeds, saving 19-65% of CPU and fan energy while meeting critical thermal constraints.

- We also consider distributed nodes that cooperate under a specific IoT application using the context engine. Since we hope to reduce redundant efforts and share data between applications, we take advantage of publicly available data sources to learn about demographics, city planning data, citizen habits, and environmental data, and how that implicates respiratory health. For example, to investigate asthma risks, ER admissions and hospitalization rates are used to model asthma complications. Since we have access to small, low power programmable devices with some, albeit limited, processing power, context engines are one opportunity to execute generalized machine learning at the edge of the cloud. This enables aggregate models and knowledge even at a local level. They can be used to provide direct and timely feedback to users in the field, without first sending all raw data to the cloud, waiting for processing, and waiting for the return delivery of useful information. We also leverage these techniques to enable low power devices to manage their own sensing behavior, such as down sampling intelligently without missing critical data, thus saving battery power and restricting power-hungry data transmissions to the cloud. We find that sending machine learning tasks to the edge of the cloud lowers energy costs in the whole system (22-44%). Additionally, using machine learning tasks like anomaly detection to process environmental context and dynamically adjust sampling intervals can increase device battery lifetime by 72x. Though we use a smart health application as a case study, these observations can be generalized to other IoT applications that span a range of devices including small sensor nodes, gateways and cloud aggregators.

This chapter contains material from “An Ontology-Driven Context Engine for the Internet of Things” by Jagannathan Venkatesh, Christine Chan and Tajana Rosing, which appears in UC San Diego Technical Report CS2015-1009, 2015. The dissertation author was one of the primary investigators and the second author of this paper.

This chapter contains material from “Context-Aware System Design” by Christine Chan, Michael H Ostertag, Alper Sinan Akyürek, and Tajana Šimunić Rosing, which appears in SPIE Defense + Security. International Society for Optics and Photonics 2017. The dissertation author was the primary investigator and the author of this paper.

This chapter contains material from “Optimal Performance-Aware Cooling on Enterprise Servers” by Christine Chan, Alper Sinan Akyürek, Baris Aksanli, and Tajana Šimunić Rosing, which was submitted for consideration in IEEE Transactions on Computer-Aided Design. IEEE 2017. The dissertation author was the primary investigator and the author of this paper.

Chapter 2

The Context Engine

The Internet of Things (IoT) refers to an environment of ubiquitous sensing and actuation, where all devices are connected to a distributed backend infrastructure. The main benefit of the IoT is the ability to use myriad sensor data, leveraged into high-level information about the entities in the system for reasoning and actuation in context-aware applications. A formal specification, or *ontology*, provides a regular data interface between different components in the system. In addition, IoT middleware is required for context-aware applications to operate in an environment with constantly changing data, sources, and context. In this chapter, we present a context engine for IoT applications founded on an ontology that specifies and reasons on context information. We explore and build upon related work on IoT needs and ontological principles. Our infrastructure leverages context information for learning and processing a changing environment over many different available nodes in the system. We provide a numerical discussion as well as formal proofs for the reduced complexity, accuracy tradeoff, and improved scalability of the context engine framework. Finally, in a sample application, we demonstrate machine learning from heterogeneous, intermittent sources, producing an output stream of context information 60x more accurate than either of the individual sensor streams alone.

2.1 Background and related work

The IoT was initially associated with simply providing value-add to user convenience products and commercial advertising. The public sector was one of the first big industries to adopt IoT technologies, widely deploying systems for the smart grid and environmental monitoring [16] to name a few. With increasing adoption comes increasing loads and responsibilities to the user base, such that the underlying system infrastructure must be engineered to meet performance, reliability and safety constraints [17, 18, 19]. High-level context provides a representation that is both lower in computational overhead and more intuitive for application developers to use in reaction to IoT sensor stimuli, such as crafting cognitive assistance for medical patients [20], or custom learning environments for online students [6]. Publications exist on various aspects of energy management in a power grid such as building demand response [21] and generation control [22]. For residential control, previous studies include appliance automation [23, 24], lighting [25], appliances [26, 27], renewables [28], and energy storage [29]. Neighborhood level control with energy storage or renewable energy sources attempts to minimize energy costs [30, 31].

Hong et al. [32] build context information based on a host of labeled environmental and user sensor data (e.g. biometrics, GPS location, interaction with phone, weather, etc.) and context rules. Lee et al. [33] present a location prediction model based on a dynamic Bayesian network, where accuracy significantly exceeded static networks. The location model enables their ultimate goal of supporting ubiquitous computing decisions. With the growing popularity of alternative education, e-learning systems can personalize learning materials and recommendations based on modeling student profiles and context [6]. There are many more mobile applications that operate on context awareness for localized user information [34], vehicular safety [35], or battery saving [5].

2.1.1 Use of machine learning

Streaming data collected from human subjects is often noisy, so sliding windows of the continuous data must be smoothed and preprocessed before inputting into an analytic or modeling framework. Some publications further apply machine learning techniques to model user behavior and interaction with their physical environment. K-means clustering is a prevalent way to automatically relate low-level data into high-level contexts [32]. Reinforcement learning is an important learning method for context awareness in IoT applications, as users are already innately involved in sensing and actuation. It invites user interaction to reinforce and guide the system towards better accuracy and intuitive actuation. For example, Madhu et al. [6] schedule reminders for a user who is cognitively or orthotically impaired. They use temporal constraint reasoning to describe a daily plan and reinforcement learning (function approximation-based learning) to find optimal actions, subject to adjustable human parameters. Rashidi et al.[36] remove reliance on labeled data by performing unsupervised learning over low-level sensor event sequences to extract patterns that represent high-level activities, even if the activities are discontinuous or varied over time.

2.1.2 Context Ontologies

Ontologies are formal data representations that categorize the vast amount of unregulated and diverse data from information sources in the IoT [37]. They help classify the data provided to applications, but the organization of the IoT still makes application deployment difficult. There are many previous publications [35, 36] that outline formal context models for domain-specific designs, but do not intend to share data or actuation beyond their original one-off applications. However, pervasive sensing provides many of the ontologies that are now adapted to the Internet of Things. One of the earliest is the resource description framework (RDF)[37], which annotates all web objects with semantic information, implemented in XML, a web-ready format. The aggregation of these annotations forms a directed graph that is already used in context-aware web applications such as search engines. While this is adequate for objects such as websites, as was its

original goals, this binary object-object connections are insufficient for the Internet of Things, which requires a richer description of relationships, as well as an easier way to query and determine these relationships, make inferences, etc. To address these limitations, the successor to RDF was the Web Ontology Language (OWL), now a web standard, which also transitioned from loosely defined and typed systems suitable for Wireless Sensor Networks to a formal ontology [38]. OWL has been designed as a hierarchical system, with sub-domains of objects (e.g. appliances) encapsulated by the domains they live within (houses). Several systems have been designed using OWL: smart spaces[38], meeting room organization [38], hierarchical modeling of the physical environment [39]. OWL-S is an implementation built on top of OWL for describing semantic services[37], which has been adopted by others for the scalability and testability of their services model[40].

However, OWL typically operates under a very strictly defined hierarchy. Although this works well for static applications that rely solely on a fixed set of data, IoT applications may have to deal with changing sources and sinks. Nodes should be removable and the system should still be able to operate to the best of its capacity. In addition, the amount of data for which each application must be responsible can grow rapidly, as the amount of infrastructure-related data (dependencies, relation annotation, etc.) that each application needs to manage can grow faster than the data itself. Another extension of OWL handles this issue: the Context Modeling Language (CML) [41]. The system is based on the Object-Role Model, which is preferable for the Internet of Things, as all context data is attributed to a physical or virtual entity (the object) and provides a particular form of information associated with it (role). The ontology reverts to a flatter hierarchy than OWL this allows an application to deal with as little context as it requires, rather than be responsible for the entire graph. Furthermore, CML provides a direct web-oriented communication language in XCML, a markup implementation that is very important for an Internet-based backend. We borrow much of our syntax from XCML. This provides an ontology for mapping names of context variables to their values. The context space includes all possible context variable names, which may grow according to any additional data types the ontol-

ogy describes. Finally, to aid the adaptability of context-aware applications, CML introduces the concept of context dimensions, which define the minimum subsets of context spaces, in which an application can perform meaningful computations and operations. By specifying dimensions in the same markup as the data, CML opens the doors for the specification itself to be changed to adapt to the data sources available.

2.1.3 IoT Software Infrastructure and Middleware

In an IoT context, middleware is software that acts as a bridge between low-level sensors and the backend, or between the backend and user-facing applications. Other works have explored context-aware middleware frameworks. Perera et al. [42] propose a sensor configuration model for the IoT that can handle sensor filtration and reasoning, implemented via asking the user a series of questions. However, the reasoning seems to fall short of an open platform for machine learning, instead limiting reasoning to an annotated dependency graph. Similarly, [43] overviews several middleware implementations, identifying certain aspects of each: ontology-based, flexible for reasoning, data filtration and adaptation, but does not present a single solution encompassing all aspects.

An application-facing infrastructure that enables these properties is missing. The various proposed ontologies are only applied towards very specific applications [32, 36, 20, 38] there is no larger system within which the applications exist, nor is there the ability for other applications to use the resources made available to the one, as should be in the real Internet of Things. The closest entity is the middleware layer implemented in [36], which provides a general framework, but only within the domain of “smart spaces”, and again, specific to the application that is designed. More importantly, none of the related works account for application adaptability. The emerging IoT implementations require the ability to merge data from different sources and backends into a common context representation, and on the application side, to seamlessly incorporate new data or formats without interrupting the application. Regardless of whether adaptation is a product of a changing environment or sources joining and leaving the sensed surface, the

framework around the applications are made static. CML[41] was the only ontology to allow application processing using different sets of available data, but the implementation covered only the ontology and not the applications. Real IoT applications need to exist in an infrastructure that fosters changing the data, sources, and internal reasoning. The context engine developed in this work fits this gap in the current landscape. We present a framework and methodology that developers inherit, allowing them to describe their source data and end-user applications in terms of their context space and dimensions. This enables applications that are flexible with new sources and dimensions, and whose behavior can be extended, even during runtime.

2.2 Hierarchical context modeling

We envision a world in which intelligence is present at every layer of device hierarchy. Smart sensors will not only sense the physical world around them, but also provide data management and analysis capabilities (e.g. detecting anomalies). IoT gateway devices will gather the data from relevant sensors in their surroundings and derive context for ever changing applications. In contrast to today’s systems, data analysis will occur at each layer in our system, and therefore demands novel and scalable software infrastructure, machine learning algorithms that can leverage data across distributed computing nodes, privacy strategies that are sensitive to individual privacy needs while providing sufficient utility across the hierarchy of devices, and communication methods that can effectively aggregate data to handle inevitable congestion while ensuring application data timeliness needs are met.

Under the old paradigm, silo-ed, standalone applications collect and process all the data they rely on, resulting in redundant computations across independent applications. Considering the expansive growth of data and wide range of computation nodes available, we propose a more scalable way to develop applications across the ecosystem. In our new paradigm (Figure 2.2), we have identified an improved approach: multiple general-purpose functional units (context engines) that each drive data processing for a single output context variable, recomposed to

be functionally equivalent to conventional monolithic application. Each low-level sensor is only sampled from minimal times, and once their raw data is processed into higher-level information (e.g. from raw GPS or radio signals to semantic “locations”), that information can be shared instead of redundantly computed.

As Figure 2.2 shows, the system operates as a hierarchy of multiple-input-single-output (MISO) context engine units to improve machine-learning reasoning while reducing data redundancy and accomplishing the same functionality as the corresponding state-of-the-art multi-input-multi-output (MIMO) applications. Smaller hierarchical functional units represent simpler data translation at the tradeoff of more functional units. This promotes the use of general data transformation in each context engine using machine learning to generate outputs in place of application-specific code. We showed the theoretical arguments for the scalability of this composition in terms of training data required and machine learning execution time in previous papers [44, 45]. Others have tested and measured the feasibility of executing traditional machine learning algorithms on resource constrained devices. The energy and delay of both linear and non-linear algorithms can be automatically determined, based largely on network bandwidth and data size. [46].

Once the context has been generated, we aggregate data prior to communicating it to the next layer in our system hierarchy by using our novel data aggregation scheme which ensures an optimal tradeoff between meeting application timeliness needs while minimizing communication costs. Thus, a context-aware application can be created by simply specifying the inputs and output of each functional unit alone, and allowing hierarchical machine learning to generate and train a model based on input and output observations. Exposing intermediate data reduces the complexity and redundancy of applications in the larger infrastructure, and enables easy data sharing among other engines. Creating such a hierarchy of context engines is very beneficial for distributed data processing, as it makes it very easy to decompose a large application into sets of context engines that can each run on various devices present in the system. Because context is derived and generated at each layer, the total amount of data sent is dramatically reduced as

compared to today's state of the art, thus making such large-scale deployments not only possible but also much more efficient.

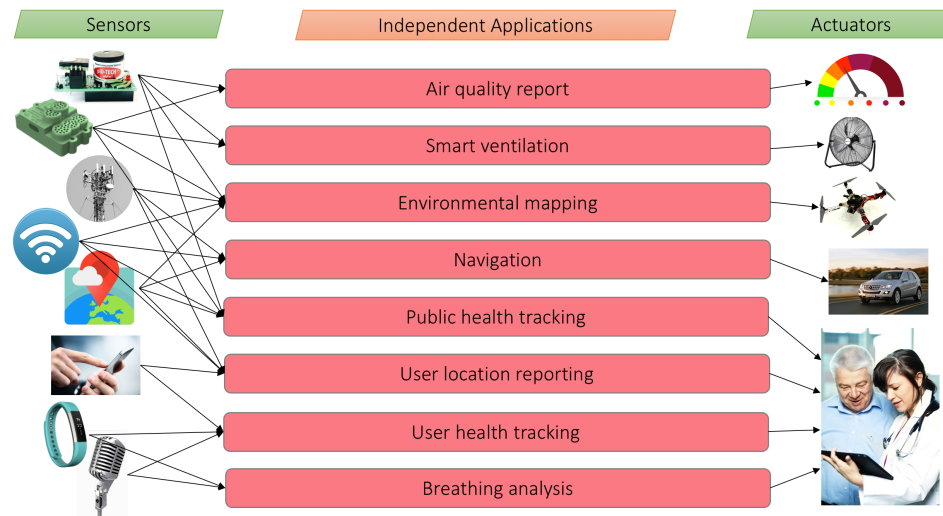


Figure 2.1: Traditional sensing application development serving multiple end-user applications

If they participate in the context engine ecosystem, each device should share their processed data, which may serve as further input data into other applications. For example, air quality sensor data has a multitude of uses. It can be used simply for local display, or to inform the actuation of building ventilation, or for regional environmental studies. In traditional application design, each programmer would have to set up data sampling and processing of the analog electrochemical sensor voltage levels into human readable gas concentration values. In a context engine ecosystem, only the first programmer would have to put in that effort - consequent applications can simply share that higher level context.

Since we have access to small, low power programmable devices with some (albeit limited) processing power, context engines are one opportunity to execute generalized machine learning at the edge of the cloud. They can be used to provide timely feedback to users in the field, without the need to send all raw data to the cloud, and wait for processing, and the return delivery of useful information. These techniques can also enable these low power devices to manage their own sensing

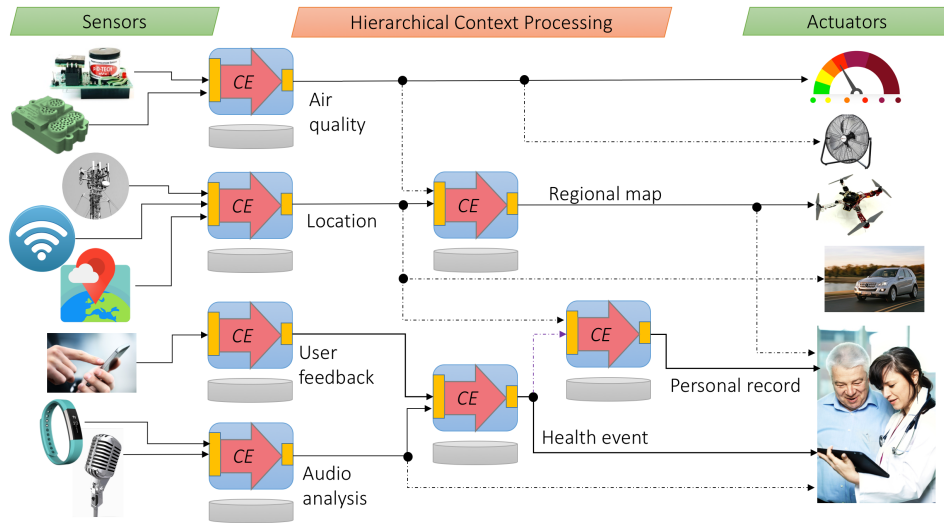


Figure 2.2: A hierarchy of *context engines* sharing intermediate data processed from various sensors to serve multiple end-user applications

behavior, such as down sampling intelligently without missing critical data, thus saving battery power and restricting power-hungry data transmissions to the cloud.

2.2.1 Generalized Data Transformation

Our approach, a modular multi-stage context engine, results in more functional units (FU) per application. An important consequence is that each FU that composes the application is a simpler translation of input data to a single output. This enables the use of a general data transformation in each context engine in place of application-specific code. Thus, a context-aware application can be created by specifying the inputs and output of each FU alone, and allowing the data transformation algorithm to incur the processing overhead generating and training a model based on input and output observations.

We leverage the ontologies that are already present in the current state of the art of IoT middleware. From a data standpoint, they regulate inputs and outputs of applications. Applications that participate in the system must enforce the ontology’s specification: discrete variables must provide a set of possible states to populate the probabilistic condition tables; continuous variables must specify

a valid range of values that can be clustered. We can exploit this ontological information for machine learning algorithms that clusters results based on the space of the input and output variables, as well as determines the training space and list of prior observations.

Matrix-based stochastic learning models express potential data dependencies as a system of equations. Some use predefined notions about the inputs to establish linear or nonlinear equations, while others start with a linear combination of the inputs whose coefficients are unknown. Over time, observed input and output data is gathered until the coefficients can be trained and a model generated. Since complex relationships can exist among the input data for an IoT application, and a purely linear model may not be sufficient [47], several works [48] [49] [50] implement learning by considering higher orders and time correlation. In our current implementation, we leverage TESLA, a learning model originally designed for solar forecasting, as the data translation algorithm in our context engine [51]. It provides efficient model generation: $O(n^\alpha)$, where n represents the number of inputs and α represents the function order of the Taylor expansion. The generic function of this expansion is established as follows:

$$\sum_{i=0}^n C_i x_i \text{ (1}^{st} \text{ order), } \sum_{i=0}^n \sum_{j=0}^i C_{ij} x_i x_j \text{ (2}^{nd} \text{ order) etc.} \quad (2.1)$$

where C_{ij} represents individual coefficients learned once observations are determined, and $x_0 = 1$ (the constant coefficient). The resulting equation is $\mathbf{A}x = \mathbf{B}$, where \mathbf{A} is the row matrix of input observations; x is the column vector of coefficients, and \mathbf{B} is the column vector of output observations, each entry correlating with the corresponding row of \mathbf{A} , and solved by least squares estimation.

One limitation to this model is that at least m independent observations are required for training, where $m = n$ for first-order, n^2 for second-order, and so on, which can become space-inefficient as the order increases. Finally, using the model is as simple as solving the equation using the learned coefficients and the input context, which produces the output context.

We demonstrate TESLA here because of its general formulation, versatility across different function orders, and applicability to context processing, but other

statistical learning approaches exhibit similar properties: for example, Bayesian Networks [48], Hidden Markov Models (HMM), and Artificial Neural Networks (ANN) can leverage input and output domain spaces to conditional probability models and parameters that define preferred paths gu2013.

2.2.2 Integration with Ontologies

The context engine architecture we propose incorporates both modularity and general data transformation, significantly reducing application-specific and implementation overhead. In addition to the overall application input and output context variables, we must identify the data flow and intermediate context required by the additional functional units. Existing context engine outputs that match the input needed by the application, that engine will be reused rather than defining and generating a new one. For example, Figure 2.3 identifies the context variable for GPS location for a particular object ("User1") using the context modeling language (CML).

```
<?xml version="1.0" ?>
<xcml:context-model xmlns:xcml="xcml">
  <xcml:context-dimension name="GPSLoc">
    <xcml:context-key name="id" value="User1"/>
    <xcml:context-key name="lat" type="float" min="-90" max="90"/>
    <xcml:context-key name="long" type="float" min="-180" max="180"/>
  </xcml:context-dimension>
</xcml:context-model>
```

Figure 2.3: Ontology specification for GPS data, with coordinates, source, and range.

If this variable specification is present, an application that require GPS location for that object can simply refer to this variable as input. If this variable is populated in the data store, either by a sensor or as the output of another application, changes to this variable will be applied. In particular, if deriving location is intermediate context from another application, it is now exposed for reuse without additional processing overhead by the current application. Intermediate context variables that are not already defined must be outlined using the

ontology, specifying both the data source and the domain or range for input or output, respectively. Currently, ontological definition of inputs and outputs allows applications to retrieve and output data to the backend infrastructure. As we mentioned in previous sections, we additionally use ontologies to generate the constraints for the statistical learning algorithms. The application designer simply specifies the input and output ontologies for each context engine. The common data transformation records observations until there are enough to train the functional model for the order of computation, at which point it begins generating the output context for each successive input observation set.

2.3 Theoretical analysis

The sequential, hierarchical approach raises questions about the complexity overhead, latency, and accuracy of breaking down a possibly compact application into a composition of steps. We validate our approach by proving that the overall computational complexity of the architecture is actually reduced with a marginal impact on output accuracy.

2.3.1 Complexity

Theorem I: Dividing a context engine into multiple context engines decreases the total computational complexity of a nonlinear system.

Proof: We show that dividing the processing of N inputs from a single context engine to multiple context engines decreases the total computational complexity. We start with a general representation of a context engine: N number of inputs and a computational complexity order α for a maximum computational overhead N^α . We divide the single engine into two stages, where there are multiple engines with an arbitrary number of inputs of A . The number of engines of the first step becomes $\frac{N}{A}$. The second stage takes the outputs of the first stage and gives the final output. The total complexity overhead of this system is $\frac{N}{A}A^\alpha + \left(\frac{N}{A}\right)^\alpha$. We look for the conditions where the two-stage has a lower complexity than the single

engine:

$$\frac{N}{A}A^\alpha + \left(\frac{N}{A}\right)^\alpha < N^\alpha \rightarrow A^{2\alpha-1}N + N^\alpha < A^\alpha N^\alpha \quad (2.2)$$

$$A^{\alpha-1}A^\alpha < N^{\alpha-1}(A^\alpha - 1) \rightarrow \left(\frac{N}{A}\right)^{\alpha-1} \left(1 - \frac{1}{A^\alpha}\right) > 1$$

Although the selection of A is arbitrary, there are two limiting conditions: A must be an integer and the number of context engines must be an integer $\left(\frac{N}{A}\right)$. Thus, the minimum for A is 2 and the maximum is $\frac{N}{2}$, i.e. 2 engines. We do not consider $A = 1$ or $A = N$, as neither contribute to division of the single-stage context engine. The final inequality is the multiplication of two terms. The first term is minimized when $A = \frac{N}{2}$ and results in 2^{-1} . The second term is minimized when $A = 2$ and results in $1 - 2^{-\alpha}$. This provides a lower bound for the result: $2^{\alpha-1}(1 - 2^{-\alpha}) = 2^{\alpha-1} - \frac{1}{2}$. If we prove that this lower bound satisfies the inequality, the multiplication result must also satisfy the inequality:

$$\left(\frac{N}{A}\right)^{\alpha-1} \left(1 - \frac{1}{A^\alpha}\right) > 2^{\alpha-1} - \frac{1}{2} > 1 \rightarrow a > \log_2 3 \approx 1.6 \quad (2.3)$$

This proves that if the complexity order of the system is greater than 1.6 (e.g. for 2^{nd} and greater integer function orders), any arbitrary division of the single engine results in a decrease in computational complexity. The corollary to this theorem is:

Theorem II: The complexity of a system of context engines is minimized when each individual engine contains 2 inputs.

Proof: Theorem I shows that dividing an engine decreases complexity if the system has a complexity order greater than 1. The number of context engines is $\frac{N}{A}$, which gets its maximum value at $A = 2$.

While context-aware applications do not necessarily fit perfectly into a system of two-input engines, as we reduce inputs into each context engine and increase the path from the initial input to output context, we reduce the overall system complexity.

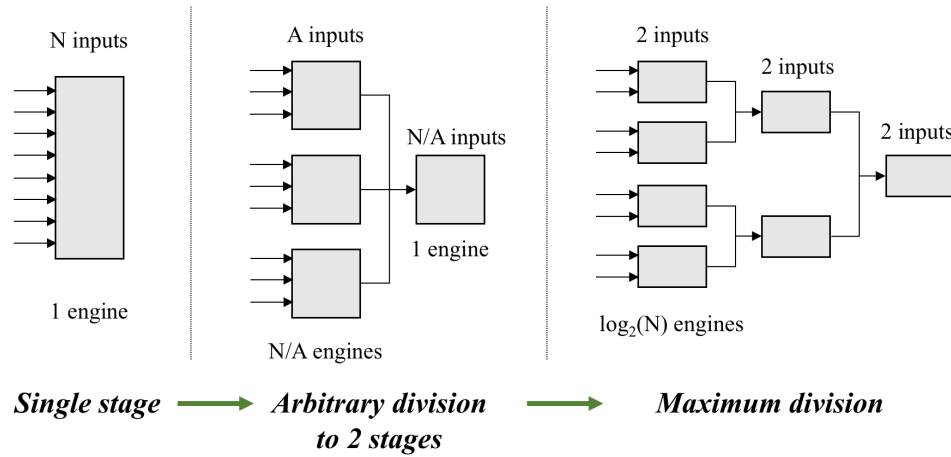


Figure 2.4: Breakdown of a single-step into lower-complexity equivalent reductions, with the minimum complexity occurring with maximum division (two-input engines on the right).

2.3.2 Accuracy

We now investigate the accuracy change between sequential and consolidated applications. We begin with a general functional representation of data transformation from statistical learning: generating a model for data transformation as a polynomial of varying complexity and functional order, which can be solved to best-fit through techniques such as regression [47]. By providing the means to vary the inputs (ontology) and relationship (functional order), complex relationships between the inputs and output can be represented and trained. While a general formulation differs based on the function order and application, this example illustrates the accuracy change between 2-input context engines and a 4-input single-stage context engine, as in Figure 2.5.

We can compare the two implementations through their respective transformation functions. We use a polynomial function as the general data transformation. The second-order Taylor expansion of f :

$$f(i_1, i_2) = f_0 + f_1 * i_1 + f_2 * i_2 + f_{11} * i_1^2 + f_{22} * i_2^2 + f_{12} * i_1 * i_2 \quad (2.4)$$

where f_{ij} are the corresponding coefficients. The other context engines (g , h , and fgh) have corresponding expansions. However, because h is composed of the

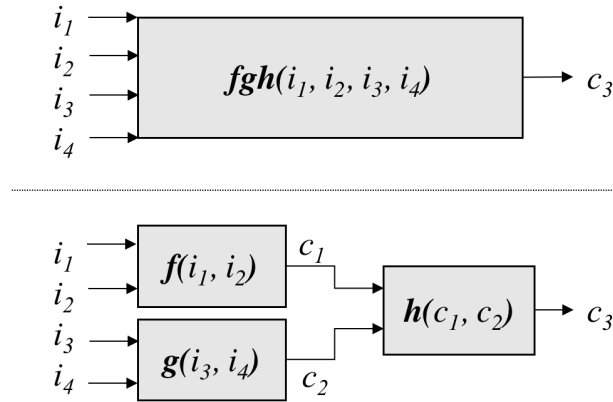


Figure 2.5: Functionally equivalent organizations of the single-stage application (top) and the sequential (bottom).

outputs of f and g , it can be represented as a function of the f_{ij} and g_{ij} coefficients and the initial input variables i_1 to i_4 . This is directly comparable to fgh , which is also a function of the initial inputs and a set of coefficients represented as λ_{ij} (e.g. $\lambda_0 + \lambda_1 * i_1 + \dots$). $h(f, g)$ evaluates to fgh exactly except for the ratio of g_1 and g_2 , which matches two different pairs of coefficients of fgh :

$$\frac{g_1}{g_2} = \frac{\lambda_{13}}{\lambda_{14}}, \quad \frac{g_1}{g_2} = \frac{\lambda_{23}}{\lambda_{24}} \quad (2.5)$$

This means that the sequential context engine will have the same accuracy as the single-stage only when the ratio of λ_{13} to λ_{14} matches the ratio of λ_{23} to λ_{24} . If the ratios do not match, 1 out of the 4 coefficients in this ratio cannot be represented, though the other 3 in the ratio as well as the remaining 7 coefficients in the equation are all represented accurately. This can be modeled as some error factor δ in the λ_{24} coefficient, which contributes $\delta * i_2 * i_4$ *truncation error* between the sequential and single-stage context engines.

We also quantify the impact of input signal noise on the sequential context engine compared to the single-stage approach. We model each input with zero-mean additive white Gaussian noise: $x_i + w_i$, a common expression of sensor noise [52]. The resulting noise coefficients propagate through the application. In the sequential case, f and g both propagate the original input noise to h . The truncation error of the sequential context engine is now compounded by additional

noise:

$$\delta * i_2 * i_4 + \delta * w_2 * i_4 + \delta * w_4 * i_2 + \delta * w_2 * w_4 \quad (2.6)$$

The first term is the truncation error we previously quantified; the second and the third terms are the scaled Gaussian values due to noise; and the last term is a chi-squared distribution also derived from noise. The significance of the error terms is entirely dependent on the relationship between the cross-product input terms i_2 and i_4 . If the output context is highly dependent on the cross products, the weight of the noise and truncation terms will pose significant error. From a system design perspective, simply selecting highly correlated input terms for context engines - an intuitive choice nonetheless - will mitigate truncation error, as the impact of the missing cross-coefficient terms is minimized.

2.3.3 Scalability

As previously mentioned, we envision IoT applications operating in an environment with dynamic computational ability. Specifically, there are different distributed compute nodes for sensing, infrastructure, and actuation, including sensor and actuation platforms, mobile devices, and backend storage and processing. The complexity arguments from the previous subsection show potential improvements even if we are confined to a single compute node. However, as the number of compute nodes grows, there is inherent scalability in the sequential context engine approach.

We leverage and extend the scalability definition from distributed systems [53] and IoT systems [54]: identifying the change in *speedup* under the conditions of 1) changing number of compute nodes for a given application (*strong scaling*) and 2) changing amount of input data (*load scaling*).

We define speedup for the context engine approach using the following

piecewise function:

$$S(k, N) = k \text{ for } 1 \leq k \leq N - 1$$

and (2.7)

$$S(k, N) = 1 \text{ for } k > N - 1$$

where k is the number of cores (for strong scaling) and N is the number of inputs (for load scaling). Since we generalize the processing in each functional unit to the same algorithm, we deal with functional order as a general term representing the polynomial model's complexity and the number of inputs.

For *strong scaling*, the hierarchical application behaves like a distributed system, taking advantage of compute nodes as they are made available. However, even with maximum division (i.e. an application broken up into $N - 1$ 2-input context engines), the speedup is ultimately capped: when more than $N - 1$ compute nodes are made available, there are more free nodes than functional units. At best, some FUs can be reallocated to more capable nodes, but at this point, the system is already overprovisioned and will scale as the system is expanded.

Load scaling, or the increase in input data, is particularly important for IoT applications, as the growing amount of data in applications should be appropriately handled. An infusion of new input can be addressed by either:

1. Increasing the number of inputs to a single (or multiple different) context engines
2. Expanding the hierarchy with more low-input context engines.

In case 1, for every m additional inputs, the complexity of the updated context engine increases from n^α to $(n + m)^\alpha$. In contrast, for case 2, assuming a maximum division of input (a binary tree of two-input context engines), for every m inputs, we add at most $m - 1$ new context engines, increasing the complexity to $(m + n - 1) * 2^\alpha$. As m increases, the complexity of case 1 grows much faster than case 2. Moreover, the second option falls in line with our goal of more modular

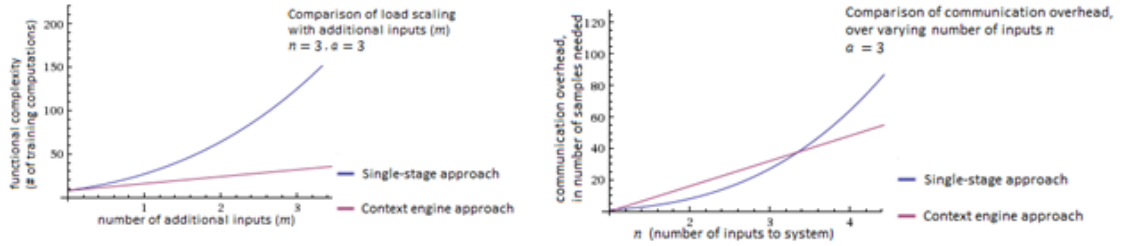


Figure 2.6: Scalability comparison between the single-stage approach and the context engine, comparing complexity with input size (left) and communication overhead over input size (right).

applications. Load scaling, with a system growth of mn^α , represents linear growth in the modular context engine approach. Figure 2.6(left) illustrates this growth compared to the equivalent single-stage application’s growth as m increases, with fixed n and $\alpha = 3$.

The increase in input data also affects the communication overhead of the application, another factor impacting scalability. Each functional unit must train its machine learning algorithm in order to generate appropriate output context. In the training phase of an n -input α -complexity single-stage application, the functional unit must receive n^α individual input samples and a corresponding n^α output samples from the source and sink devices to calculate the context engine’s coefficients using TESLA. Similarly, a sequential application with maximum division (2-input context engines) requires $2(n - 1) * 2^\alpha$, or $2^{\alpha+1} * (n - 1)$ input and output samples. Figure 2.6(right) illustrates the communication overhead of the context engine approach vs. the consolidated approach over the number of inputs.

2.4 Case study: Fitness tracker

Fitness trackers currently dominate the wearables market [55], with many interface options available to the user, including smartphone and desktop applications, and development APIs opened to development enthusiasts. Users often wear devices that report partially redundant information, even if they were designed to collect disparate data at a low level. This example collects user activity

from two independent devices - a Fitbit Flex step counter [56] and an Android smartphone running the mobile application Moves [57]. While both data streams report a user’s step count, they arrive at that conclusion in different ways and with different reliabilities. We have a simple goal of obtaining an accurate daily step count for a single user based on these two data streams, by learning when and where to trust one data stream over the other.

2.4.1 Input sources

Fitbit reports minute-by-minute step counts for a user, based on accelerometer readings. The data traces are simple. Each piece of context data only contains a start and end time for the interval (currently fixed at 1 minute) and the number of steps counted. While the device actually reports more, such as a user’s sleep mode and inferred activity levels, our application does not need those pieces of data and thus leaves it out of the context dimension.

Moves traces include higher-level activity readings such as semantic location names, type of motion, and a step count if the user is walking. These readings are inferred from a variety of low-level sensors and crowd-sourced data (smartphone accelerometer, GPS, social location check-in service, etc). The traces are divided into “segments”, where the user is either sedentary, moving from one point to another, or moving around within one location. Each segment is bookended by a start and end time, but it may also exercise different combinations of context keys, some of which are organized into nested structures, as shown in Figure 2.7. When the user is at a location for an extended period of time, it records the GPS coordinates, and may include a semantic locality name depending on availability. The semantic location name may be estimated from a web-based social location service like Foursquare [58], or manually entered by the user - this naming source is also recorded and can imply a measure of confidence in the location. If the user is in transit, Moves records instantaneous positions in a list of timestamped GPS coordinates, called “trackpoints”.

The user manually collects the verification data for an actual step count. The start and end timestamps are required to align collected data with Fitbit

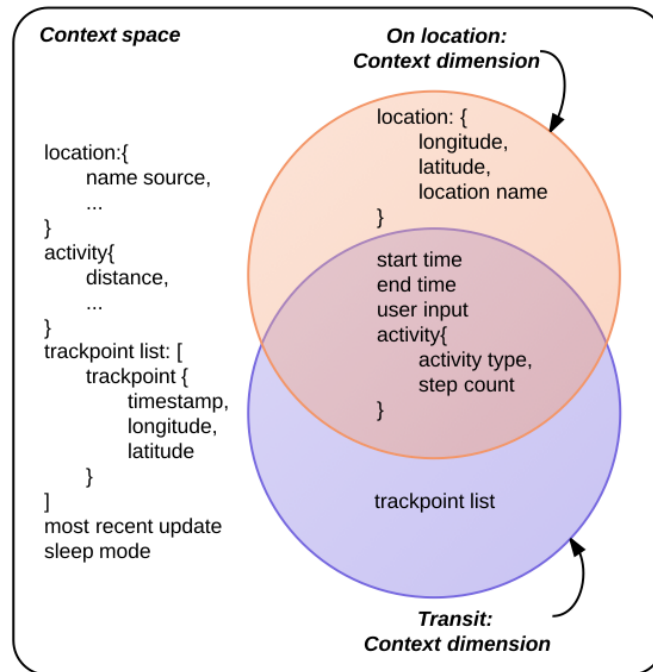


Figure 2.7: Relevant variables filtered from the context space into an application’s context dimensions

and Moves data, and the unique context variable here is “user input”. When the boolean “user input” is True, it implies highest confidence in the user-supplied data.

2.4.2 Context engine organization

From observation, we have a sense of systematic errors from Moves - for example, the GPS tracking has high latency when waking from sleep and misinterprets movement upon catching up to tracking. Intermittently, it derives the step count from the distance traveled divided by a universal average stride length (thus undercounting steps for a shorter person). Both devices are susceptible to losing battery, losing data due to connection or cloud service failures, or simply being misplaced by an absent-minded user. Fitbit has a tendency to misinterpret miscellaneous movement (typing, doing dishes, etc.) as steps, while missing less easily discernible steps (carrying groceries, hands in pocket). By taking both devices into the context-aware application, and leveraging the fact that Fitbit and Moves give

slightly overlapping but different information, we have the opportunity to weigh their data given contextual reliability and fall back to one if the other goes offline. We classify segments of a user’s day coarsely based on data from Moves. When the user is in one place, the phone records data as a location and associates a list of activities with it. However, when the user is in transit, the system records a single movement activity (transit, walking, cycling, etc.) with a series of trackpoints tracking the movement. The two context dimensions in Figure 2.7, both extracted from the same context space, reflects this. The two context dimensions allow the application to parse and reason across two different scopes of data. The Bayesian network in Figure 2.8 describes the relationships between observable data from Fitbit and Moves, each node representing some variable summarizing the user’s activity. Using the specification from Fitbit’s API, we classify the step counts per minute into low, medium or high activity levels. The context engine trains the network on each incoming data segment and constructs the associated conditional probability based on whether Fitbit or Moves is more accurate for each segment (relative to the ground truth). The dependencies of the nodes are determined by the classification of activity levels (low, medium, high), and the perceived Moves activity (on location, walking, or using transportation). Each instance where Fitbit (or Moves) agrees with the ground data (within margins of the activity level thresholds) increases the weight of the edge leading from that node to the “Fitbit is accurate” node (or “Moves is accurate” node).

In Figure 2.8, the edges connecting dependencies were manually assigned, and only their weights were learned. Without a priori knowledge of the relationship between Fitbit and Moves, the Bayesian network would start as a fully connected graph, and the learning algorithm would eventually prune the edges which do not in fact connect dependent nodes.

2.4.3 Results

We compare the accuracy of these three data streams - Fitbit, Moves, and estimation learned on Fitbit and Moves. The edge weights of the Bayesian network are trained on two days’ worth of data. Since each day is naturally divided into

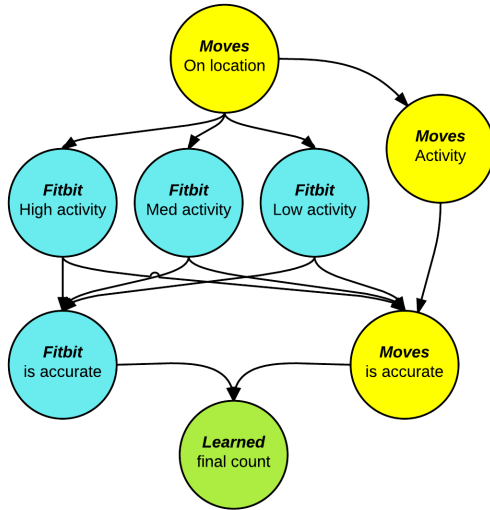


Figure 2.8: Bayesian Network to learn accurate step count from Fitbit and Moves data

a different total number of segments depending on how often the user changes activities or locations, this ranges between 37-45 segments. After the learning phase, the cross product of all these variables gives a confidence for each data stream that selects the most trusted source for each segment. The final daily total of step counts is compared to the “ground truth” for that day’s total, and the “accuracy” represents how closely the estimated step count falls relative to the actual number of steps taken. We define accuracy as the probability of designating the output as “correct”, under a Gaussian distribution. The mean is selected as the “ground truth” data and the standard deviation set such that the $\pm 15\%$ range has a 90% accuracy. This margin is given because small discrepancies in the absolute number of steps counted across a day (which ranges in the thousands on average) should be reasonably expected.

Figure 2.9 shows the performance of our learning algorithm on a small training set, for three representative days. In Sample 1, even though both Fitbit and Moves are grossly inaccurate in counting total steps for the day, the other contextual data they provide about a user’s location and activity type can help greatly in learning which one to trust for a particular segment in the day. Thus,

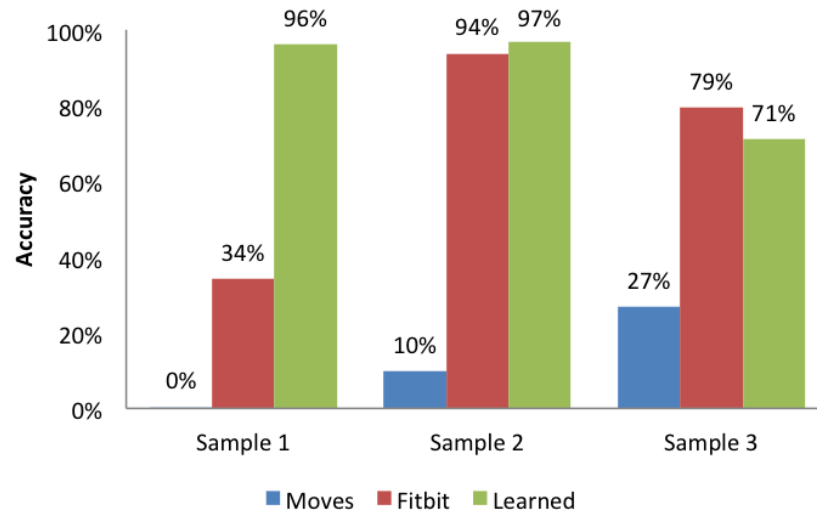


Figure 2.9: Bayesian Network to learn accurate step count from Fitbit and Moves data

for each segment, as long as one of them reports accurate data, and the learning algorithms picks the correct one, the final count can still be very accurate. In Sample 2, the Fitbit stream is highly accurate, while Moves is not. While the node weights of the Bayesian network represent confidence in a data stream, they do not guarantee accuracy by correctly choosing to trust Fitbit in most segments and Moves in a minority of segments, the final step count is still highly accurate. On the other hand, the penalty of trusting in the wrong stream can negatively impact the learned result, as shown in Sample 3. Such an “anomalous” day may include long periods of time spent underground where GPS localization is ineffective, or where the user is engaged in vigorous and repetitive activity while standing relatively still, such as organizing equipment in a lab environment.

By learning when to trust and when to discard particular data streams based on the context of that data collection, we can produce an output stream with accuracy much higher than either of the individual sensor streams alone. In this case study, we see an average 60x accuracy improvement over using a single stream of data when learning over just 2 days.

2.5 Conclusion

The Internet of Things represents the next iteration of ubiquitous computing, incorporating heterogeneous sensing and actuation in a web-based backend. With the vast data collection and application execution that dynamically changes with contextual environment, there is a growing need for scalable ways to deal with this volume of data and computation. The conventional method of independent, ad-hoc application development includes many redundancies in both sensor communication and data manipulation. We have shown that a hierarchical division of labor is provably more scalable, with a controlled accuracy tradeoff.

However, the onset of significant commercial development of IoT devices communicating with divergent backends and in constantly-changing formats severely complicates the main goal of the IoT: context-aware computing. Upon reviewing the related IoT work, we found a key component of the new IoT middleware missing: the ability to unify and operate on ever-changing sources and context in a low-overhead manner. To resolve this issue, we developed the ontology-driven context engine. Leveraging and expanding the XCML context ontology, we unified data translation, filtering, and preprocessing into a format readily readable and expandable by context engine implementations. We developed a methodology to implement context-aware applications: upstream context engine instances for flexible context preprocessing and extension, and downstream instances for application logic and actuation. Using a base context engine, we implemented an application that demonstrates the ability to learn and follow different paths of reasoning based on available data.

In the next chapter, we discuss how machine context can be leveraged to optimize a single node of the hierarchy. In the cloud, high performance servers support the massive volume of data processing. Within a single node, there are many sources of context that describe the efficiency of the underlying hardware resources, including ambient temperature, cooling capacity, and availability of compute resources like CPU cycles and data bandwidth. We will leverage these context parameters to optimize various metrics of execution such as energy consumption and speed. In Chapter 4, we will move beyond the single node and consider the

cooperative execution of devices with different capabilities.

This chapter contains material from “An Ontology-Driven Context Engine for the Internet of Things” by Jagannathan Venkatesh, Christine Chan and Tajana Rosing, which appears in UC San Diego Technical Report CS2015-1009, 2015. The dissertation author was one of the primary investigators and the second author of this paper.

This chapter contains material from “A Modular Approach to Context-Aware IoT Applications” by Jagannathan Venkatesh, Christine Chan, Alper Sinan Akyürek and Tajana Šimunić Rosing, which appears in Proceedings of the International Conference on Internet-of-Things Design and Implementation (IoTDI), IEEE 2016. The dissertation author was one of the primary investigators and the second author of this paper.

Chapter 3

Device context: performance, power and thermal controls

Datacenters are the conventional backbone of processing in the cloud, comprised of a complex hierarchy of devices that support a wide variety of application domains. Due to the huge scale of operation, power, cooling, and combined energy cost management of these datacenters at various levels of hierarchy are the subject of many studies, both academic and commercial [59, 60]. For example, a high-end database server is equipped with many sensors that inform the system of many aspects of its hardware state, including granular power sensors, multiple temperature sensors at different locations, hard drive health, and more, all managed by a dedicated side-band processor. It would be infeasible for each server's fans, core power modes, and data bandwidth to be managed individually from the whole datacenter's perspective. However, if each server in a rack can summarize its operating context in relation to performance, this may enable the hierarchy of computing elements to incorporate its hardware context in a scalable manner. Structure is needed to sense and manage that context so that it does not interfere with the intended application operations.

In this chapter, we discuss how a single server node in the IoT monitors and manages its own operating context. To maintain the integrity of hardware components, processors manage workload scheduling and on-chip thermal management dynamically, while powerful server chassis fans work in combination with

the building HVAC or passive heat removers to maintain a thermal set point [61]. The power consumption of these fans grows cubically with the speed settings [62]. There are efforts to reduce datacenter cooling power by reconfiguring rack organization (e.g. hot/cold aisles [63]), creative chillers, and task allocation across multicore processors and even room placement [64] [65]. In an individual server, the largest power consumers are the processor chip and cooling subsystem (we measured 37% and 29% respectively). In a typical datacenter of 20,000 servers at a 1.5 power usage effectiveness (PUE), 24% of its monthly budget goes towards the utility bill[66]. Any improvement in the server energy consumption can dramatically lower power budgets, improve service reliability in case of power instabilities, and ultimately improve profit margins for the datacenter operator[67].

Database query software is written assuming that underlying hardware resources including CPU cycles, memory access and IO bandwidth are fully available. However, the operating context may limit this resource availability - e.g. power caps or thermal constraints are aspects of the physical environment that can limit software application behavior. In the following sections, we address a critical source of data performance degradation that is often neglected. Mechanical disturbances generated by the cooling system can cause temporary crashes or misses in spinning storage systems, which in turn inflate workload execution times and server uptime electric bills [11]. Even small disk latencies can cascade into large effects on final application performance - 5% disk latency can lead to over 40% slowdown in the total performance [13], while others have measured a 60% disk delay leading to 170% final delay in a database query execution [11]. These transient problems can be very difficult to diagnose in deployment or to replicate in a lab setting without the correct surrounding environmental factors. They are also neglected by existing thermal models and management policies.

Since current enclosure, cabinet and raised-floor room designs are insufficient in eliminating all vibrations [13], we turn to software-based detection and control. Enterprise servers already have a side-band “service processor” to monitor hardware sensors, execute power management, and log maintenance events, accessed via the Intelligent Platform Management Interface (IPMI). This would be

an appropriate platform to detect vibrations from the fan controller and respond accordingly. Orthogonal to mechanical upgrades, a software update also allows for fast and low-cost adaptation In the event of hardware configuration changes.

Fortunately, though datacenter workloads can be highly demanding of the hardware platform, and increasingly complex in terms of software optimization, they are also fairly well-known and predictable at a large scale [14]. By modeling the workloads in terms of their resource consumption, server operators can predict the application’s needs and manipulate the operating conditions such as temperature and core availability to improve performance. Most current strategies focus on manipulating processing resources such as multi-core task scheduling and frequency scaling. Notably, we approach the server efficiency problem by targeting the cooling-performance relationship. Our results are based on real physical telemetry of a late-model multi-threaded, multi-core server processor running a standard database benchmark suite (TPC-H [15]). The proposed server model and simulated control policy demonstrates up to 3.3x speed up over state of the art policies, leading to 19-65% energy reduction while still meeting thermal constraints. We hope these insights can help server designers, database administrators and data-center operators design and maintain more energy efficient systems for database applications.

3.1 Background and related work

Here, we give an overview of prior work done in three main research areas that involve server data performance as supported by hard disk drives. First, we identify representations of database applications with respect to their hardware utilization and reliance on data accesses. Second, we discuss existing physical and mechanical designs that affect hard drive performance in datacenters. We close by summarizing state of the art server management strategies, in particular power, thermal and cooling policies, and discuss our contributions to the area.

3.1.1 Database Workload Modeling

Database performance can be quantified and analyzed many different ways. End-to-end metrics such as total execution time are used to signal critical failures or crisis status [14]. For a more detailed understanding, applications can be divided into phases of software demands and elemental operations, but the complexity of database platforms necessitate the use of machine learning techniques rather than relying on expert design[68]. To predict total execution time of separate database queries using design-time characteristics, some have found that clustering techniques out-perform regression for multi-variate feature sets [69].

An orthogonal method of representing workloads is to inspect their interactions with hardware resources, which lends more naturally to hardware management policies. For example, a particular query behavior can be described with microarchitectural statistics (e.g. IPC and cache-miss), and transitions between behavior can be modeled with a Markov decision process, leading to thermal management decisions [70]. A query can also be described in terms of the size, location, and frequency of disk accesses [71, 72]. However, these static approaches neglect how additional interactions with the physical environment can change software behavior and resulting execution. These policy solvers may choose generally optimal execution plans given ideal drive performance, but miss dynamic degradation in drive throughput due to external mechanical disturbances, resulting in inaccurate performance modeling.

3.1.2 Mechanical considerations: shock and vibration

Vibrations and shock can have significant detrimental effects on hard disk drive operation [73, 74]. Many server vendors and large customers have made design improvements in drive enclosures [9, 75], server chassis [76], racks[77], and even the raised-floor datacenter rooms and buildings[13, 78], to preserve drive data integrity and performance in the face of vibrations. These improvements include sturdier material choices, physical re-organization of vibration sources (fan arrays, hard drives), and signal processing to cancel sensed vibrations. The vibration protections only target well-known sources such as the spinning hard drive motors

themselves, physical drops, and HVAC building cooling systems [79]. Their success metrics are geared towards lowering hard drive failure rates, generally caused by head crashes (i.e. when the read-write head makes contact with the disk platters, causing irreverible damage)[80]. Liquid cooling [81, 82] would reduce mechanical disturbances to the system, but are prohibitively expensive for today’s commodity systems. To our knowledge, there are no solutions currently in the market that account for the persistent, dynamically changing vibrations generated from fan cooling within the server. Concurrently, there are no metrics that quantify vibrational effects in terms of instantaneous but non-lasting drive performance degradation.

3.1.3 Power, Thermal and Cooling Management

The largest power consumers in servers are the processor chip and cooling subsystem. Fans have a cubically growing motor power consumption profile [62], while processor leakage power grows quadratically with increasing temperature (i.e. lower fans). For a fixed workload, there is a single optimal point where some fan speed achieves the lowest combined processor leakage power and fan motor power [83]. High, fluctuating temperatures are correlated with poor drive reliability [84, 85]. We show that the observed disk performance degradation is likely due to interactions with the cooling system, and not temperatures *per se*. To reduce the thermal load, the processor can gate the clock or perform dynamic voltage and frequency scaling (DVFS), at the cost of direct reduction in performance [86]. In workloads where the bottleneck lies outside of the core (in memory, for example), frequency scaling may have unexpected effects - what is optimal from the core’s perspective may not yield desirable results for the larger system [87].

For cooling, a standard industrial policy proportional-integral-derivative (PID) control [88], which some newer solutions are based on [89]. Task assignment can be done with some awareness of datacenter physical layout [90, 91], but these techniques fail to account for the cooling interactions at the lower level of server fans. Traditionally, the thermal effect on performance is only measured in terms of core compute speed [86], even by studies of disk-heavy database query

performance [92]. We assert that even when thermal is not considered an issue by conventional standards (e.g. high temperature), data performance can still suffer, because cooling has a significant side-effect - existing work has shown that internal server fans can negatively impact drive throughput by anywhere between 60-88% [11, 93]. A comprehensive understanding of the system enables model-predictive control to maintain a stable system state and make guarantees about system behavior. Relevant models include thermal circuit simulators [94], time-based temperature predictors [95][96] or workload-based temperature prediction [89]. Well-defined hardware configurations and operating ranges lend themselves to control-theoretic solutions for cooling decisions [97]. If performance and accuracy constraints change in the field, application-level integration can make system management more efficient and stable [98]. While these strategies leverage the tradeoffs between processor cooling power consumption, and core execution speed, they neglect the relationship between cooling and application performance. Thus, they may yield subpar performance for data-intensive workloads.

In comparison to solutions in the current state of the art, our contributions are three-fold:

- We *quantify the fan-disk interactions* that cause difficult-to-diagnose performance degradation in data-intensive workloads. Our measurements taken from in a real operating datacenter and lab settings show up to a 88% hit on disk write throughput when fans are at their maximum setting.
- We develop a *model of a server to represent dependencies between server performance and physical effects*, including power, thermal and cooling. Using analytical models as opposed to conventional simulators, we enable formal optimization of the overall system.
- We use convex optimization to design a *proactive policy that performs optimally efficient fan management*. Compared to existing controllers and those proposed in literature, our model-predictive control yields provably higher energy savings (up to 80%) and faster workload completion times (up to 70%) while meeting temperature constraints.

The rest of this document is organized as follows: Section 3.2 documents our measurements of cooling and performance interactions in a real, operating datacenter server. In Section 3.3, we develop a system model based on physical measurements of thermal, cooling and disk performance. Section 3.4 formulates and solves the hardware thermal management problem optimally for runtime energy. Finally in Section 3.5, we evaluate how the optimal hardware management policy performs as compared to current state of the art.

3.2 Measuring cooling and performance interactions

First, we present a methodology for characterizing any server disk’s response to vibrations that it may encounter in a typical datacenter. Our parametric characterization suite of experiments measures the vibrational sensitivity of a diverse set of disks. In all experiments, the ambient temperature is tightly controlled, isolating any drive performance effects to mechanical sources.

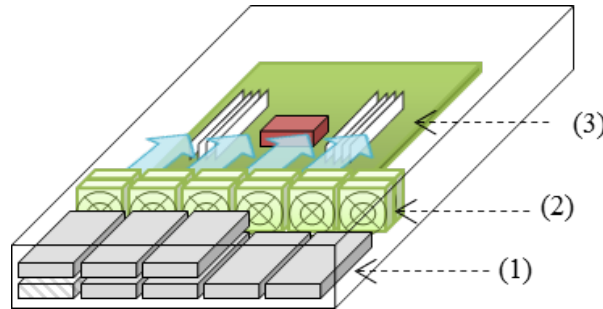
3.2.1 Measurement methodology

We recorded vibrations from several points on server racks in an operative datacenter, using tri-axial accelerometers. These vibrations can be quantified on two different axes - the overall acceleration or total energy, called the *amplitude*, and the *frequency* or component frequencies of the signal. The amplitude is a scalar calculated from the power spectral density (PSD) function, or the root mean square value of multiple signals, in units of *grms*. The frequencies are in *Hertz*. We found that a rack server in an operative datacenter will typically experience vibrational frequencies ranging from 20 to 2000Hz, and amplitudes from 0 to 2 grms.

Within the measured parameters, the vibrations are reproduced in a lab environment with an Unholtz-Dickie model K170 electrodynamic programmable vibrational table [99]. Specifications of the platform and environment are listed in Table 3.1. The test server is mounted on top of the table, as we monitor the

Table 3.1: Test server specifications

Processor	8 cores @ 3.0GHz, 40nm
Memory	16 x 16GB DIMM
Operating system	Solaris 11.1, firmware 8.2.1
DBMS	Oracle 11.2.0.3
Idle processor power	75W
Idle server power	267W
Typical server power range	330-600W
Maximum air flow	145 cubic feet per minute (cfm)
Maximum fan power	180W
Room temperature	25°C
Chassis internal temperature	30°C

**Figure 3.1:** Server organization with (1) hard disks and (2) fan assembly directing airflow towards (3) the motherboard.

same points where it would have come in contact with a rack mount to ensure that the vibrations are faithfully transmitted. The test server has a commonly used single-socket, multi-core and multi-threaded processor. It has two memory sockets on either side of the processor, 6 fan modules, and 8 disk drive slots. The drive slots are loaded with a broad range of disk models as described in Table 3.2, including the commodity SATA drives, enterprise SAS drives, and solid state drives (SSDs). Since SSDs do not depend on moving parts to read data, they act as our control drives - as expected, they were impervious to vibrations and their results are omitted for clarity.

Table 3.2: Disk drive models specifications

Code name	Manufacturer	Technology	Spin speed (RPM)
FUJSATA	Fujitsu MHY2200BS	SATA	5400
HITSATA	Hitachi Travelstar E5K500	SATA	5400
SEASAS A,B	Seagate Savvio 10K.3 ST930003S	SAS	10000
HITSAS	Hitachi Ultrastar C10K600	SAS	1000
INTELSSD	Intel 710 SSDSA2BZ300G3	SSD	-

Table 3.3: Overview of measured disk drive behavior

Code name	Write speed at min fan (MB/s)	Write speed at max fan (MB/s)
FUJSATA	31.2	6.2
HITSATA	37.0	14.6
SEASAS A,B	72.2	72.2
HITSAS	81.6	81.6
INTELSSD	206.8	206.4

Fan speeds are controlled through pulse width modulation (PWM). This electrical “pulse” does not contribute to mechanical vibrations. The available fan speeds are 0-100% at increments of 10% (given some tachometer error) but in practice, fans are observed to be at least 50% when a server is active. We can temporarily override the built-in fan control algorithm to manually set fan speeds via the Intelligent Platform Management Interface (IPMI).

To expose the true disk behavior, we disable the buffer cache that would have hidden disk access latency from the user. We run a pure I/O generator which issues random sustained writes to the disk, utilizing 100% of the I/O bus bandwidth. We quantify the effect of fan speeds on disk performance in terms of data write throughput. The minimum and maximum write speeds measured while the server is experiencing no external vibrations are reported in Table 3.3. In Section 3.5, we will evaluate realistic database benchmarks with more variable I/O bandwidth requirements.

3.2.2 Amplitude test with random frequencies

We study the effect of external vibrations varying in “amplitude”, defined as the total combined signal strength of each component signal in the frequency profile. The vibrations are generated on the shake table while fan speeds are set to 50% PWM, and accelerometers are placed at rack-contact points on the server to verify the total amplitude of vibrations delivered - this is why the test points do not exactly line up at increments of 0.2. We ran experiments on the disks under profiles that cover a different collection of frequencies (20-800Hz in Figure 3.2 and 20-2000Hz in Figure 3.3).

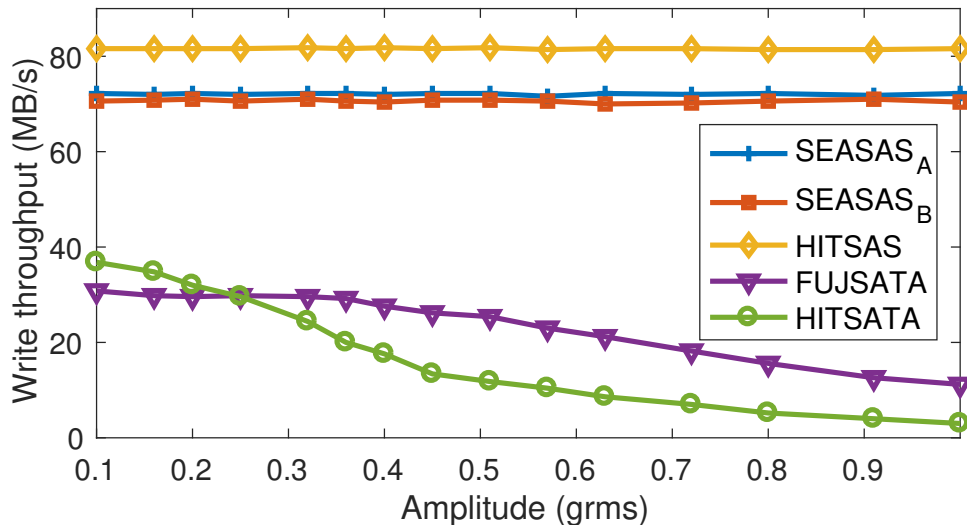


Figure 3.2: Throughput dependence on vibrational amplitude, component frequencies ranging 20-800Hz

Although lower throughputs generally follow higher amplitudes, the sensitivity curve varies across hard drives and across frequency profiles. Of the two SATA drives spinning at the same speed (5400 RPM), FUJSATA performs better than HITSATA for $grms < 0.2$. At $grms = 0.63$, HITSATA writes at 8.6 MB/s in the first profile and 3 MB/s in the second. SAS drives are more resilient, but they start showing signs of performance degradation around $grms = 1.27$. The largest drop among the SAS drives is 10.5% on HITSAS and the largest drop among the SATA drives when HITSATA stalls at 0 MB/s at $grms = 1.8$.

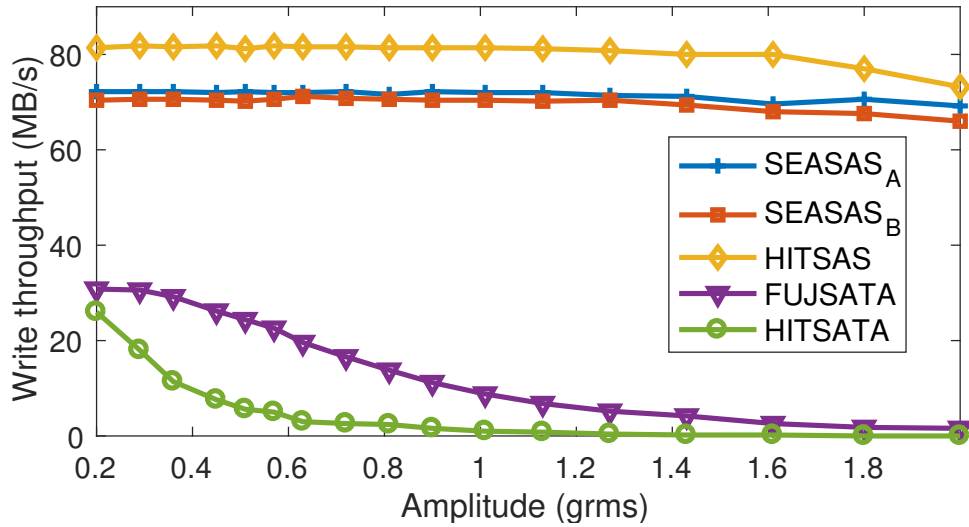


Figure 3.3: Throughput dependence on vibrational amplitude, component frequencies ranging 20-2000Hz

3.2.3 Frequency test with fixed amplitude

This experiment characterizes the hard disk response to external vibrations of varying frequencies. From on-site measurements at datacenters and observing Figures 3.2 and 3.3, we fixed the amplitude of vibrations at 0.17g, where drives performed well in general, but had the potential to experience throughput degradation. We sweep through frequencies between 20 to 2000Hz and monitor the change in disk throughput (Figure 3.4). The response to different frequencies is irregular and there is neither a distinct “zone” of performance degradation, nor any obvious ratio between the frequency value or write throughput. Certain frequencies that cause performance degradation have a very narrow band. Even though more obvious degradation is seen at higher frequencies, there are narrow bands where disk performance returns close to its ideal. SATA drive throughput drops to 0MB/s at various points, while SAS drives fluctuate by 1-2%.

3.2.4 Fan sweep test

Here, we isolate the effect of internal vibrations generated by the full range of possible fan speeds by bolting the server to the stationary shake table. With

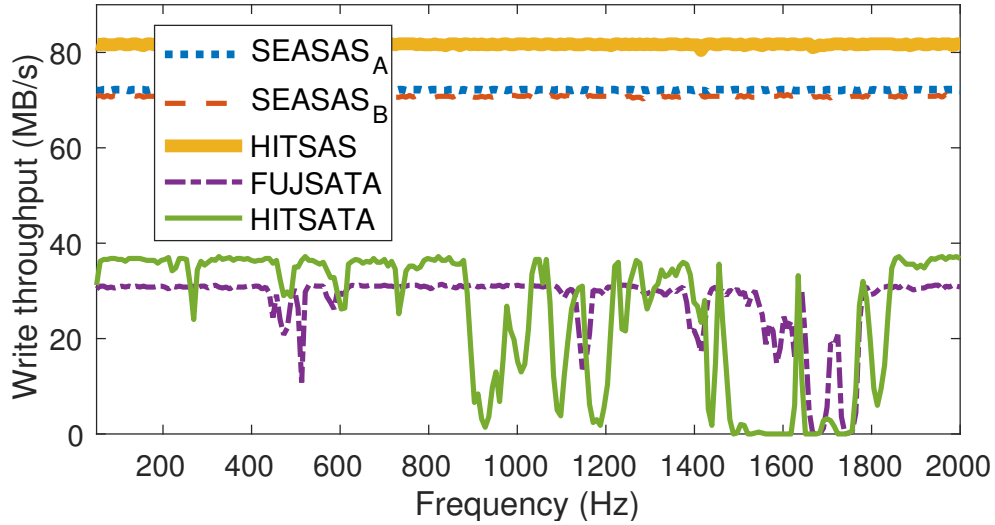


Figure 3.4: Throughput dependence on vibrational frequency with amplitude fixed at 0.17g

each change in stimuli, the disk drive throughputs take 20 seconds to respond. In our experience, the processor shuts down within 10 seconds of turning off the fans, while self-reporting on-die temperatures up to 91°C immediately before crashing. Consequently, it is challenging to accurately measure system characteristics in fine-grained steps at low fan speeds. We step through fan speeds from 100% to 0% PWM at 10% step sizes to obtain stable results. Figure 3.5 shows the average degradation of write throughput on fan speeds, normalized to the maximum throughput measured on each disk. There are no observable vibrational effects below 50% PWM. SATA drives show the most throughput degradation, down to 35% and 12% of their maximum value. The SAS drives show degradation only at the maximum fan setting - HITSAS loses about 2% of its throughput.

With these experiments, we have characterized the relationship between hard disk performance and vibrations they experience. The possible effects of vibrations external to the server (represented by the amplitude and frequency sweep tests) are not easily mitigated - short of expensive hardware rehaul. Although the mechanical study of these drive differences is out of the scope of this work, we do observe that enterprise SAS drives are consistently more resilient than commodity SATA drives. Enterprise drives tend to have a heavier and more stable chassis,

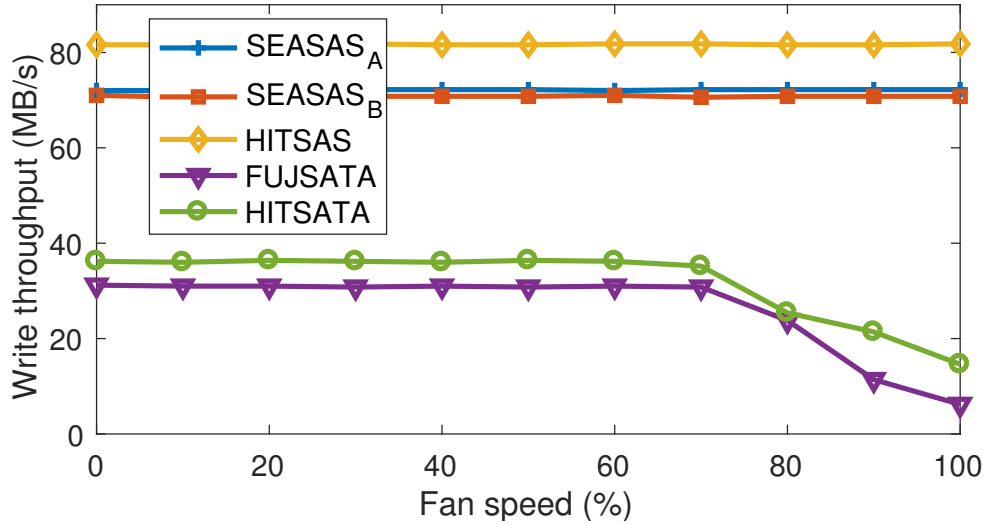


Figure 3.5: Throughput dependence on fan speeds (no external vibrations)

more expensive servos controlling the read/write head, better spindle motor shaft capturing, and better air flow control, all of which improve resilience against environmental vibrations [9]. Since our motivation is to find solutions for in-server hardware management, and the majority of deployed drives in cost-sensitive datacenters are commodity SATA, we choose to focus on the relationship between internal fans and SATA disk performance. Based on measurements presented here, this leaves a 65-88% drive performance gap that we hope to close with intelligent fan control policies.

3.3 Integrated system model for performance, power, thermal and cooling

In this section, we describe models of workloads that we use for optimizing power, thermal and cooling systems. These models are essential for enabling the optimal hardware manager we present in the following section. They are based on physical instrumentation and software tracing of database workloads run on the server described in Table 3.1.

Figure 3.6 shows the models developed based on the data from the actual

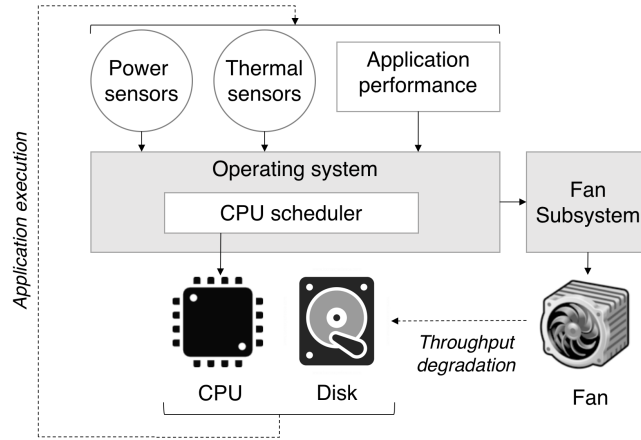


Figure 3.6: Subsystems and dependencies in the server model

processing core, hard drive, power and thermal sensors, and fan cooling subsystems. Application performance is modeled based on a standard database benchmark suite, with a unique but predictable pattern of hardware resource utilization. Power and thermal models estimate the thermal response and heat exchange between the processor chip and the fan cooling system. We use measurements of disk performance from the previous section to model the effect each cooling decision has on IO throughput and system performance. All these components are combined to represent the behavior of a high-end server running typical database workloads.

3.3.1 Workload representation

We chose the TPC-H benchmark suite to represent data-intensive workloads that commonly run in datacenters [15]. TPC-H is a decision-support benchmark consisting of 22 queries representing different business-oriented queries on large datasets. Depending on the size of the dataset, the entire suite can take on the order of hours or days to complete. The benchmark specification states that performance is defined by the query throughput (i.e. query-per-hour) for a fixed dataset, processor parallelism and memory size. For each individual query of a 40GB database size, with 4 parallel core threads allowed, and 128GB RAM, we calculate performance as the execution time required. We extract the model for

each query using the single-user “power test” scenario, as opposed to the “throughput test” which represents a multi-user environment.

Other researchers have had success categorizing database workloads solely on their observed disk activity fluctuations, without tracking the semantics at an application level [71, 72]. Since different database operations in a single query can activate parallel cores, memory, and the IO bus in different patterns, we extend the model to represent a more comprehensive view of the system operating constraints, using a database manager that enables parallel queries where appropriate. We monitored the system using built-in trace commands (*mpstat*, *iostat*, *vmstat*) and a database monitor Oracle Enterprise Manager. Resource utilization can be described by vectors in a multi-core scenario in the form $\langle c_0, \dots, c_{N-1}, io \rangle$ where c_i represents the utilization between 0-100% for physical core i out of N cores, and io represents the percentage of maximum IO bandwidth (machine specification is 300MB/s).

We observe similarities among the observed utilization points and we model these similarities as system states. We use k-means clustering to quantify these similarities, where each cluster corresponds to a distinct system state. When determining a cluster for each observation, distortion is defined as the sum of the squared distances between each observation vector and its closest centroid [100]. The distortion decreases non-uniformly as the number of clusters increases. The elbow test described in [101] determines an appropriate number of clusters (k) to determine the point that gives the most benefit (in terms of reducing distortion) relative to an increase in clusters. Consider the decreased distortion per increment in k as the quantifiable benefit of increasing k . Then the first derivative represents the rate of gain in benefit. Furthermore, to find the k setting that yields a highest gain in benefit vs. increase in k (and consequently, lower benefit for $k + 1$) we can take the derivative of the rate of gain. Thus, identifying some minimum in the second derivative shows us an appropriate k using the “elbow” test.

With this method, we find that four clusters provide a good tradeoff between number of clusters vs. distortion value. With all execution grouped into one of these four clusters, on average each query can be described with a chain of 97

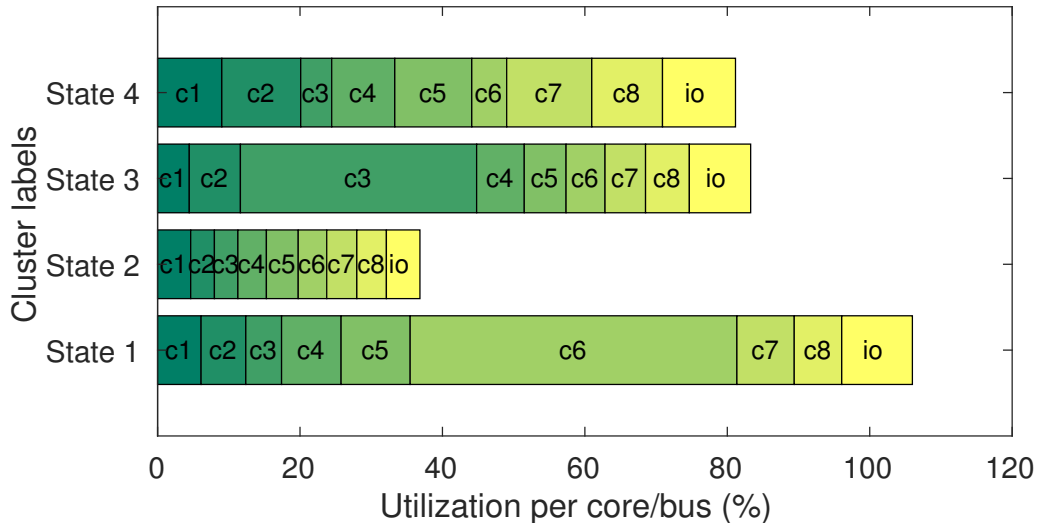


Figure 3.7: States as defined from CPU and IO bandwidth utilization vector clustering

states. The chosen centroids are visualized in Figure 3.8. At the high level, most of the TPC-H queries consist of reading data from separate tables in parallel before sorting and/or joining them. Parallel operations show multiple cores being active - e.g. State 2 may be issuing multiple small read requests, while State 4 is issuing large bulk transfers. The joining and aggregation of parallel work present as one particular core being very active and others being relatively idle (e.g. State 1 and State 3).

The average length of time spent in each state per occurrence varies between states. The performance traces were collected while the server was in a very cool room, so the processor stayed cool even with fans at low speed. This measured time is considered the “ideal” time since there are no vibration-induced delays. The duration of a state may be extended dynamically if the state is IO-dependent and fan speeds are high - as measurements show in the previous section.

For the rest of this chapter, we consider each query as a series of intervals, where each interval executes a single workload state. For example, the shortest *query 2* is represented with 6 states of various lengths, while the longest *query 1* has 330 state changes. The average query length across all 22 queries is 120 states where the standard deviation is 88 states. For each single workload state, since

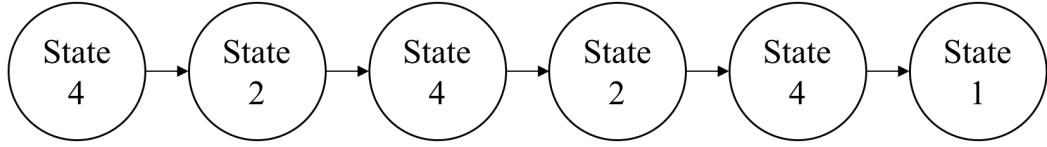


Figure 3.8: Model representation of query 2, with a chain of 6 states

the hardware utilization patterns are fixed, their power and thermal responses will also be predictable. The power and thermal profiles will change as the workload state changes. They will be evaluated on an interval-by-interval basis to determine necessary changes in cooling control.

3.3.2 Power model

In most server systems, including ours, the processor and the cooling system represent the majority of total server power consumption and have a wide dynamic range [102]. The processor’s high power density also dominates dynamic changes in chip temperature at runtime. We choose to focus on modeling these two components accurately.

Processor power dissipation is comprised of dynamic power - dependent on workload state w - and static “leakage” power - dependent on temperatures \vec{T} . Our processor’s maximum power dissipation by design (the “thermal design power”) is 240W; we observe a typical range of 80-200W consumed by the processor, depending on utilization. We estimate dynamic power required to execute each of these states by linearly scaling the dynamic range of each core by the utilization factor [103]. For each workload state defined in the previous section, the utilization level is fixed, hence the dynamic power consumption $\varepsilon_{dynamic}(w_i)$ is also fixed. We approximate the static power $\varepsilon_{static}(\vec{T})$ as linearly dependent on temperature. For the typical range of operating temperatures in a server, this has been shown to have an error less than 5% [104].

A commonly used cubic fan power model is presented in [62]. To use this model, we require a reference constant r_f that can be easily measured once for

any given server, to obtain a known fan speed f_r and its corresponding power consumption level p_r . Our final fan power model is defined as:

$$\begin{aligned} \varepsilon_{fan}(f) &= r_f f^3 \\ \text{where } r_f &= \frac{p_r}{(f_r)^3} \end{aligned} \tag{3.1}$$

For the purposes of the static power model, we will assume that the temperature does not change significantly over the course of a workload interval, and that fan speeds are only reevaluated once per workload interval. Thus, each interval power depends on the single workload state, the starting temperature, and the set fan speed. The processor- and fan-centric power model can then be calculated at runtime as:

$$\begin{aligned} \varepsilon_{power}(w, \vec{T}, f) &= \varepsilon_{dynamic}(w) \\ &+ \varepsilon_{static}(\vec{T}) \\ &+ \varepsilon_{fan}(f) \end{aligned} \tag{3.2}$$

3.3.3 Thermal model

This section discusses the major heat producing and extracting components in our server, which can be represented with electrical analogies [94][97]. Components that actually consume power (such as the processing cores and L2 caches) behave as heat sources, modeled as power sources in the circuit. The effectiveness of the heat sink in dissipating extra heat relative to any given power consumption is determined partly by its materials and surface area A (conductive resistance), but more dynamically by the airflow V passing through it (convective resistance, with δ between 0.8 and 1. The air flow rate increases linearly with fan speeds as dictated by the duty cycle of the fan motors (PWM), while the driving fan power P_V rises cubically with air flow rate [62].

$$R_{conv} \propto \frac{1}{AV^\delta} \tag{3.3}$$

We extract an analytical model based on initial simulations run on the widely accepted HotSpot tool [94]. The goal is to identify a differentiable model

to be used later in the formal problem formulation in Section 3.4. While retaining the RC response, we simplify the model to only model the runtime variations of the hottest core and the fan cooling capacity. The underlying hardware has a thermal response time constant τ that is dependent on the given fan speed. Each workload has a time-invariant steady state temperature T_{ss} if it is allowed to run indefinitely at a given fan speed. Thus, beginning at some initial temperature T_0 , for any single workload executing for a certain amount of time t , a cooling-dependent time constant τ dictates how quickly the system approaches the steady state temperature T_{ss} . We obtained the actual values for T_{ss} and τ based on HotSpot experiments.

$$T(t) = T_{ss} + (T_0 - T_{ss})e^{-\frac{t}{\tau}} \quad (3.4)$$

Considered piecewise, the temperature at the end of each interval is the “initial” temperature of the next. The heat sink temperature falls linearly with the convective resistance - thus, temperature T_i decays as an exponential function of the given fan speed f_i . Recall that the actual length of each interval t varies, since we take into account both the nominal interval length of that particular state and any delay that the fan may incur. Equation 3.4 describes the instantaneous temperature at the end of interval i (i.e. the start of $i + 1$) as:

$$T_{i+1} = T_i e^{-\frac{t}{\tau}} + T_{ss}(1 - e^{-\frac{t}{\tau}}) \quad (3.5)$$

To verify, we compare 18 seconds of time series temperature data from our analytical model and a full HotSpot simulation. This evaluation now takes seconds instead of the minutes or hours that HotSpot simulation takes. Across the range of our expected workloads, the average error is less than 1%; thus we can approximate HotSpot’s validated model quite closely.

3.3.4 Cooling-vs-disk interaction model

In typical enterprise servers, vibrations are transmitted from the fan motors by mechanical coupling to the housing for the disk drives. In earlier work, an empirical curve was used in [93] without making assumptions about the exact relation

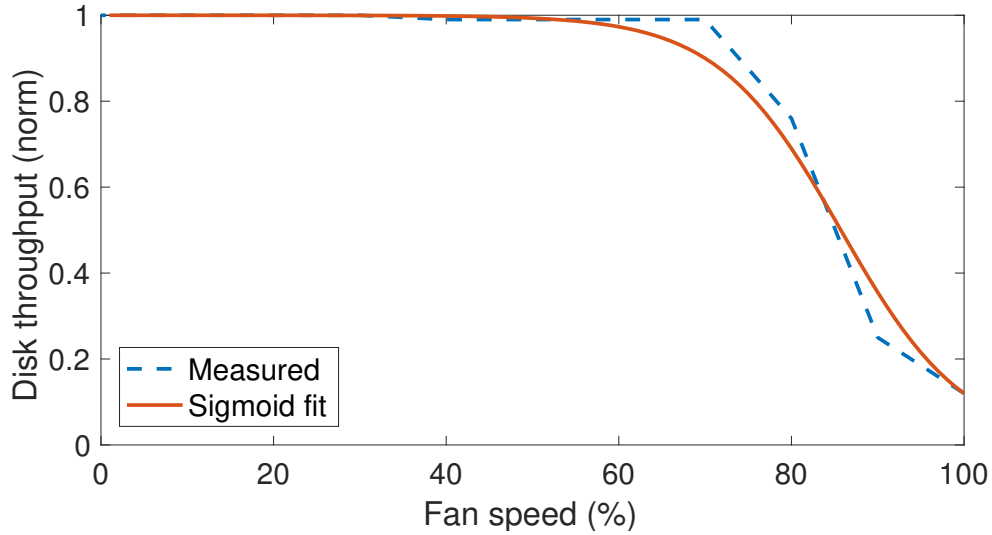


Figure 3.9: Fan speed-disk throughput interaction fit to a sigmoid function

between fan speeds and vibrational amplitudes. We find that this relationship can be generalized as a sigmoid function of fan speed f . We model the Fujitsu SATA drives in our particular server with $\alpha = 1034.65$, $\beta = 1033.65$, $\gamma = 8.04$, for an R^2 value of 0.98 and average relative error of 2.7%. This curve is shown in Figure 3.9.

$$\text{ThroughputFactor}(f) = \frac{\alpha}{\beta + e^{\gamma \cdot f}} \quad (3.6)$$

The workload model already contains information about the ideal runtime $c(w)$ of each workload state w , assuming full availability of the disk bandwidth (Section 3.3.1). It has been shown that throughput degradation has a superlinear effect that cascades into the overall application delay [11] [13]. In lieu of modeling memory hierarchy and database storage structures in detail, we make a conservative estimate for the relationship between available disk throughput and the minimal effect on overall application delay. We assume that the final delay caused by fan degradation is at least inversely proportional to the throughput. The resulting execution time $\varepsilon_{time}(w, f)$ needed for executing a single instance of a workload state at a certain fan speed is then defined as:

$$\varepsilon_{time}(w, f) = c(w) \cdot \frac{\beta + e^{\gamma \cdot f}}{\alpha} \quad (3.7)$$

For example, if only half the nominal throughput is achievable in a data-reading workload state, the state will take at least twice as long to complete.

3.3.5 Combining model dynamics

This section summarizes the notations and governing dynamics in the proposed system model. Each database workload is represented as a chain of workload states. A workload state w_i identifies the system resource vector during some execution interval i . The ideal average duration of each state $c(w)$ was found in Section 3.3.1. It is known *a priori* but may increase during execution if the workload w_i has high IO dependence and the fan speed f_i is high enough to affect disk performance according to the cooling-performance model (Equation 3.6). Thus, the time spent in any single state is a function ε_{time} of the workload and the fan speed - the ideal duration with perfect disk performance would be $\varepsilon_{time}(w_i, f_i = 0)$.

The *initial* temperature of any given interval i is a historical (fixed) value T_{i-1} from the perspective of that interval. The *final* temperature of that interval T_i is calculated according to the initial temperature, the dynamic power dissipated by the current workload state, leakage power dissipation due to the starting temperature, and the cooling capacity of some chosen fan speed. This becomes the *initial* temperature of the next interval $i + 1$. Our fan model assumes an accurate actuator that sets the fan speed according to the given control signal at each subsequent control step.

Energy consumed (ε_{energy}) is a product of power consumed (Equation 3.2: ε_{power}) and the period of time ε_{time} . Thus, the execution time D_N and total energy consumption E_N for a finite workload of N discrete system states are defined as:

$$D_N = \sum_{i=1}^N \varepsilon_{time}(w_i, f_i) \quad (3.8)$$

$$E_N = \sum_{i=1}^N \varepsilon_{power}(w_i, T_{i-1}, f_i) \cdot \varepsilon_{time}(w_i, f_i) \quad (3.9)$$

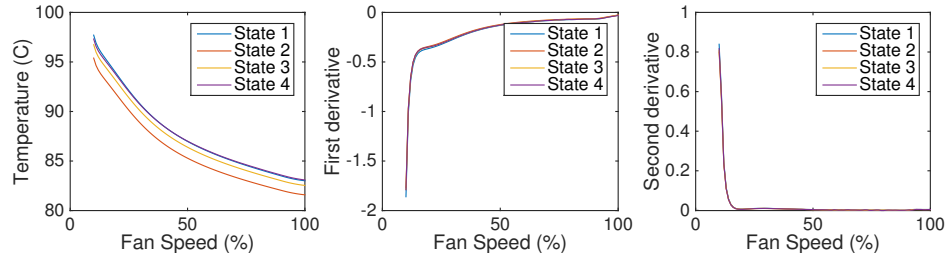


Figure 3.10: Numerical functions of temperature and derivatives, with respect to fan speeds

3.4 Energy-optimal cooling control

We consider a formal constrained optimization problem to define fan speeds that minimize the total cost of system operation. The generic cost C_N of executing N intervals is the sum of each interval cost ε_{cost} . Two costs of interest could be energy consumption E_N or execution delay D_N as described in the previous section. Whichever the cost in question, it must be minimized while keeping the temperature of all components under the threshold T_{limit} at all times, or formalized as:

$$\boxed{\min_f [C_N(f)] \text{ s.t. } T_i \leq T_{limit} \forall i \in [0, N]} \quad (3.10)$$

For example, to minimize the energy cost of execution Equation 3.10 would be rewritten with $C_N = E_N$, where the energy consumption E_N is an accumulation of power consumption values ε_{power} scaled by the interval lengths of ε_{time} . The cost can be defined using some combination of the thermal, cooling and performance models from Section 3.3, where each model depends on the fan speed selections and also each other.

For optimizations problems, Slater's condition states that any feasible solution to the Lagrangian Dual problem is also an optimal solution for a convex objective function[105]. Since the constraint function is the temperature, it suffices to analyze the various power components along with the time degradation dependency for convexity.

Lemma 3.4.1. *Processor and fan power consumption power are convex with re-*

spect to fan speeds.

Proof. Power consumption consists of three components: 1) dynamic power is independent of fan speeds; 2) fan power is a cubic polynomial of fan speed; 3) static power is a linear function of temperature.

Dynamic power is a function of core utilization, thus Equation (3.11) shows the independence of dynamic power from the fan speed selection.

$$\frac{\partial^n \varepsilon_{dynamic}}{\partial f^n} = 0, \forall n \quad (3.11)$$

The power consumption of the fan is a cubic polynomial with positive first and second derivatives.

$$\frac{\partial^2 \varepsilon_{fan}(f_i)}{\partial f_i^2} = 6r_f f_i > 0 \quad (3.12)$$

Static power has a linear relation with temperature, thus the convexity of static power is the same as of temperature. Consider the temperature function in Equation (3.5). The steady state temperature and time constants are numeric values obtained through experiments, so we prove the convexity of temperature through numeric differentiation in Figure 3.10. The second derivative is non-negative for all fan speeds, so temperature as a function of fan speeds is convex. \square

Lemma 3.4.2. *Execution delay is convex with respect to fan speeds.*

Proof. Total execution time of any given interval is a function of the workload being executed, which has a minimum delay ($c(w)$), and the fan speed, which may further slow down the workload. The delay function (Equation 3.7) is an exponential, and its second derivative:

$$\frac{\partial^2 \varepsilon_{time}(w, f_i)}{\partial f_i^2} = \frac{c(w)\gamma^2}{\alpha} e^{\gamma f_i} > 0 \quad (3.13)$$

Since α , β , and *gamma* are all positive constants, the second derivative is always positive, hence the delay function is convex with respect to fans. \square

Theorem 3.4.3. *Total energy cost of any application executed on this hardware platform is convex with respect to fan speeds.*

Proof. Energy consumption is defined as the product of power consumption and the total time that power is dissipated across. The final convexity of energy is calculated as:

$$\begin{aligned} \frac{\partial^2 E_i}{\partial f_i^2} &= \frac{\partial^2 \varepsilon_{power}(f_i)}{\partial f_i^2} \\ &+ \frac{\partial^2 \varepsilon_{time}(f_i)}{\partial f_i^2} \\ &+ 2 \frac{\partial \varepsilon_{power}(f_i)}{\partial f_i} \frac{\partial \varepsilon_{time}(f_i)}{\partial f_i} \end{aligned} \quad (3.14)$$

The first derivative of the energy cost function is always positive, such that energy monotonically increases with fan speeds. Moreover, since we find that the second derivative of the energy cost function with respect to fan speeds is always positive, the problem is convex. \square

In this section, we have proven that energy, power, and delay are all convex with respect to fan speeds, which allows us to next solve this optimization problem using its Lagrangian Dual.

3.4.1 Convex optimal formulation

We wish to minimize the cost C_N while ensuring that the system remains strictly **below** the temperature constraints (T_{limit}) for all time intervals. The Lagrangian with KKT multipliers is formulated as such, where each λ_i represents the constraint at interval i :

$$\mathcal{L} = C_N + \sum_{i=1}^N (\max(\vec{T}_i) - T_{limit}) \lambda_i \quad (3.15)$$

We need to solve for the fan assignment f at every interval j such that the Lagrangian is minimized.

$$\frac{\partial \mathcal{L}}{\partial f_j} = \frac{\partial}{\partial f_j} (C_N + \sum_{i=1}^N (\max(\vec{T}_i) - T_{limit}) \lambda_i) = 0, \forall j \in [1, N] \quad (3.16)$$

The total cost C_N is a summation of all interval costs, and is dependent on all fan speeds. We assume that each interval's fan mainly affects its own interval cost, and less so intervals before or after it. This means dropping the derivative of the static power term (ε_{static}), since it is the only term that carries the hysteresis in terms of fan-dependent temperature. That is, the dominant dependency of total cost C_N on f_j is the the cost of that interval $\varepsilon_{cost}(w_j, \vec{T}_j, f_j)$. Thus, $\frac{\partial C_N}{\partial f_j}$ simplifies to $\frac{\partial \varepsilon_{cost j}}{\partial f_j}$. Additionally, since temperatures in the past are not dependent on current or future fan settings, the summation will begin at the relevant interval j instead of 1.

As we showed in Section 3.3, the critical element in the chip temperature vector \vec{T}_i is the hottest one. Now recall the analytical temperature model from Equation 3.5:

$$T_{i+1} = T_i e^{\frac{-t}{\tau}} + T_{ss}(1 - e^{\frac{-t}{\tau}}) \quad (3.17)$$

The first term is an exponent of an exponent of fan speeds and converges to 1. For differentiation then, that simplifies the temperature function to $\Delta T = T_{ss}(1 - e^{\frac{-t}{\tau}})$. Substituting this into the constraint comparison of Equation 3.15 for consecutive intervals j and $j + 1$:

$$\frac{\partial}{\partial f_j} \varepsilon_{cost j} + \sum_{i=j}^N \frac{\partial}{\partial f_j} \Delta T_i \lambda_i = 0 \quad (3.18)$$

$$\frac{\partial}{\partial f_{j+1}} \varepsilon_{cost j+1} + \sum_{i=j+1}^N \frac{\partial}{\partial f_{j+1}} \Delta T_i \lambda_i = 0 \quad (3.19)$$

Next, we normalize Equation 3.18 by $\frac{\partial \Delta T_j}{\partial f_j}$ and Equation 3.19 by $\frac{\partial \Delta T_{j+1}}{\partial f_{j+1}}$, then take the difference. Since temperatures must stay strictly within constraints, λ_j must be equal to 0. Most of the terms in the expanded summations simplify, resulting in this equality:

$$\boxed{\frac{\frac{\partial \varepsilon_{cost j}}{\partial f_j}}{\frac{\partial \Delta T_j}{\partial f_j}} - \frac{\frac{\partial \varepsilon_{cost j+1}}{\partial f_{j+1}}}{\frac{\partial \Delta T_{j+1}}{\partial f_{j+1}}} = 0} \quad (3.20)$$

Intuitively, this specifies that the ratio $\partial ratio$ between *execution cost* and the *thermal pressure* should be held constant across intervals. This simplifies the Lagrangian problem into finding a fan setting where this ratio can be kept constant throughout runtime. However, this rule does not specify the actual value of that ratio.

3.4.2 Optimal algorithm design

We use *energy* as an example of an optimization objective in the rest of this paper. The pseudocode for the interior point search for a solution is described in Algorithm 1, solving for a vector of optimal fan speeds for a given workload. In each workload interval, the controller takes the *current workload state* and a *target ratio* as inputs, and solves the combined system model to output the closest permissible *fan speed* that produces a matching ratio, to fulfill Equation 3.20. In lieu of physical sensors, we use Equations 3.2 and 3.5 to represent power and thermal interactions. Equation 3.7 dictates the effect a fan setting has on the execution time of each interval.

To begin, we evaluate the target ratio at the lowest possible fan speed (lines 2-5). The fan speed setting of each query’s first interval drives a target ratio for all following intervals. According to Equation 3.20, this is a potential value of the initial ratio, $ratio_0$, that should be matched for the rest of execution in order to achieve the minimal cost of execution, or *minimal energy* in our case. The rest of the simulation (power dissipation and temperature simulation) follows this decision. For all following intervals, there are fixed costs that are independent of the fan decision, including dynamic power dissipation and static power dissipation (lines 7-8), after which the solver attempts to set a fan speed that matches $ratio_0$ as closely as possible (line 9). After making the interval decision, the solver completes timing and thermal modeling (lines 11-12).

Due to physical limitations of the fan speed and a possibility of overloading the processing workload, there may not be any feasible fan speeds that satisfy $ratio_0$, resulting in temperature violations. Thus, temperature should be checked at the end of the run (line 17). If the constraints are met, the corresponding lowest-

Algorithm 1 Search for energy-optimal fan speeds

```

1: for  $f_0 =$  each increasing fan setting do
2:   for the first interval do
3:      $ratio_0 \Leftarrow$  given  $f_0$ , find  $\partial\varepsilon_{energy}/\partial\Delta T$ 
4:      $t_i \Leftarrow$  given  $\{workload, t_0, f_0\}$ , find temperature
5:   end for
6:   for each following interval  $i$ : do
7:      $dynamicPower \Leftarrow$  fixed for the  $workload$ 
8:      $staticPower \Leftarrow$  fixed for  $t_i$ 
9:      $f_i \Leftarrow$  find fan to match  $ratio_0$ 
10:     $fanPower \Leftarrow$  calculate fan power
11:     $intervalTime \Leftarrow$  find fan-induced delay
12:     $t_{i+1} \Leftarrow$  advance temperature
13:    if  $t_{i+1}$  violates constraints then
14:      continue to next  $f_0$ 
15:    end if
16:  end for
17:  if workload completes within constraints then
18:    minimum cost fan assignment found!
19:    return solution  $\bar{f}$ 
20:  end if
21: end for

```

cost fan assignment is selected as an optimal solution. If one of the constraints are violated, the workload chain is re-evaluated, but starting with the next lowest possible fan speed (loop to line 1). If all fan speed have been exhausted and there is still no solution that meets all constraints, that means the the problem is infeasible, and the only resort is to slow down the CPU workload with DVFS.

Since the total cost is always increasing with fan speeds, the algorithm is described linearly here for clarity, but can be sped up by doing a binary search. Pragmatically, the number of system states and quantized fan speeds are both limited (e.g. only 10 fan settings in our actual server); these values can be pre-

computed and stored in a lookup table for execution at runtime.

3.5 Results

In this section, we demonstrate the effectiveness of our proposed solution by comparing against the state of the art and other proposed solutions in literature. We describe the hardware and software setup of our physical measurements on the real server, as well as the parameters of our modeling and simulation. We describe three state of the art fan control strategies and compare with our results. Finally, we discuss the robustness of our modeling and optimal solver, analyzing how optimization results might change at various levels of model inaccuracies.

3.5.1 Experimental Setup

We model the same server instrumented and measured in Section 3.2, a SPARC T4-1 server with 8 cores running at 2.85GHz, with 8 DIMM modules of 16GB each. We use commodity SATA disks as they are preferred by cost-sensitive datacenters for their low cost per storage density. Buffer caches are enabled to capture the real response of applications along with power, thermal, cooling and disk performance issues.

We evaluate the management policies with a mixed workload of database queries and compute-intensive batch jobs. TPC-H is a decision support benchmark representing databases requests [15]. The queries comprise combinations of operations such as sequential scan, index scan, merge join, and hashing functions. All queries operate near the thermal threshold 85°C. SPEC CPU 2006, on the other hand, is a benchmark suite targeted towards compute-intensive workloads [106]. As per a datacenter environment, the server will be shared with other jobs other than a single database thread. We assume there are four co-located compute tasks on the processor, represented in our power and thermal simulations as single-threaded tasks that consume 8W each (this number was obtained from averaging the power consumption of SPEC CPU 2006 benchmarks). With this mixed workload on our physical system, we encounter both thermal issues due to heavy

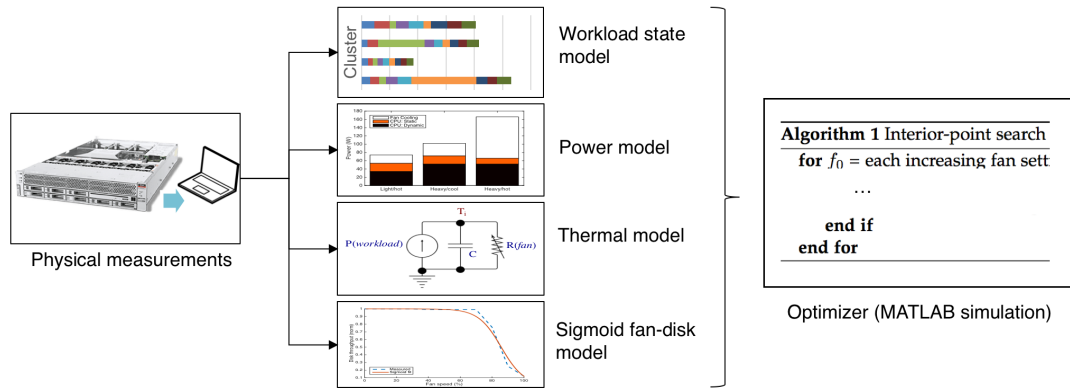


Figure 3.11: Simulation setup with sub-models.

computation, and I/O performance issues due to reliance on the disk access rates.

Figure 3.11 summarizes the organization of data from each sub-model evaluated by each policy. We monitor sensor statistics and event logs on a real server through IPMI [107]. Most enterprise servers already have a side-band controller implementing IPMI to handle server management, reading hardware sensors, and enforcing power modes. Any fan control algorithm should be realistically implemented in this side band controller. The user programmability of this controller is extremely limited, hence we chose to simulate the physically-based models in software. Disk access statistics are collected through *iostat* reports, estimating the number and average service times of queued and active transactions per sampling interval (every second). These event logs are converted into a discrete workload model as described in Section 3.3. The advancement of workload states varies, measured at a granularity of 10ms. Since the packaging thermal time constant is on the order of seconds, the fan control interval is set to 1s. Finally, we use MATLAB to coordinate our physical experimental traces and analytical models, and test multiple algorithms for the effects of different fan assignments.

We compare our proposed controller against three others compared them in terms of delay and energy cost when running mixed SPEC and TPCB workloads. They represent a range of sophistication and complexity in hardware management

Table 3.4: Comparison to related fan control strategies

Algorithm Name	PID, PID-1 [88]	Adaptive PID [89]	JETC [96]	Optimal
Temperature Sensing	Physical sensors	Floorplan sim	Floorplan sim	Numerical model
Fan Control Input	Heat sink temperature	Heat sink temperature, fan speed	Multi-core temperature, power	Core temperature, workload fan speed
Performance considerations	(None)	CPU throttling	Core migration, throttling	Disk Delays
Reaction Horizon	Reactive	Workload prediction	Temperature prediction	Workload dependent
Workload Verification	(Agnostic)	Synthetic	SPEC	TPC-H + SPEC

schemes. For all policies, progressive power gating is applied in emergency cases if the main controller fails to maintain temperatures under the specified threshold (85°C in our system). The first (PID [88]) is time-tested strategy used in many control systems in various engineering fields, representing the state of the art. The next two strategies (Adaptive PID [89] and JETC [96]) were proposed in literature; they make cooling decisions by accounting for temperature conditions as well as CPU performance degradation.

Proportional-Integral-Derivative (PID) [88] is completely agnostic to workload and reacts only to temperature sensor feedback. Being a reactive method, it responds much slower to temperature fluctuations than proactive controllers, and by nature allows both over- and under-corrections before arriving at a steady solution. We show results for “PID-1” which is the same strategy with a setpoint conservatively set below the threshold (by 1°C in our case) to reduce temperature violations. The tuning parameters are determined using the Ziegler-Nichols closed loop tuning method [108]. It operates at a control interval of 10 seconds.

Adaptive PID [89] refines the PID assignment into two zones and scales the tuning parameters dynamically based on the current fan region. It uses the Ziegler-Nichols closed-loop tuning method [108] to obtain PID parameters specific to a high and low fan setting (15% and 65% of the maximum, in our experiments). For all fan speeds between those two settings, the parameters are linearly interpolated, aiming to reach faster convergence. The original proposal for APID stated a control interval of 30s, aiming to converge the fan control within hundreds of seconds. In our experience, a maximum control interval of 10s is required to maintain steady chip temperatures.

Joint Energy, Temperature and Cooling Manager (JETC) [96] uses proactive core migration to control heat generation. In each control interval, this policy predicts the upcoming power dissipation and resulting temperatures. Using the RC thermal model proposed in [94], it then calculates the required cooling capacity to bring temperatures to the system thermal setpoint, and sets the fan speed accordingly. JETC re-evaluates control decisions every second, attempting to stabilize temperatures on the order of 10 ms. While making these decisions, the fan controller aims to minimize fan setting changes during runtime.

Our **Energy-Optimal** controller implements the search described in Algorithm 1, minimizing for total energy consumption. The search is executed offline, then applied to a known query at runtime. For each workload state in a query, it sets the fan to maintain a constant ratio between the change in energy and the change in temperatures. Unlike other policies, control decisions are made when the workload state changes instead of a fixed control interval. In practice, the control interval is on the order of seconds.

3.5.2 Controller policy results comparison

The energy and delay results from a select number of TPC-H queries are shown in Figure 3.12. By design, the energy-optimal solver yields fan settings that yield the lowest possible server energy consumption required to complete a workload. Our energy savings come predominantly from faster completion times and lower overall fan speeds. This controller operates as close to the tempera-

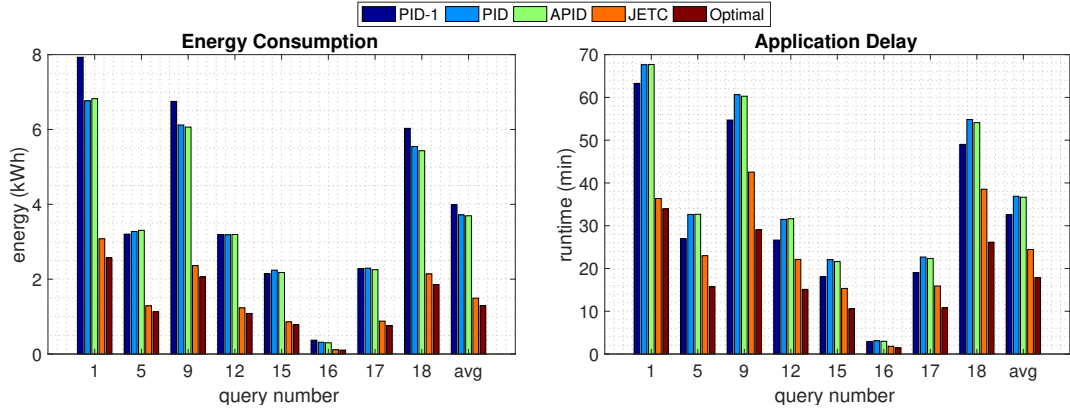


Figure 3.12: Final results of various algorithms on selected TPC-H queries

ture threshold as possible without crossing it, unlike the oscillatory nature of PID solutions. *We guarantee zero temperature violations with lowest energy consumption possible.* Only the PID-1 controller is able to keep temperatures below the threshold, at a much higher cost in terms of delay and energy since it sets an artificially lower thermal setpoint. The slow convergence of APID leads the policy to violate temperature constraints. Compared to these heuristic solutions, our policy performs 2x faster at a 65% lower energy cost. JETC relies on a detailed thermal simulation of the processor package. At each decision interval, it predicts the upcoming temperatures and power dissipation, then calculates the heat sink cooling capacity needed for such a power density to maintain temperatures under specified constraints. For a slowly-varying workload, this can yield very stable temperatures. Because of this model-based calculation, it converges to a solution much faster than the PID-based controls. However, because JETC relies on migration-based management, its artificially actual fan control results in unstable temperatures (this effect was also noted in [89]). To manage these unstable temperatures, this control requires higher fans on average. Compared to JETC, our optimal policy only finishes 1.2x faster on average, but using 19% less energy to finish a workload.

Table 3.5 summarizes some statistics about how each controller behaves in terms of fan speeds and resulting temperatures over all tested queries. By

Table 3.5: Behavior of fan control strategies

Algorithm Name	PID	APID	JETC	Optimal
Average fan (%)	77	58	63	57
Std dev of fans (%)	10.18	12.77	6.59	20.12
Time in emergency (%)	28.6	32.1	12.1	0
Avg. temperature (°C)	84.9	74.1	84.2	84.9
Peak temperature (°C)	91.5	91.2	85.5	85.0

design, the optimal controller never exceeds the temperature threshold, while the PID-based solutions naturally overshoot the temperature setpoint regularly. In these cases, the application will be further delayed and the total server energy consumption will continue to rise. It is clear that JETC achieves its stated goal of “flattening” the fan profile, however in cases where the workload may vary widely, this leads to a very high variance in its temperature. Meanwhile, the optimal solution finds the minimal fan speeds to keep temperatures within limits immediately without need for oscillation like the PID-based solutions, resulting in sharp changes in fan settings immediately after each workload state change.

3.5.3 Sensitivity to model inaccuracy

Our optimal solver relies on several models to represent physical subsystems in the server and search for a solution at design time. In this section, we investigate the effects of error in the power, thermal, and cooling-performance models. These studies how the effectiveness of the convex optimal formulation even when component models are inaccurate.

Power model accuracy

Errors in the power model are described with additive Gaussian white noise in Equation 3.2 used by the solver at each interval. The signal-to-noise ratio (SNR) is shown in decibels (dB), where a higher SNR represents a “clearer” original signal

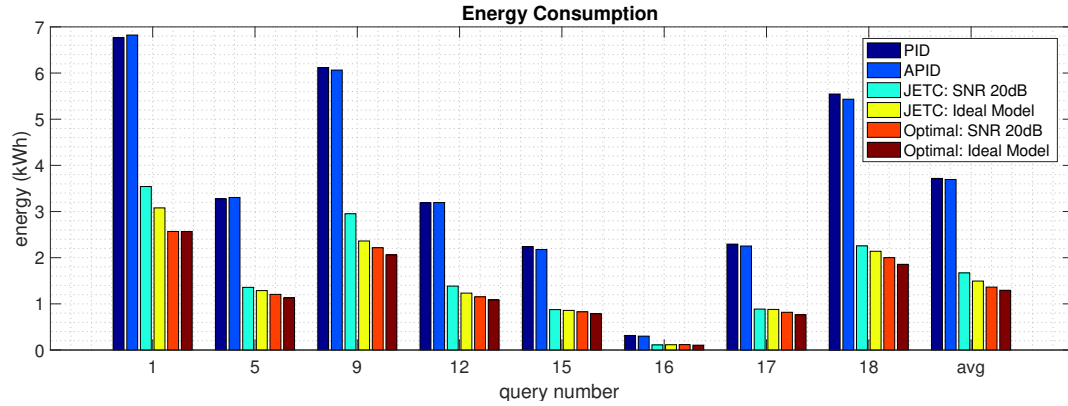


Figure 3.13: Efficacy of the optimal solver with inaccurate power modeling

relative to the noise. Figure 3.13 shows that a noisy power model will result in sub-optimal results and higher energy cost, though the optimal policy still outperforms the next-best JETC policy for most queries. An SNR of 20dB indicates that original signal is 100 times more powerful than the noise. Due to noise added to the power model, the optimal policy consumes on average 6.3% more energy as compared to the optimal policy that uses power model with no noise added. JETC consumes 8.5% more energy when noise is added to the power model as compared to no noise. Heuristic PID and APID policies are not sensitive to the noise in the power model, as they respond only to temperature readings, so their results remain the same regardless of the model. Thus, optimal policies benefit relative to PID and APID policies is reduced on average from 65% to 63% with power model of 20dB SNR as compared to when model with no noise is used.

Thermal model accuracy

We investigate the effects of error in the temperature model (T_{ss} in Equation 3.5). We again use additive Gaussian white noise to estimate errors in the thermal model. Figure 3.14 shows that due to a 20dB SNR in the temperature model (i.e. noise is 1% of the original signal), the optimal policy consumes 6% more energy as compared to when there is no noise in the model. It still outperforms PID by

63%. JETC has a much higher penalty due to noise in the thermal model - its energy cost increases by 95%, consuming 2x as much energy as the optimal policy at the same noise level. We conclude that while both the optimal policy and JETC depend strongly on a temperature model to achieve their objectives, the optimal policy is more robust to reasonable levels of added noise.

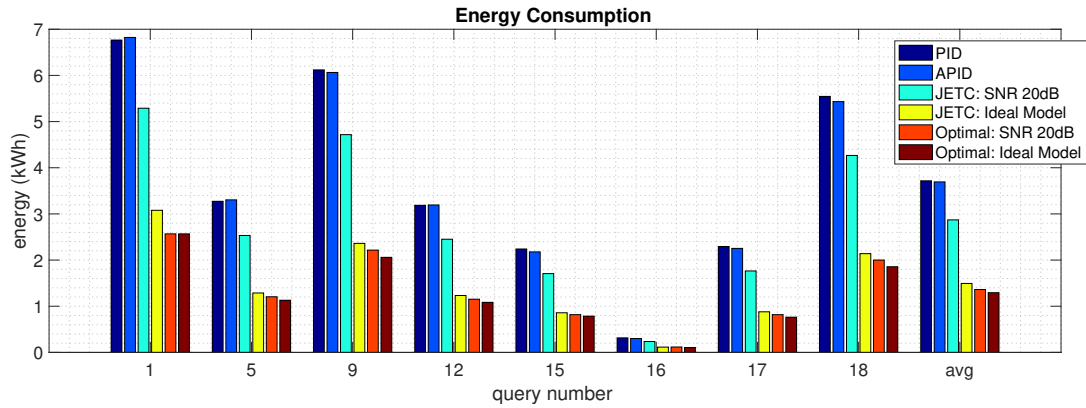


Figure 3.14: Efficacy of the optimal solver with inaccurate temperature prediction

Disk delay model

A major motivation of this work was the realization that fan speeds have a detrimental effect on disk performance [11, 93]. The control algorithm should be aware of the sigmoid shape of disk response to fan settings. The original delay model we proposed in Equation 3.7 is exponential with respect to fan speeds. Figure 3.15, illustrates two alternative models, a piecewise linear function and a constant model (the state of the art assumption), as well as their first derivatives which dominate the convex optimization solution in Equation 3.20. To test these results, we substitute each of these alternative models into the optimal solver calculation, and simulate the final results for a system that still responds with an exponential disk delay relative to fan speeds.

Figure 3.16 shows that if the solver tries to obtain an energy-optimal solution while assuming that disk throughput is independent of fan speeds, it uses

31% more time and 71% more energy to complete the workload on average. A piecewise linear model would have a delay penalty of 4% compared to an accurate exponential model, using 12% more energy in total. This proves the significance of our modeling work - inaccurate knowledge of the physical system leads to sub-optimal results. However, our optimal formulation still provides energy efficiency benefits in the case of a deficient disk model - even the constant model saves 37% system energy compared to PID.

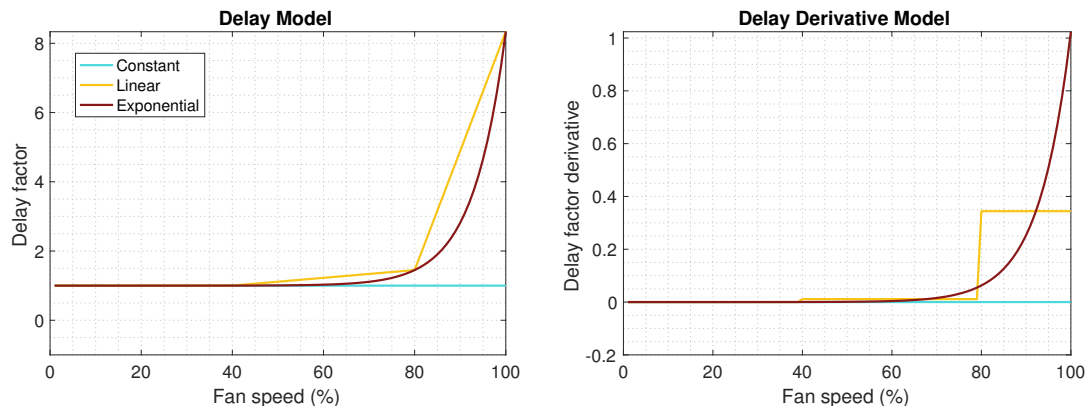


Figure 3.15: Alternative disk delay models used by the optimal solver

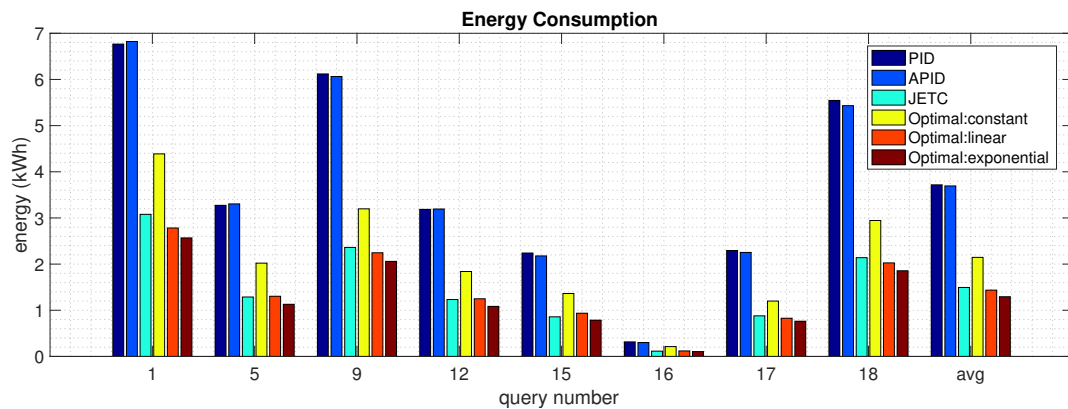


Figure 3.16: Efficacy of the optimal solver with inaccurate delay models, compared against default “exponential” function

Table 3.6: Overhead of fan control strategies

Algorithm	PID [88]	APID [89]	JETC [96]	Optimal
Offline Design Effort	Medium (tuning)	Medium (tuning)	High (modeling)	High (modeling, solution search)
Online Decision Delay	7.81 μ s	7.89 μ s	4.14 μ s	1.83 μ s
Simulation time for query 1	5.8 min	7.3 min	6.7 min	3.7 min

3.5.4 Overhead

Table 3.6 summarizes a comparison of the design and runtime sources of overhead for each policy. It also shows the example solution time for the longest query in the suite, *q1*. Our simulations throughout this section were performed on an Intel Core i5 unning at 2.6 GHz.

PID [88] requires manual tuning before deployment; the Adaptive PID controller [89] has to be tuned twice. Iteratively tuning the controller terms takes many multiples of the machine’s thermal time constant - minutes per iteration. In contrast, both JETC [96] and our strategy require up front modeling effort. Both strategies model the system’s physical characteristics including thermal time constants, heat capacity, cooling capacity and workload power consumption. Our solver also models the resource utilization and power consumption of the workload before it runs. The workload trace clustering implemented in Python took less than 10 seconds over thousands of samples collected over 4.5 hours of actual database execution. Since this is a learning algorithm, it can be re-executed periodically to fine tune the workload model, in case application characteristics change. In addition, our optimal solver performs a search at design time for fan settings given starting temperatures and a known workload. This search can be completed in MATLAB in approximately 3.7 minutes for the longest query.

Runtime delays for each policy may include the time to compute a decision. For PID and APID, the controller has to compute the derivative and integral of

historical errors, which contributes to the delay of over $7\mu s$ for each decision. They also have delayed reactions to real system stimuli at runtime as opposed to the proactive decisions made by JETC and our optimal policy. Because JETC relies on second-to-second granular temperature prediction, it simulates the processor floorplan in detail before calculating a change in fan speed. This calculation takes on average $4.14\ \mu s$ in our experiments. By spending all the design effort in accurately modeling the system, our runtime overhead costs ($1.83\ \mu s$ per decision) come only from accessing the power and temperature sensors, fan tachometer, and looking up the precomputed solution.

3.6 Conclusion

In this chapter, we measure many sources of hardware context within a single node, and optimized fan control with this contextual data to lower energy costs by up to 65%. For each subsystem in the server, we discuss the physical sensing basis and how we define the context data extracted. We observe and quantify hard disk sensitivity to environmental disturbances such as datacenter HVAC cooling vibrations, and server fan vibrations, their relationship to data-intensive workloads, and consequent energy wastage. We identify a previously-ignored link between the cooling system and application performance, where commodity drive sensitivity to fans can cause up to 88% lower performance in realistic cooling situations. We also develop and integrate interdependent analytical models for performance, power, thermal, cooling. Finally, we define a multi-model objective function that can be solved to find optimally low-cost fan speeds, saving 19-65% of CPU and fan energy while guaranteeing that critical thermal constraints are still met 100% of the time.

This context extraction methodology may be generalized to many different well-engineered and well-known systems. For platforms that operate in more unpredictable scenarios, such a closed-form solution focused on each device may be infeasible. In the next chapter, we discuss situations and applications where users and ambient environments are highly variable. While analytical optimizations may

be inappropriate for those scenarios, machine learning techniques provide a scalable way to exploit both human and environmental context data across a hierarchy of diverse devices.

This chapter contains material from “Optimal Performance-Aware Cooling on Enterprise Servers” by Christine Chan, Alper Sinan Akyürek, Baris Aksanli, and Tajana Šimunić Rosing, which was submitted for consideration in IEEE Transactions on Computer-Aided Design. IEEE 2017. The dissertation author was the primary investigator and the author of this paper.

This chapter contains material from “Correcting vibration-induced performance degradation in enterprise servers” by Christine Chan, Boxiang Pan, Kenny Gross, Kalyan Vaidyanathan, and Tajana Šimunić Rosing, which appears in SIGMETRICS Performance Evaluation Review. ACM 2013. The dissertation author was the primary investigator and the author of this paper.

Chapter 4

Environmental context: Air quality and smart health

Beyond datacenter workloads, user-centric Internet of Things (IoT) applications function in many environments that may be varied and unpredictable. In this chapter, we return to a wider view of the IoT, where context extraction is performed not only internally in each device, but also communicated across a set of devices. The scope of context for an application then consists of human and environmental data in addition to the machine performance metrics from previous chapters. Because of the variable deployment environment, devices should leverage environmental context to drive operation, including actuating their sleep modes, sampling intervals, or communication intervals.

We use a sample healthcare system to illustrate the different scopes and scales of data collection in a single application, which may be processed by machine learning tasks distributed over a hierarchy of devices. The application combines large scale datasets, collected at a population level, with local sensing, and applies context-aware functionality to improve the battery efficiency of resource-constrained nodes at the edge. Realistically, the same learning tasks may be executed on multiple devices, albeit on different data sizes. This task allocation is part of the different cost tradeoffs like energy efficiency, processing speed, or battery life of sensor nodes. We will discuss how to leverage the heterogeneity of devices in this hierarchy to meet different system design goals. Though still under development,

the ultimate goal is to display actionable information such as personalized asthma risk to a user. Though we use a smart health as a case study, these observations can be generalized to other IoT applications that span a range of devices like small sensor nodes, gateways and cloud aggregators.

4.1 Background and Related Work

Many researchers have utilized machine learning and data mining techniques to draw new insight from large studies of geographical models. Some have proven hypotheses about the effects of urbanization on asthma hospitalization rates [109, 110] or the link between air pollutants vs. asthma incidence [111]. The DELPHI project is a large-scale integration of electronic health records (EHR), environmental data, medication usage and physical activity monitors. They present a specific personalized dashboard that summarizes this data and displays relevant data any specific patient [12]. This is a quickly developing area. As a field, we have some understanding of correlations between environmental/geographical factors for widespread conditions like asthma, but are notably missing automated system for individual patients to incorporate this data in daily life.

As sensor networks and context-aware computing become increasingly flexible and personalized, much effort has been made towards empowering citizens to monitor and analyze their own environment. Healthcare or fitness applications in the IoT require hardware and software systems for crowd sourced air quality monitoring [112, 113] leading to more comprehensive studies of populations and geographic factors in public health [114]. Wearable pollutant monitors have seen success in the market, whether for personal or industrial use - the degree of connectivity varies [115, 116]. And to effectively integrate sensed data from various sources, many communication protocols have been researched for maximum usability and efficiency, such as the pub/sub model for crowd sourcing [117], and aggregation inside the network [118, 119].

There are many ideas and prototypes for crowd sensing, but precious little validation, so we are motivated to find the connection to higher-level actionable

health information. Electrochemical sensors are a cost efficient solution for gas measurement, with price tags on the order of hundreds of US dollars as compared to those used at regulatory stations, which cost tens of thousands of dollars. The EPA includes measurements of six critical air pollutants in the definition of the air quality index (AQI) [120], three of which our platform measures by default. The previous stage of this work produced a mobile modular sensor platform that detects air pollution indicators including nitrogen dioxide, ozone and carbon monoxide at a parts-per-million (ppm) granularity, reporting either batched or real-time results directly to users via a mobile application and aggregated all data to the Internet [121, 122]. Our proposed platform has increased processing power such that it can execute some machine learning algorithms, and can be paired with an IoT gateway or directly upload data via Wi-Fi to the cloud. In the interest of maintaining scalable data volumes and communication bandwidth, we will implement and study the opportunities for intelligent processing natively on the sensor node, instead of naively aggregating all data in the cloud.

4.2 A modular context-sensing platform

As a test platform, we developed an air quality sensing board, which served as a context-generating sensor where context is derived from local environmental conditions (e.g. pollutants used as air quality indicators, temperature, humidity). The platform was designed with the primary goal of accurately sensing pollutants while maintaining modularity and extensibility, allowing end users and researchers to reconfigure the board and replace modules as required to tailor the functionality to their specific end needs. The result is a platform that can interface with various sensing modalities using standard communication protocols (I2C, SPI, analog, UART, USB, and BLE) and process the data with an ARM Cortex M3 microprocessor.

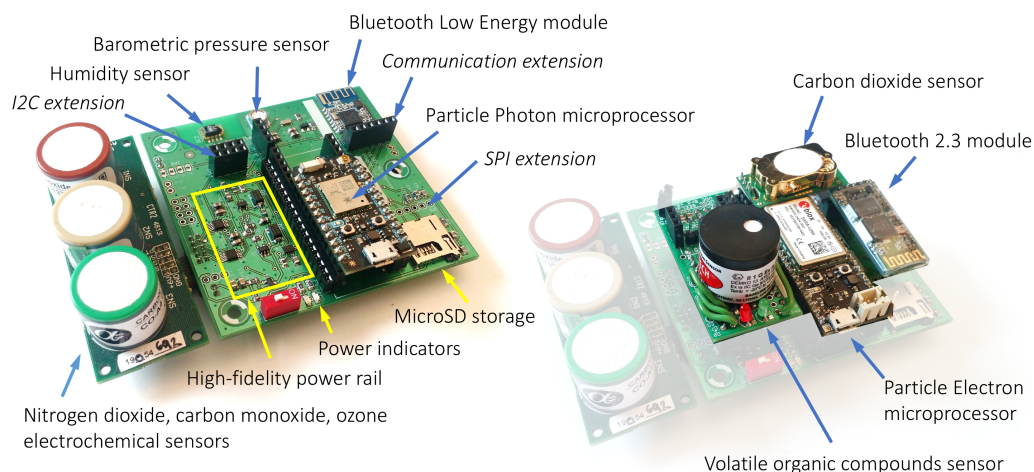


Figure 4.1: The AQ board with electrochemical sensors, supporting humidity and pressure sensors (left), and with VOC sensors, Bluetooth module and alternative processor (right).

4.2.1 Sensors

The air quality sensing platform can interface with any 3.3 or 5.0 V sensor that communicates using I2C, SPI, analog, or UART. The interface options permit a variety of sensing modalities to be used. Our configuration utilized electrochemical sensors for traditional air quality indicators (NO_2 , CO , O_x), nondispersive infrared sensors for CO_2 , photoionization detectors for volatile organic compounds (VOCs), and a variety of environmental sensors (temperature, humidity, barometric pressure). A list of tested sensors is shown in Table 4.1.

The base configuration of the platform uses three electrochemical sensors for monitoring the traditional air quality indicators, which include NO_2 (Alphasense $\text{NO}_2\text{-A43F}$), O_x (Alphasense $\text{O}_3\text{-A431}$), and CO (Alphasense CO-A4). These chemicals are defined in the US National Ambient Air Quality Standards as part of the Clean Air Act. While these three sensors were selected for our application, Alphasense makes a variety of other pin compatible sensing elements for hydrogen sulfide (H_2S) from sewage treatment plants, nitric oxide (NO) from automobile emissions, and sulfur dioxide (SO_2) from coal power plants. The sensors are

Table 4.1: Sensing elements of the air quality sensing platform, including built-in and optional extensions.

Sensor Name	Measured element	Communication
Sensiron SHT11	Humidity	Proprietary 2-wire
Sensiron SHT11	Temperature	Proprietary 2-wire
TE Connectivity MS5540C	Barometric pressure	SPI
TE Connectivity MS5540C	Temperature	SPI
Alphasense NO2-A43F	Nitrogen dioxide (NO ₂)	Analog
Alphasense O3-A431	Ozone (O _x)	Analog
Alphasense CO-A4	Carbon Monoxide (CO)	Analog
Alphasense PT1000	Temperature	Analog
Telaire T6713	Carbon Dioxide (CO ₂)	I2C
Mocon pID-TECH eVx	Volatile Organic Compounds	Analog
ams iAQ-Core	Volatile Organic Compounds	I2C
STMicroelectronics STC3105	Power Consumption	I2C

mounted to a companion analog front end (AFE) from Alphasense, which assists with voltage regulation and signal amplification. Electrochemical sensors offer a high level of accuracy at a low current consumption. Each sensing element has two electrodes which give analog outputs for the working and auxiliary electrodes. The difference in signals is approximately linear with respect to the gas concentration detected but have dependencies with temperature, humidity, barometric pressure, and cross-sensitivities with other gases. By accurately measuring the environmental conditions, the effect of these dependencies on the measured signal can be reduced. Due to the kinetics of the sensing mechanism, the gas sensors have a long required warm up period that prevents systems from power cycling for energy savings. In our experience, a warmup period of 30 minutes is sufficient to reach valid values. After which, the sensing elements will respond to a change in environmental conditions with a 60, 45, and 20 s response time for NO₂, O_x, and CO, respectively.

The environmental sensors (MS5540C and SHT11) are important for correcting the environmentally related offset in electrochemical sensor readings and for accurately measuring temperature, humidity, and pressure, which can affect medical conditions. The sensors self-calibrate and are able to deliver accurate readings after a short delay at startup. The TE Connectivity MS5540C is a barometric pressure sensor capable of measuring across a 10 to 1100 mbar range with 0.1 mbar resolution. Across 0°C to 50°C , the sensor is accurate to within ± 1 mbar and has a typical drift of -1 mbar per year. The Sensiron SHT11 is a relative humidity sensor capable of measuring across the full range of relative humidity (0 to 100% RH) with a 0.05% RH resolution. Both sensors come equipped with temperature sensors with $\pm 0.8^{\circ}\text{C}$ and $\pm 0.4^{\circ}\text{C}$ accuracy, respectively. The sensors stabilize to environmental changes in under 30 seconds, which is sufficiently fast to capture changes in the local environment.

All sensing elements rely on a passive diffusion mechanism for pollutants to reach the sensing elements. While an active air flow rate can result in faster response times and potentially more accurate readings, an exhaust fan has prohibitively high power consumption for a mobile, embedded platform. To study the effect of air flow and enclosure design, we are in the process of producing multiple 3D printed enclosures that promote different levels and directions of air flow for comparative testing.

The electrochemical sensors generate an analog output, which is connected through a header to a pair of ADCs (TI ADS6115). The 16-bit ADC has a programmable gain amplifier before the conversion circuitry, allowing the effective full-scale range between differential inputs to vary between $\pm 0.256\text{ V}$ and $\pm 4.096\text{ V}$. The ADC has a maximum full-scale range of $\pm 6.144\text{ V}$, but it cannot be fully utilized with the 5.0 V supply rail of the board. The adjustable gain is a useful feature that permits dynamic sampling resolution for capturing small variations of typically low signals and the high signals near pollution sources. The ADC is operated in a single-shot conversion mode where all channels are sampled sequentially in accordance with our desired sampling rate. The ADC has the ability to perform continuous conversions, but it is not practical to sample at 8 Hz when

the required sampling rate is 1/5 to 1/30 Hz. By not continuously sampling, the average ADC power consumption drops from 150 μA to $< 2\mu\text{A}$.

The platform has headers for connecting additional sensors. In our configuration, total VOCs (TVOCs) and carbon dioxide were important aspects for quantifying indoor air pollution. VOCs are a major indicator of air quality in indoor environment and are outgassed from a variety of building materials, cleaning agents, paints, and microbial agents [123]. Carbon dioxide concentrations have been linked to decreases in concentration and productivity [124] and can be used to estimate occupancy [125]. Both of these sensors (shown in Figure 4.1 connect to headers on the topside of the platform. These sensors were added to the air quality sensing platform with no design changes, showcasing the modularity of the design. Different sensors can be connected as required for end user applications.

Initial data was collected by co-locating a single sensor platform with a reference station at the Colorado Department of Health and Environment monitoring site. Using 2 weeks of minute-by-minute data, data analysis and verification of the gas chemistry model was performed [126]. This calibration model was used in the conversions for all remaining boards. Depending on the mode of deployment and availability of other nearby sensors, we expect to adjust the model computation for each individual board based on reference data from either a regulatory air quality monitoring station, or a calibrated mobile sensor node. The mean standard deviation with direct emissions across 6 sensors over 2.75 hours: 27.8 ppb. Mean standard deviation in ambient environment without direct emissions in a 4.5 hr period: 8.3 ppb. We can adjust the ADC gains dynamically online, and update the model parameters through the processor.

4.2.2 Processing and communication

The air quality sensing platform is compatible with the Particle Photon [127] and the Particle Electron [128]. The Particle Photon runs on an ARM Cortex-M3 32-bit core at 24 MHz with a real-time clock (RTC), 1 MB flash, and 128 KB SRAM (ST Micro STM32F205). Application code is written in C++. It has a Wi-Fi module (Broadcom BCM43362) through which it can upload data to the internet

and have its firmware updated wirelessly. The Broadcom Wi-Fi module has a single antenna connection and can support IEEE 802.11 b/g/n. The Particle Electron has the same ARM processor but is installed with a 3G cellular module instead of the Wi-Fi module. The U-bloc SARA-U260 and G350 chipsets provide 3G and 2G cellular connectivity, respectively. Although developed for direct interface with the Particle devices, Arduino boards could be substituted with minor additional wiring. Each of the Particle modules offer a secondary power connection to the RTC and SRAM, allowing the board to retain a correct timestamp and critical configuration values in low-power situations that may cause the microprocessor to reset.

The microprocessor collects readings directly from digital sensors and digitized values from the ADCs. It arbitrates the various buses and slaves and then compiles the data into a JSON-formatted message. Each sample record currently includes a full real-time timestamp, uptime of the board, and values from all sensors on board, but it can be updated to include power management statuses, calibrations, or contextual output from the on-board Context Engine. Messages are sent in ASCII over UART to the Bluetooth Low Energy (BLE) 4.0 module (HM-11), which wirelessly relays the message to a paired personal device. BLE is well suited to the air quality sensing application because it offers a secure, low-power communication channel. The low bandwidth of BLE is not a problem with the low sampling rate of the system. For stationary deployments, Wi-Fi and 3G protocols can be used to communicate to a backend server. The networks are prevalent within densely and moderately populated areas. While BLE is provided by default for communication, any communication module that communicates serially can be added to the communication extension header, such as the HC-06 Bluetooth 2.3 module for backwards compatibility.

In addition to sending data, the platform has a two-way interface with a mobile application on Android phones (currently available for Android 4.4+). JSON-formatted commands allow the user to remotely toggle whether data is stored locally on the SD card, change the sampling interval, adjust configuration parameters for the analog sensors, and update the conversion model on the fly.

4.2.3 Power

The air quality sensing platform can be operated in a stationary or mobile mode, drawing power from a rechargeable battery or wired USB connection, respectively. We equipped a 1800 mAh lithium-ion battery, but for a higher energy density, a primary cell option could be used if it connects to a standard 2-pin (2 mm spacing), keyed power connector. When connected to a powered USB line, the battery is recharged through an on-board battery charging IC. Topside LEDs serve as power and charging indicators.

In addition to a time-based sleep mode, the Photon can enter deep sleep mode and be woken up by hardware interrupts. On our platform, interrupts can be generated by the BLE module to signal when a nearby device has been connected or by programmable alert pins on the ADCs, so that the board may conserve energy by going into a low power mode until pollutant levels reach a desired threshold without manual polling.

Air quality measurements are low-frequency signals, varying on the time-frame of minutes and hours. Combining a low sampling rate with deep sleep mode on the microprocessor enables significant power savings. For reference across the operating temperature range of -20°C to $+60^{\circ}\text{C}$, the Particle Photon consumes 30 mA in normal operational mode with the Wi-Fi off, 1 mA in sleep mode, and $99\ \mu\text{A}$ in deep sleep mode.

The sampling rate can be adjusted for the required application (e.g. mobile sensing may require a longer battery life than a stationary deployment). On the scale of environmental sampling rates ($< 1\ \text{Hz}$), the $17\ \mu\text{s}$ wake up time does not affect performance. The smallest sampling period is defined by component hold times and bus delays, taking 290 ms to update readings from all sensors on board. The sampling rate is appropriate for monitoring ambient conditions, but further shortening the awake time can improve the operational lifetime of the device.

Our base configuration equipped with the Particle Photon has two primary methods of communication: BLE and Wi-Fi. The BLE module is based on the TI CC2541 chipset, which has current consumption of 15 mA and 8.5 mA when transmitting and receiving, respectively, but only $600\ \mu\text{A}$ in sleep mode. The

Wi-Fi module has even higher power consumption at 80 mA average. Due to the low average power that is inherent to the BLE protocol, BLE can be in regular communication with nearby devices, but the high power cost of Wi-Fi prevents it from being regularly deployed in mobile settings. If it is needed, the Wi-Fi can be cycled on for short periods for burst communication to a backend database. If real-time measurements are not required in an application, data can be stored to a local SD card for later analysis, removing the need for higher power communication.

If VOC or CO₂ measurements are required for an application, the board should be configured into a stationary, plugged position due to the high current draws of the sensors. The VOC sensors pull 39.7 mA and the CO₂ sensor draws an additional 30.2 mA baseline with large spikes in current (500-600 mA for 560 ms) when the infrared lamp turns on. While the system can provide these current draws, the battery would not last long enough for sustained mobile deployment. A summary of the current consumption in different operational modes can be seen in Table 4.2.

Table 4.2: Power consumption in various configurations. “Env” refers to the set of basic environmental sensors (humidity, temperature, and pressure) and “AQI” refers to the pollutant sensors.

Microcontroller	Bluetooth	WiFi	Sensors	Power (mW)
	Off	Off	None	0.42
Deep Sleep	Deep Sleep	Off	Env	10.15
	Deep Sleep	Off	AQI+Env	52.19
Idling	Off	Off	None	144.20
	Deep Sleep	Off	Env	153.26
	Deep Sleep	Off	AQI+Env	192.40
Active (24 MHz)	Connected	Off	Env	231.62
	Connected	Off	AQI	249.92
	Deep Sleep	Connected	AQI+Env	519.94

Our platform currently consumes 192.4 mW during active sampling and

processing, and 52.2 mW in deep sleep mode. With a 1.8 Ahr battery, a sensor node lasts 5.0 days at a 5 s sampling interval and 5.6 days at a 30 s sampling interval. This is a conservative characterization. Power efficiency improves by optimizing code use and aggressively power gating various modules on the board whenever possible. For reference, other commercially available air quality sensors, such as the AirBeam [115] and CairClip [116], can only measure up to 10 and 36 hrs on a single charge, respectively.

4.3 Context-aware Smart Health application

In healthcare settings, the data collection and aggregation hierarchy includes components such as wearable monitors, environmental sensors, in-home care equipment, and larger electronic health records (EHR). Heightened privacy concerns in healthcare settings further motivate the need for allocating tasks to nodes close to the edge of the cloud, limiting the communication and exposure of sensitive medical data. The main idea of our application is to integrate contextual information at two very different scales. First, we build an initial model for regional risks towards asthma. To bring this large-scale data down to a relatable and useful data point for an individual, we combine it with personal environmental data collected at a local level. In fact, the context engine framework is general enough that the same coding infrastructure used for our previous fitness work in Chapter 2.4 are easily instantiated for a healthcare application. For example, a context engine running on an air quality sensor nodes works to detect anomalies in continuous air quality data. The anomalies can then be collected by the patients cell phone, running an intermediate content engine, that will correlate air quality anomalies with physical activity and respiratory metrics to learn when asthma related problems occur. At a larger scale, a context engine running in the cloud may use data from many individuals context engine outputs to gather correlations between asthma, geographic location, air quality and physical activity. Meanwhile, the intermediate data generated can be shared for other uses, reducing the computational load for those applications.

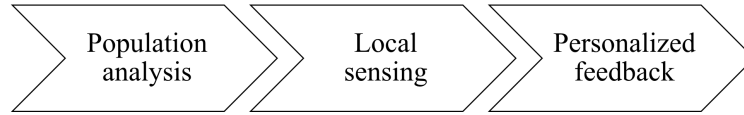


Figure 4.2: A workflow for building our personalized health application based on population health data

In this section, we present our experimental setup for deploying a distributed sensing and health application. Our set up includes an original modular sensing platform that fits into the IoT as a sensor node, capable of both collecting raw data and providing some limited amount of energy-efficient computation. In Section 4.4, we show measurement results and discussions for the several possible configurations of running this application on an ecosystem of heterogeneous nodes.

4.3.1 Population health

At a population level, hospitalization rates serve as an indicator for health, and the primary diagnosis for any particular emergency room admission or hospitalization represents the overall risk that a population has towards that condition. We have preliminary results based on annual county-level data from the EPA [129] and California Department of Public Health [130]. The EPA data includes daily pollutant averages and maximum hourly counts, as well as temperature and humidity data, for each county in California. The demographic data includes hospital and emergency room asthma admission rates per 10,000 persons, for each county annually, as well as the percentage of residents who live within 0.5 miles of a public park, and their commute habits. The pollutant count and demographic data show linear correlations for the concentration of pollutants such as CO and Ozone and the likelihood that more citizens will be admitted to the ER for asthma complications. On the other hand, the percentage of citizens who live within 0.5 mile of a park is negatively correlated with the county’s rate of hospital admissions for asthma. The model goes beyond just linear relationships - it includes up to 2nd or 3rd order cross-dependencies between different input variables (depending on the

amount of public data available).

Based on this model, for any given environmental sample of gas pollutants and related cross-dependencies (temperature, humidity), we predict the related asthma risk. We divide the dataset into 10 subsets for cross-validation of the regression model. Currently, using various subsets of the environmental data, we can predict a county's hospital and emergency room admission rate for asthma with 25-35% error. This is useful to test hypotheses on the general population by leveraging existing historical data, without instrumentation. Once correlations are identified between variables of interest, then we can refine the top-down model with personal details as they are made incrementally available. Respiratory sensitivity varies widely among citizens even asthma patients can have different triggers depending on their vulnerabilities. There is value in helping each individual track and learn how their condition responds to acute triggers (e.g. known allergens) as well as their vulnerability according to daily air exposure. Going forward, we will also move into monthly and seasonal data. We also plan to include data from more census and city planning sources, such as what fraction of the population lives or works in proximity to highways, industrial areas, bus depots, or trains.

4.3.2 Personal health

With the population health model, citizens can now have a idea of what environmental factors generally have effects on respiratory health. However, they cannot apply this knowledge to their daily lives until they have 1) up-to-date data about their immediate environment (e.g. carbon monoxide concentration in the surrounding city block is 2 parts per million), 2) a translation from that current, local data into a quantifiable health-related terms (e.g. CO 2 ppm is a safe level of air quality, no action needed), and possibly 3) personalized factors such as allergies, medical history, or level of aerobic activity. Such a personal application should give insights on whether the user is going through typical exposure, or may need to be alerted to sudden and unexpected changes in their environment.

Since each person's health and activities vary, it is important for us to automatically sample and extract their behavioral context. For such an application,

we can split up the tasks in order of:

1. **Environmental raw sampling:** Although the Alphasense electrochemical gas sensors we use have some temperature adjustment, calculating the cross dependencies with other changing environmental factors, including temperature, humidity, barometric pressure, and other gases) is non-trivial, requiring at least 2nd order level polynomials with high-precision coefficients.
2. **Sensor data summarization and anomaly detection:** To reduce the amount of transmissions from sensor nodes to collector nodes in the network, we can use anomaly detection techniques to filter out “typical” samples from “anomalous” ones. We use a SMCTC-based [131] particle filter to predict the expected value according to an irregular time series stream of air quality values. If an incoming sensing sample deviates from the expectation, it is labeled as an anomaly. The results can be used to reduce sensor power consumption and communication congestion by down sampling whenever samples are “typical” instead of “anomalous”.
3. **Nonlinear modeling of health risk for patient feedback:**

Finally, we build and update Taylor-expansion based nonlinear regression models (TESLA) [51] to learn which environmental factors have the highest correlation with asthma incidents at a population level, as described in the previous section.

In the remaining sections, we will describe the experimental setup for evaluating several variations of this application implementation on a hierarchy of devices. we will describe a specific sensor node design that is capable of collecting multiple environmental samples and processing some data.

4.4 System evaluation and results




4.4.1 Smart Health Experimental Setup

We roughly categorize the available system into three different types of compute nodes, in order of increasing resources: sensor nodes, gateways and edge servers. In our testbed, they are represented by the air quality sensor platform (Particle Photon system-on-chip with ARM Cortex M3), Raspberry Pi 3 Model 3 (Broadcom system-on-chip with ARM Cortex A53) [132] and an edge server (similar to Intel Xeon D Processor [133] or Dell Edge Gateway 5000 [134]). The three application subtasks will be delegated to different devices. The sampling task is uniquely executed on the air quality sensor. The modeling task is an aggregation function and must run on an IoT gateway or server.

The Raspberry Pi 3 (RPi) has a Broadcom BCM2837 SoC, where the processor runs at 1.2 GHz by default, and the typical power consumption we have observed is between 1.2W - 2.5W depending on utilization, and up to 3.5W when communicating with Wi-Fi. The edge server we used runs at 2.6 GHz and consumes between 12W - 45W depending on the utilization, and is constantly connected to the Internet via Wi-Fi. The air quality board communicates with gateways via Bluetooth, or directly with an edge server using Wi-Fi. The RPi has built in Bluetooth and Wi-Fi capabilities, we assign it to communicate with the edge server via Wi-Fi. The RPi consumes 2.9-3.5W when using Wi-Fi[46]), while the AQ board consumes 100mW when transmitting on BLE, and 150mW when transmitting on Wi-Fi.

In this chapter, we use end-to-end metrics such as total execution time and total device power consumption to quantify the execution of different devices. We can further inspect details of subsystems if needed, to identify bottlenecks and optimize performance (e.g. using performance counters in the operating system, easily accessible in Linux), but considering the scale of our system, we choose to take the nominal performance for granted and focus on the ecosystem scale instead.

Table 4.3: Three devices representing a range of capabilities in the cloud

Node:	Sensor board	Gateway	Edge server
Product name	Particle Photon	Raspberry Pi 3	Macbook Pro (Mid 2014)
			
Chipset	ARM Cortex-M3	Broadcom BCM2837 with ARM Cortex A53	Intel Core i5
Clock speed	24 MHz	1.2 GHz	2.6 GHz
Power draw	192-520mW	1.2-2.5W	20-35W

4.4.2 Individual devices

We first evaluate the capabilities of each node for running the three subtasks in our health system (listed in Section 4.3.2). For the machine learning tasks, there are various configurations to choose from for accuracy and speed of execution. Figures 4.3 and 4.4 show the costs of executing tasks on various devices, in terms of delay and energy respectively. In anomaly detection, we use a particle filter with l represents the number of iterations and p represents the number of particles. We observe that execution is always slowest on the AQ board and fastest on a server, but energy efficiency varies more.

In a real hierarchical system where context engines are delegated to various connected nodes, each sensor nodes, gateways and cloud servers would be running different workloads with different data sizes, requiring different communications.

4.4.3 Ecosystem utilization

Next, we'll consider this heterogeneous system, where sensor nodes are running smaller workloads than servers, but we also include data transmission costs.

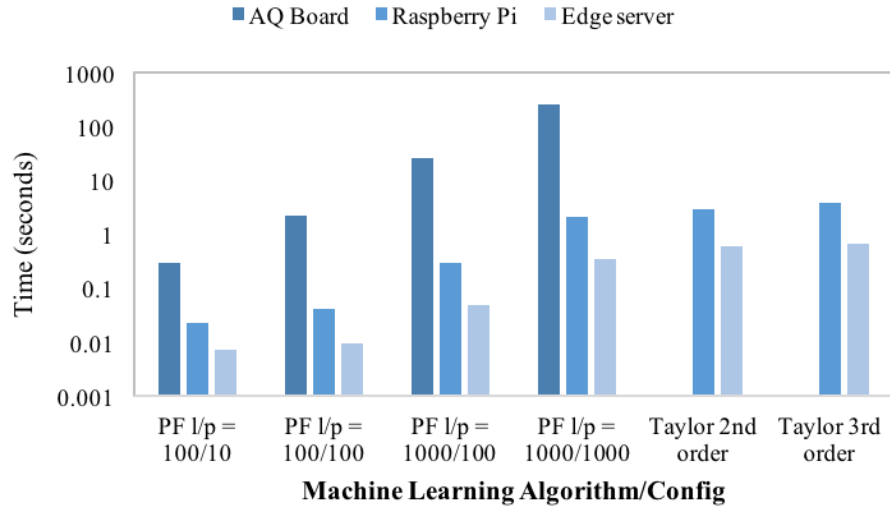


Figure 4.3: For different configurations: Total execution time for anomaly detection and health risk model on different nodes. Note that y-axis is in log scale.

For anomaly detection on the limited AQ board, we choose the 100/100 configuration (100 particles and running 100 iterations for time series prediction) - that is a reasonable processor and memory usage to still allow for timely sampling and communication activities. For the particle filtering model running on the larger Raspberry Pi and server, we choose the 1000/100 configuration, accounting for the fact that they will be processing more diverse, larger sets of data. A third-order Taylor series is used in the non-linear model. Table 4.4 shows some examples from the spectrum of distributed computation vs. cloud aggregation. System 1 and 2 spread tasks between the AQ board and the Raspberry Pi only. System 3 and 5 use only the AQ board and the edge server. System 4 utilizes all three nodes.

In a simple two-node system, the AQ board collects all environmental samples and transmits them to the cloud, one of which will calculate the risk model. The Taylor-expansion based model requires second by second data, and is updated each time a new sample arrives. The sensor communicates with the gateway RPi over Bluetooth or directly to an edge server via WiFi. The radios are turned off when not in use. We round up the transmission time for each data sample to 1 second. If the AQ Board does not perform anomaly detection, it must send out

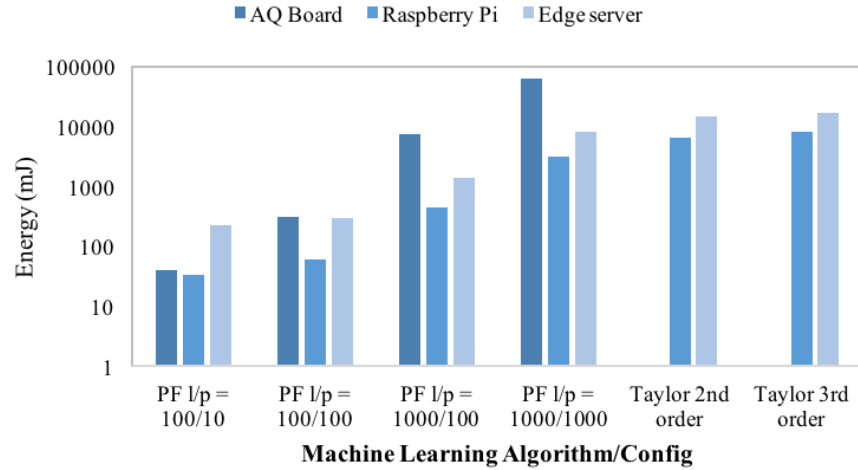


Figure 4.4: For different configurations/data sizes: Total energy consumption for anomaly detection and health risk model on different nodes. Note that y-axis is in log scale.

a new data sample every second (86400 samples per day) to support timely risk model updates. But if instead of transmitting all its data immediately, it is able to run anomaly detection locally, it may be able to batch its mundane data until something “interesting” shows up on the sensors. Each invocation of anomaly detection at the edge of the cloud may perform slower in an isolated speed test, but it is very useful for cutting down the amount of meaningless/redundant information that gets transmitted. In the case with anomaly detection available, we begin by setting a default sampling interval of 30 seconds (2880 samples per day). If an air quality sample deviates from its expected value, we set a higher sampling rate of 1 Hz. Thus, the risk model gets occasional “confirmation” sample readings at 30s intervals when data is not changing, and gets updated, detailed samples every second if new data arises that might change the model computation.

We estimate the total system energy across nodes, for different system configurations and different real-life sensing situations. Figure 4.5 shows the difference in energy consumption for these various task allocations, given different amounts of anomalies detected in a day. For example, “1 min daily” indicates that roughly a total 1 minute of air samples per day are anomalous and require higher sampling

Table 4.4: Task distributions to context engines residing on each node

Node:	Sensor board	Gateway	Edge server
System 1	Sampling Anomaly detection	Risk model	
System 2	Sampling	Anomaly Detection Risk model	
System 3	Sampling Anomaly detection		Risk model
System 4	Sampling	Anomaly detection	Risk model
System 5	Sampling		Anomaly Detection Risk model

and transmission rates to the risk model. The more anomalies there are in a day, the more energy all systems must expend to process them - but it also widens the gap between systems that distributed workloads to smaller sensor nodes compared to those that aggregate in the cloud. To compare fairly among systems that still include the most powerful edge server (Systems 3-5), we see an energy difference of 21.5-44.35%.

While the total energy consumption of an ecosystem certainly affects the bottom line for the application vendor, there are additional critical constraints to be met for devices at the “edge” of the cloud - such as battery life. In the far right columns of Figure 4.4, it seems that there is little difference between the energy consumption of Systems 4 and 5. But Figure 4.6 shows that the lifetime of a battery-fun sensor node in those systems can be dramatically different. The AQ board Photon processor and the Raspberry Pi may both be realistically run on portable batteries, and are often sold as such[135]. Figure 4.7 shows a potential lifetime of the gateway for a full battery charge of 1800mAh and 10,000mAh respectively. As we observed before in Figure 4.4, the AQ board is not the most energy efficient at running machine learning code in isolation, but when considering the larger coordinated system, it greatly benefits from automatically down-sampling

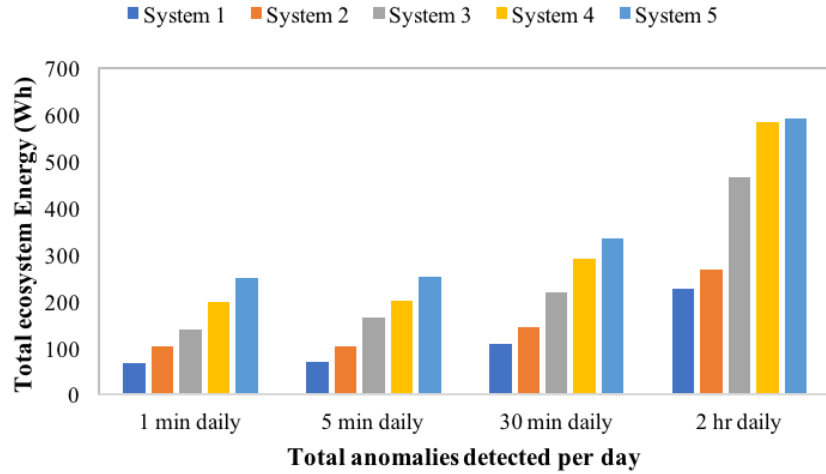


Figure 4.5: The total daily energy consumption of nodes in the ecosystem, for different task allocations across nodes, and different levels of variation in actual air quality sampled

and going into sleep mode when inactive. Where the AQ board transmits via Wi-Fi (to edge server), 69-76% of battery life is consumed for communication. When it transmits data via Bluetooth (to Raspberry Pi 3), it consumes 24.1% by default, and 18.6% when down-sampling after anomaly detection. Comparing System 5 (sending all 1 Hz samples to the edge server) and System 1/3 (sending 30 Hz typical samples and 1 Hz anomalous samples to the Raspberry Pi/edge server), we can multiply the battery lifetime by 22-72x. The lower end of that battery lifetime increase can be achieved if the node eventually detects many anomalies (2 hours total daily), leading to fewer opportunities to down sample, but the higher end of the lifetime increase can be achieved in cases where most data collected is typical, such that only occasional samples have to be delivered to the cloud.

Additionally, using Wi-Fi to communicate from the sensor node to the cloud is more power-expensive than communicating by Bluetooth to a nearby gateway. The use of the anomaly detection engine already reduces new data sample transmission from the sensor node by at least 88.6%, in the case of 2 hours worth of anomalous data throughout the day. If there are fewer anomalies, i.e. fewer unique samples to communicate, the transmission volume can be reduced by 96%. By ar-

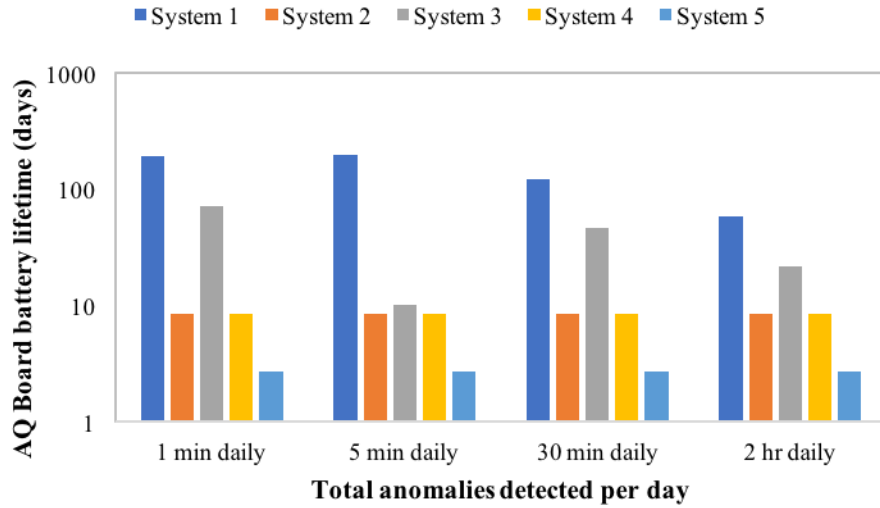


Figure 4.6: Sensor node battery lifetime. Note that y-axis is in log scale.

ranging the tasks such that the sensor node only has to communicate via Bluetooth to the Raspberry Pi, we can extend the battery lifetime of the AQ board by 96.2%, 73.2%, 94.2% and 87.5% for each case of total anomalies detected daily. The mode of communication may dramatically affect the deployment and operations budget. While it may seem more convenient to aggregate data directly from a sensor node to the Internet (a selling point of the Particle Photon we chose), battery limitations may motivate designers to provide connectivity via closer Bluetooth-enabled gateways instead.

It comes as no surprise that low-power sensing devices are far slower at performing machine learning functions than servers when compared head-to-head. However, delegating part of automation tasks to them can yield great benefits. In this section, we have studied the energy effects of delegating the same tasks to different machines and with different communication methods. We found that there is a system-level benefit to running an anomaly detection algorithm on a sensor node or a gateway, yielding far higher battery lifetimes for resource-constrained nodes (up to 72x in specific cases) and lower energy costs in the larger system (22-45%).

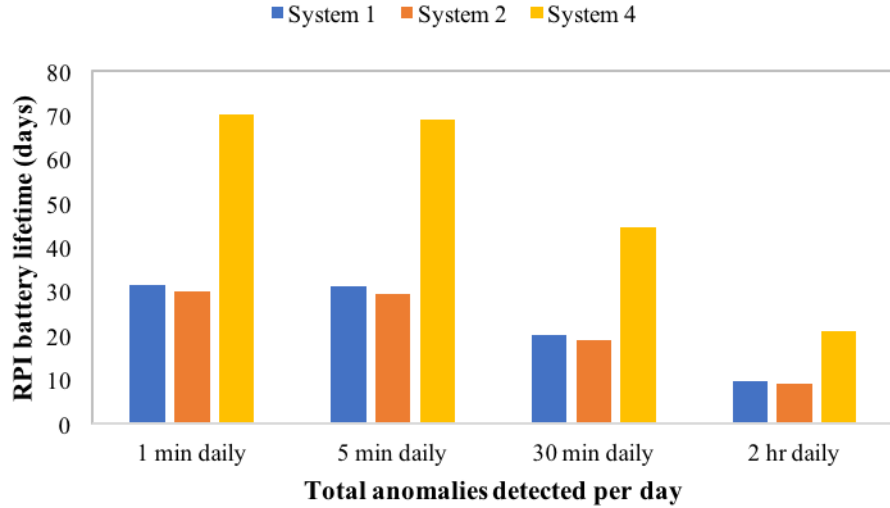


Figure 4.7: Gateway (Raspberry Pi) battery lifetime. Systems 3 and 5 do not utilize Raspberry Pis and are omitted.

4.5 Conclusion

In this chapter, we demonstrate the versatility of our proposed *context engine* framework to be deployed across a hierarchy of devices for a smart health application. The rise of Internet of Things and availability of personal connected sensors enables citizens to have more detailed knowledge of their immediate surroundings. To fully leverage the opportunity that ubiquitous data and vast Internet connectivity provide, we must develop new ways for devices and software to interact and share valuable insights. To integrate personal context with large population-size datasets, we also present a custom sensor platform that connects individual users to the world of data through either a IoT gateway or an edge server, or both. The air quality board we describe in this chapter is a low cost, embedded component of the IoT that enables both experts and lay-users to measure the air quality of their immediate surroundings. Notably, the hardware and software design are intended to be reconfigurable and extendable with minimal effort. Because IoT devices provide both computation and communication of various levels, they give us many different ways to implement the data analysis that is required to process

raw data into high level, human-readable feedback. We propose several examples of how subtasks in a health application might be distributed to various nodes in the system as context engine instantiations, and compare those distributions with respect to energy consumption and device lifetimes. By judiciously allocating machine learning tasks to sensor nodes at the edge of the cloud, we can greatly reduce the volume of data transmission between devices. An anomaly detection engine reduces new data sample transmission from the sensor node by at least 88.6%. Overall, total system energy can be reduced by 22-45%, and the battery lifetimes of power-constrained devices can be lengthened by 22-72x compared to systems that blindly aggregate all sensor data in the cloud.

This chapter contains material from “Context-Aware System Design” by Christine Chan, Michael H Ostertag, Alper Sinan Akyürek, and Tajana Šimunić Rosing, which appears in SPIE Defense + Security. International Society for Optics and Photonics 2017. The dissertation author was the primary investigator and the author of this paper.

Chapter 5

Summary and Future Work

5.1 Thesis Summary

The Internet of Things boasts the potential of pervasive sensing and actuation, with an ever-growing network of platforms seamlessly communicating unprecedented volumes of context data about the devices themselves, as well as users and their environment. However, current software infrastructure and platform design methodologies cannot achieve this vision at reasonable costs. This dissertation presents a middleware that efficiently allocates context extraction efforts across a network of devices, with provably scalable characteristics. We discuss context-aware computing for both isolated, well-engineered machines and networked devices deployed in a more varied environment among human interaction.

5.1.1 Hierarchical context processing middleware

Our modular approach to context-aware IoT application is the context engines framework, where applications are composed of many functional units that contain general-purpose machine learning code. The composition and organization of these units, or engines, can yield lower computational complexity of the system for any non-linear calculation, and guarantees and linear growth with increased system load. We illustrate the ease of implementation with a basic fitness application that includes overlapping sensor data from multiple unreliable sources. Using

a context engine that supplies a generic Dynamic Bayesian Network, we demonstrate the ability to learn and follow different paths of reasoning based on available data, improving accuracy by 60x over using a single stream of unreliable data.

5.1.2 Device context optimization

At the level of a single node, a machine is responsible for monitoring and managing its own operating context. In addition, to operate in the paradigm of the multiple-input single-output context engine framework, it can support the scalability of hierarchical context processing by summarizing several of its context streams into a model. In a detailed model of a high-end server, we study the effects of internal device context variables such as power consumption, heat dissipation, cooling capacity, data bandwidth availability and their interactions. We identify a previously-ignored link between the cooling system and application performance, where commodity drive sensitivity to fans can cause up to 88% lower performance in realistic cooling situations. We also develop and integrate interdependent analytical models for performance, power, thermal, cooling. Finally, we define a multi-model objective function that can be solved to find optimally low-cost fan speeds, saving 19-65% of CPU and fan energy while guaranteeing that critical thermal constraints are still met 100% of the time.

5.1.3 Environmental context in a hierarchy

We investigate some system-level design issues for a sample smart health application that extracts air quality context from sensors and combines that data with larger datasets available in the cloud. We use our context engine infrastructure to demonstrate the benefit of hierarchical processing as opposed to single-stage monolithic implementations across different architectures and devices. By intelligently sending some tasks to the edge of the cloud, and using environmental data to dynamically adjust sensor intervals, we can lower energy costs in the whole system by 22-44% and improve battery lifetimes for resource-constrained nodes by 22-72x compared to systems that aggregate all sensor data to the cloud.

5.2 Future Work

Although we have commonly used medical applications as examples in this thesis, the system infrastructure, hierarchical machine learning, privacy and aggregate data communication features we discuss can be used for application development across other IoT verticals like smart grid, smart cities, and intelligent environments. The idea of using general machine learning techniques in specific IoT applications is gaining traction in many commercial projects [136]. For example, Google Assistant SDK (also branded as “AIY projects” [137]) abstracts away expert machine learning knowledge, letting users simply choose the class of machine learning techniques and plug-and-play with their own sensed inputs.

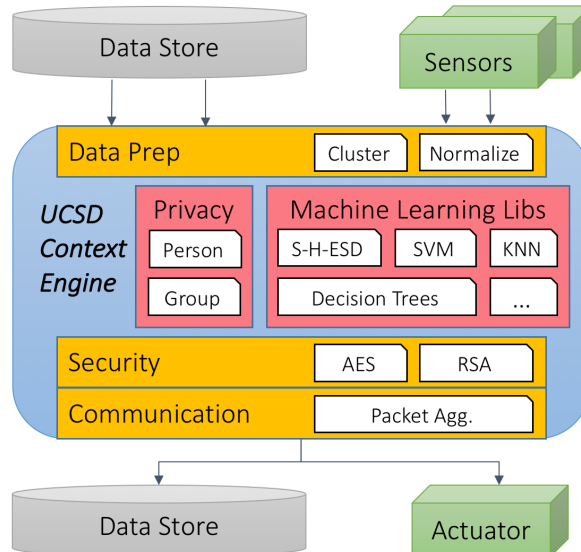


Figure 5.1: Proposed internal structure of a context engine. Current iteration does not provide privacy mechanisms.

The *context engine*, as discussed in Chapter 2, is under development and has been showcased at several venues. Code in development can be found at <https://github.com/UCSD-SEELab/ContextEngine27>. As of June 2017, it is implemented in Python 2.7 and runs on Desktop Ubuntu [138] and Raspberry Pi [132]. As shown in Figure 5.1, it provides several different clustering, anomaly detection, and regression algorithms along with test scripts for users to quickly train and

test new data. It can receive serial data streams as data input, or interface with the Global Data Plane, a distributed data platform for the IoT developed at UC Berkeley [139]. Some specific kernels are available in C++ to be run on smaller devices such as Arduino microcontrollers, and were utilized for the experiments in Chapter 4. The data security and communication functionalities have been tested independently but are yet to be integrated into the framework.

While some IoT frameworks address important deployment issues, such as reducing the communication overhead and congestion caused by a large distributed network of these sensors [140], most do not adequately address privacy or security concerns that come with handling sensitive medical information. To secure data in transmission, we have prototyped some context engines that supply RSA key generation and AES encryption. Further work is required to integrated this as a part of the class output interface.

Aside from end-to-end data security using encryption, maintaining the privacy of data content is also vital. At the IoT scale, the impact of any privacy breaches is exponentially increased. Differential privacy techniques are particularly relevant in the IoT where many personal data streams are combined and mined for more abstract information about populations [141, 142]. These techniques obfuscate identifiable personal data by adding data noise to aggregated data, while preserving functionality of the main processing application. This tradeoff should be monitored and implemented in context engines that serve as data aggregators or gateways between sensor nodes and the cloud.

Large scale IoT deployments depend on being able to guarantee safety and privacy of individuals who use them. Solving these issues is critical to acceptance and adoption. Further work on our context engine framework must integrate security, privacy, and energy-efficiency while providing objective, comprehensive information in timely fashion. The generated models and population-level trends from our hierarchical configuration of context engines will provide unprecedented insights from an otherwise unwieldy amount of data. Once complete, our framework will enable scalable, secure and privacy-aware processing of IoT data, more suitable for medicine and other critical applications.

Bibliography

- [1] “Gartner Says the Internet of Things Will Transform the Data Center.” [Online]. Available: <http://www.gartner.com/newsroom/id/2684616>
- [2] “White paper: Cisco VNI Forecast and Methodology, 2015-2020.” [Online]. Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>
- [3] J. A. Stankovic, “Research Directions for the Internet of Things,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, Feb. 2014.
- [4] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Communications, Surveys, and Tutorials*, pp. 414–454, 2013.
- [5] J. Hammer and T. Yan, “Poster: A virtual sensing framework for mobile phones,” in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services (MobiSys)*, 2014.
- [6] S. K. Madhu, V. C. Raj, and R. M. Suresh, “An ontology-based framework for context-aware adaptive e-learning system,” in *Proceedings of the International Conference on Computer Communication and Informatics (ICCI)*, 2013.
- [7] H. H. Liu, *Software performance and scalability: a quantitative approach*. John Wiley & Sons, 2011, vol. 7.
- [8] F. Moore, “Tiered storage takes center stage,” *Horison Information Strategies*, vol. 22, 2011.
- [9] D. Anderson, J. Dykes, and E. Riedel, “More than an interface-SCSI vs. ATA.” in *FAST*, vol. 2, 2003, p. 3.
- [10] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and

- S. Yang, “The ramcloud storage system,” *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 7, 2015.
- [11] C. S. Chan, Y. Jin, Y.-K. Wu, K. Gross, K. Vaidyanathan, and T. Rosing, “Fan-speed-aware scheduling of data intensive jobs,” in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 409–414.
- [12] Y. Katsis, C. Baru, T. Chan, S. Dasgupta, C. Farcas, W. Griswold, J. Huang, L. Ohno-Machado, Y. Papakonstantinou, F. Raab, and K. Patrick, “Delphi: Data e-platform for personalized population health,” in *e-Health Networking, Applications & Services (Healthcom), 2013 IEEE 15th International Conference on*. IEEE, 2013, pp. 115–119.
- [13] J. Turner, “Effects of data center vibration on compute system performance,” in *Proceedings of the First USENIX conference on Sustainable information technology*. USENIX Association, 2010, pp. 5–5.
- [14] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, “Fingerprinting the datacenter: automated classification of performance crises,” in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 111–124.
- [15] TPC-H, “TPC-H Benchmark Suite,” 2011. [Online]. Available: <http://www.tpc.org/tpch/>
- [16] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of things for smart cities,” *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.
- [17] M. Pradhan, F. Gökgöz, N. Bau, and D. Ota, “Approach towards application of commercial off-the-shelf internet of things devices in the military domain,” in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 2016, pp. 245–250.
- [18] P. Fraga-Lamas, T. M. Fernández-Caramés, M. Suárez-Albela, L. Castedo, and M. González-López, “A review on internet of things for defense and public safety,” *Sensors*, vol. 16, no. 10, p. 1644, 2016.
- [19] N. Suri, M. Tortonesi, J. Michaelis, P. Budulas, G. Benincasa, S. Russell, C. Stefanelli, and R. Winkler, “Analyzing the applicability of internet of things to the battlefield environment,” in *Military Communications and Information Systems (ICMCIS), 2016 International Conference on*. IEEE, 2016, pp. 1–8.

- [20] M. Rudary, S. Singh, and M. E. Pollack, “Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 91.
- [21] M. R. V. Moghadam, R. T. Ma, and R. Zhang, “Distributed frequency control in smart grids via randomized demand response,” *IEEE Transactions on Smart Grid*, vol. 5, no. 6, pp. 2798–2809, 2014.
- [22] H. K. M. Paredes, A. Costabeber, and P. Tenti, “Application of conservative power theory to cooperative control of distributed compensators in smart grids,” in *Nonsinusoidal Currents and Compensation (ISNCC), 2010 International School on*. IEEE, 2010, pp. 190–196.
- [23] J. Venkatesh, B. Aksanli, J.-C. Junqua, P. Morin, and T. S. Rosing, “Homesim: Comprehensive, smart, residential electrical energy simulation and scheduling,” in *Green Computing Conference (IGCC), 2013 International*. IEEE, 2013, pp. 1–8.
- [24] L. Klein, J.-y. Kwak, G. Kavulya, F. Jazizadeh, B. Becerik-Gerber, P. Varakantham, and M. Tambe, “Coordinating occupant behavior for building energy and comfort management using multi-agent systems,” *Automation in construction*, vol. 22, pp. 525–536, 2012.
- [25] C. Pang, V. Vyatkin, Y. Deng, and M. Sorouri, “Virtual smart metering in automation and simulation of energy-efficient lighting system,” in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.
- [26] P. Yi, X. Dong, A. Iwayemi, C. Zhou, and S. Li, “Real-time opportunistic scheduling for residential demand response,” *IEEE Transactions on smart grid*, vol. 4, no. 1, pp. 227–234, 2013.
- [27] J. Venkatesh, B. Aksanli, and T. S. Rosing, “Residential energy simulation and scheduling: A case study approach,” in *Computers and Communications (ISCC), 2013 IEEE Symposium on*. IEEE, 2013, pp. 000 161–000 166.
- [28] N. Banerjee, S. Rollins, and K. Moran, “Automating energy management in green homes,” in *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*. ACM, 2011, pp. 19–24.
- [29] A. Mishra, D. Irwin, P. Shenoy, J. Kurose, and T. Zhu, “Smartcharge: Cutting the electricity bill in smart homes with energy storage,” in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, 2012, p. 29.

- [30] Z. Huang, T. Zhu, Y. Gu, D. Irwin, A. Mishra, and P. Shenoy, "Minimizing electricity costs by sharing energy in sustainable microgrids," in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, 2014, pp. 120–129.
- [31] T. Zhu, Z. Huang, A. Sharma, J. Su, D. Irwin, A. Mishra, D. Menasche, and P. Shenoy, "Sharing renewable energy in smart microgrids," in *Cyber-Physical Systems (ICCPs), 2013 ACM/IEEE International Conference on*. IEEE, 2013, pp. 219–228.
- [32] J.-H. Hong, S.-I. Yang, and S.-B. Cho, "Conamsn: A context-aware messenger using dynamic bayesian networks with wearable sensors," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4680–4686, 2010.
- [33] S. Lee and K. C. Lee, "Context-prediction performance by a dynamic bayesian network: Emphasis on location prediction in ubiquitous decision support environment," *Expert Systems with Applications*, vol. 39, no. 5, p. 49084914, 2012.
- [34] "Google Now," Dec. 2014. [Online]. Available: <http://www.androidcentral.com/google-now>
- [35] K. Lee, J. Flinn, T. J. Giuli, B. Noble, and C. Peplin, "Amc: verifying user interface properties for vehicular applications," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013, pp. 1–12.
- [36] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE transactions on knowledge and data engineering*, vol. 23, no. 4, pp. 527–539, 2011.
- [37] S. Staab and R. Studer, *Handbook of Ontologies*. Springer Science and Business, 2010.
- [38] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *The knowledge engineering review*, vol. 18, no. 03, pp. 197–207, 2003.
- [39] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, "An ontology-based context model in intelligent environments," in *Proceedings of communication networks and distributed systems modeling and simulation conference*, vol. 2004. San Diego, CA, USA., 2004, pp. 270–275.
- [40] W. Wang, "A comprehensive ontology for knowledge representation in the internet of things," in *11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

- [41] M. Nebeling, M. Grossniklaus, S. Leone, and M. Norrie, “Xcml: providing context-aware language extensions for the specification of multi-device web applications,” *World Wide Web (WWW)*, vol. 15, no. 4, pp. 447–481, 2012.
- [42] C. Perera, A. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos, “Context aware sensor configuration model for internet of things,” in *Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035*. CEUR-WS. org, 2013, pp. 253–256.
- [43] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “A survey of middleware for internet of things,” in *Recent Trends in Wireless and Mobile Networks*. Springer, 2011, pp. 288–296.
- [44] J. Venkatesh, C. Chan, A. S. Akyurek, and T. S. Rosing, “A modular approach to context-aware IoT applications,” in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 235–240.
- [45] J. Venkatesh, B. Aksanli, C. S. Chan, A. S. Akyürek, and T. S. Rosing, “Scalable-application design for the IoT,” *IEEE Software*, vol. 34, no. 1, pp. 62–70, 2017.
- [46] W. Cui, Y. Kim, and T. S. Rosing, “Cross-platform machine learning characterization for task allocation in IoT ecosystems,” in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2017, pp. 1–7.
- [47] B. O. Akyürek, A. S. Akyürek, J. Kleissl, and T. Šimunić Rosing, “Tesla: Taylor expanded solar analog forecasting,” in *Proceedings of IEEE Smart-GridComm '13*, 2014.
- [48] A. Battaglini, J. Lilliestam, A. Haas, and A. Patt, “Development of supersmart grids for a more efficient utilisation of electricity from renewable sources,” *Journal of Cleaner Production*, 2009.
- [49] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, 2013.
- [50] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, “Smart semantic middleware for the internet of things,” in *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*, 2008.

- [51] B. O. Akyurek, A. S. Akyurek, J. Kleissl, and T. Š. Rosing, “TESLA: Taylor expanded solar analog forecasting,” in *Smart Grid Communications (Smart-GridComm), 2014 IEEE International Conference on*. IEEE, 2014, pp. 127–132.
- [52] S. Zahedi and C. Bisdikian, “A framework for qoi-inspired analysis for sensor network deployment planning,” in *WICON '07*, 2007.
- [53] P. Jogalekar and M. Woodside, “Evaluating the scalability of distributed systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 6, pp. 589–603, 2000.
- [54] D. Uckelmann, M. Harrison, and F. Michahelles, *Architecting the Internet of Things*. Springer Berlin Heidelberg, 2011, ch. An Architectural Approach Towards the Future Internet of Things, pp. 1–24.
- [55] J. Moar, “Smart health & fitness wearables: Device strategies, trends & forecasts 2014-2019,” *Juniper Research*, 2014.
- [56] “Fitbit Developer API.” [Online]. Available: <https://dev.fitbit.com/>
- [57] “Moves - Activity Diary for iPhone and Android.” [Online]. Available: <https://moves-app.com/>
- [58] “foursquare for Developers.” [Online]. Available: <https://developer.foursquare.com/>
- [59] B. Aksanli and T. Rosing, “Providing regulation services and managing data center peak power budgets,” in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 143.
- [60] C. Miller. Solar powered data centers. [Online]. Available: <http://www.datacenterknowledge.com/solar-powered-data-centers/>
- [61] A. Capozzoli and G. Primiceri, “Cooling systems in data centers: state of art and emerging technologies,” *Energy Procedia*, vol. 83, pp. 484–493, 2015.
- [62] M. K. Patterson, “The effect of data center temperature on energy efficiency,” in *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. IThERM 2008. 11th Intersociety Conference on*. IEEE, 2008, pp. 1167–1174.
- [63] R. F. Sullivan, “Alternating cold and hot aisles provides more reliable cooling for server farms,” *Uptime Institute*, 2000.

- [64] J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma, “Making scheduling” cool”: Temperature-aware workload placement in data centers.” in *USENIX annual technical conference, General Track*, 2005, pp. 61–75.
- [65] R. Ayoub, S. Sharifi, and T. S. Rosing, “Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 295–298.
- [66] L. A. Barroso, J. Clidaras, and U. Hözlze, “The datacenter as a computer: An introduction to the design of warehouse-scale machines,” *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.
- [67] K. Vaidyanathan, K. Gross, and S. Sondur, “Ambient temperature optimization for enterprise servers: Key to large-scale energy savings,” in *Energy Efficient Electronic Systems (E3S), 2015 Fourth Berkeley Symposium on*. IEEE, 2015, pp. 1–3.
- [68] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, “Learning-based query performance modeling and prediction,” in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 390–401.
- [69] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, “Predicting multiple metrics for queries: Better decisions enabled by machine learning,” in *Data Engineering, 2009. ICDE’09. IEEE 25th International Conference on*. IEEE, 2009, pp. 592–603.
- [70] H. Jung, P. Rong, and M. Pedram, “Stochastic modeling of a thermally-managed multi-core system,” in *Proceedings of the 45th annual Design Automation Conference*. ACM, 2008, pp. 728–733.
- [71] S. Sankar and K. Vaid, “Storage characterization for unstructured data in online services applications,” in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 148–157.
- [72] C. Delimitrou, S. Sankar, B. Khessib, K. Vaid, and C. Kozyrakis, “Time and cost-efficient modeling and generation of large-scale tpcc/tpce/tpch workloads,” in *Topics in Performance Evaluation, Measurement and Characterization*. Springer, 2012, pp. 146–162.
- [73] T. M. Ruwart and Y. Lu, “Performance impact of external vibration on consumer-grade and enterprise-class disk drives,” in *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on*. IEEE, 2005, pp. 307–315.

- [74] J. Yang, C. P. Tan, Z. He, Z. Y. Ching, and C. C. Tan, "An effective system-level vibration prediction analysis approach for data storage system chassis," *Microsystem Technologies*, pp. 1–9, 2016.
- [75] D. Anderson, K. Green, O. Herrera, K. Schneebeli, and P. Urbisci, "Disk-drive chassis for reducing transmission of vibrations between disk-drive units of a disk-drive array," Nov. 28 2000, uS Patent 6,154,361. [Online]. Available: <https://www.google.com/patents/US6154361>
- [76] X. Tan, B. G. Chen, B. J. Zhang, B. C. Liu, B. N. Ahuja, and I. J. Zhang, "An advanced rack server system design for rotational vibration (rv) performance," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2016 15th IEEE Intersociety Conference on*. IEEE, 2016, pp. 1320–1325.
- [77] Y. Uzuka, K. Itakura, N. Yamaoka, and T. Kitamura, "Power supply, cooling and mechanical technologies for green it," *Fujitsu Sci. Tech. J*, vol. 47, no. 2, pp. 157–163, 2011.
- [78] M. A. Bell, "Use best practices to design data center facilities," *Gartner Research. April*, vol. 22, 2005.
- [79] S. Legtchenko, X. Li, A. I. Rowstron, A. Donnelly, and R. Black, "Flamingo: Enabling evolvable hdd-based near-line storage." in *FAST*, 2016, pp. 213–226.
- [80] S. Sankar, M. Shaw, K. Vaid, and S. Gurumurthi, "Datacenter scale evaluation of the impact of temperature on hard disk drive failures," *ACM Transactions on Storage (TOS)*, vol. 9, no. 2, p. 6, 2013.
- [81] K. Ebrahimi, G. F. Jones, and A. S. Fleischer, "A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities," *Renewable and Sustainable Energy Reviews*, vol. 31, pp. 622–638, 2014.
- [82] A. Sridhar, A. Vincenzi, D. Atienza, and T. Brunschwiler, "3d-ice: A compact thermal model for early-stage design of liquid-cooled ics," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2576–2589, 2014.
- [83] M. Zapater, J. L. Ayala, J. M. Moya, K. Vaidyanathan, K. Gross, and A. K. Coskun, "Leakage and temperature aware server control for improving energy efficiency in data centers," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 266–269.
- [84] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: why some (might)

- like it hot,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 163–174, 2012.
- [85] S. Sankar, M. Shaw, and K. Vaid, “Impact of temperature on hard disk drive reliability in large datacenters,” in *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 2011, pp. 530–537.
- [86] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *ACM SIGARCH Computer Architecture News*, vol. 34. IEEE Computer Society, 2006, pp. 78–88.
- [87] A. Pahlevan, J. Picorel, A. P. Zarandi, D. Rossi, M. Zapater, A. Bartolini, P. G. Del Valle, D. Atienza, L. Benini, and B. Falsafi, “Towards near-threshold server processors,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 7–12.
- [88] Q.-G. Wang, T.-H. Lee, H.-W. Fung, Q. Bi, and Y. Zhang, “PID tuning for improved performance,” *Control Systems Technology, IEEE Transactions on*, vol. 7, no. 4, pp. 457–465, 1999.
- [89] J. Kim, M. M. Sabry, D. Atienza, K. Vaidyanathan, and K. Gross, “Global fan speed control considering non-ideal temperature measurements in enterprise servers,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–6.
- [90] C. E. Bash, C. D. Patel, and R. K. Sharma, “Dynamic thermal management of air cooled data centers,” in *Thermal and Thermomechanical Phenomena in Electronics Systems, 2006. IThERM’06. The Tenth Intersociety Conference on*. IEEE, 2006, pp. 8–pp.
- [91] L. Parolini, B. Sinopoli, B. H. Krogh, and Z. Wang, “A cyber–physical systems approach to data center modeling and control for energy efficiency,” *Proceedings of the IEEE*, vol. 100, no. 1, pp. 254–268, 2012.
- [92] J. Meza, M. A. Shah, P. Ranganathan, M. Fitzner, and J. Veazey, “Tracking the power in an enterprise decision support system,” in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 261–266.
- [93] C. S. Chan, B. Pan, K. Gross, K. Vaidyanathan, and T. Š. Rosing, “Correcting vibration-induced performance degradation in enterprise servers,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 3, pp. 83–88, 2014.

- [94] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*. IEEE, 2002, pp. 17–28.
- [95] A. Coskun, J. Ayala, D. Atienza, T. Rosing, and Y. Leblebici, "Dynamic thermal management in 3d multicore architectures," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, Apr. 2009, pp. 1410–1415.
- [96] R. Ayoub, R. Nath, and T. Rosing, "JETC: Joint energy thermal and cooling management for memory and cpu subsystems in servers," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–12.
- [97] R. Ayoub, K. Indukuri, and T. S. Rosing, "Temperature aware dynamic workload scheduling in multsocket cpu servers," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 9, pp. 1359–1372, 2011.
- [98] C. Hankendi, A. K. Coskun, and H. Hoffmann, "Adapt&cap: Coordinating system-and application-level adaptation for power-constrained systems," *IEEE Design & Test*, vol. 33, no. 1, 2016.
- [99] "K-Series Electrodynamic Shakers | Unholtz Dickie." [Online]. Available: <http://www.udco.com/products/electrodynamic-shaker-systems/k-series/>
- [100] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning with application to clustering with side-information," *Advances in neural information processing systems*, vol. 15, pp. 505–512, 2003.
- [101] R. L. Thorndike, "Who belongs in the family?" *Psychometrika*, vol. 18, no. 4, pp. 267–276, Dec. 1953.
- [102] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments." International Symposium on Computer Architecture-IEEE, 2006.
- [103] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.

- [104] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate Temperature-dependent Integrated Circuit Leakage Power Estimation is Easy," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '07. San Jose, CA, USA: EDA Consortium, 2007, pp. 1526–1531.
- [105] J. M. Borwein and A. S. Lewis, *Convex analysis and nonlinear optimization: theory and examples*. Springer Science & Business Media, 2010.
- [106] SPEC, "SPEC CPU 2006 Benchmarks," 2006. [Online]. Available: <https://www.spec.org/cpu2006/>
- [107] I. H.-P. N. Dell, "IPMI intelligent platform management interface specification second generation," *V2. 0*, vol. 590, 2009.
- [108] D. Valrio and J. S. da Costa, "Tuning of fractional PID controllers with ZieglerNichols-type rules," *Signal Processing*, vol. 86, no. 10, pp. 2771–2784, 2006.
- [109] D. Ayres-Sampaio, A. C. Teodoro, N. Sillero, C. Santos, J. Fonseca, and A. Freitas, "An investigation of the environmental determinants of asthma hospitalizations: an applied spatial approach," *Applied Geography*, vol. 47, pp. 10–19, 2014.
- [110] H. R. Anderson, B. K. Butland, A. van Donkelaar, M. Brauer, D. P. Strachan, T. Clayton, R. van Dingenen, M. Amann, B. Brunekreef, A. Cohen, F. Dentener, C. Lai, L. N. Lamsai, and R. V. Martin, "Satellite-based estimates of ambient air pollution and global variations in childhood asthma prevalence," Ph.D. dissertation, University of British Columbia, 2015.
- [111] N.-H. Hsieh and C.-M. Liao, "Fluctuations in air pollution give risk warning signals of asthma hospitalization," *Atmospheric environment*, vol. 75, pp. 206–216, 2013.
- [112] J. E. Thompson, "Crowd-sourced air quality studies: A review of the literature & portable sensors," *Trends in Environmental Analytical Chemistry*, vol. 11, pp. 23–34, 2016.
- [113] J. K. Y. Ng, J. Wang, K. Y. Lam, C. H. C. Kam, and S. Han, "Capturing and Analyzing Pervasive Data for SmartHealth," in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, May 2014, pp. 985–992.
- [114] J. F. Sallis, B. E. Saelens, L. D. Frank, T. L. Conway, D. J. Slymen, K. L. Cain, J. E. Chapman, and J. Kerr, "Neighborhood built environment and income: Examining multiple health outcomes," *Social Science & Medicine*, vol. 68, no. 7, pp. 1285–1293, Apr. 2009.

- [115] T. Space, “AirBeam Technical Specifications, Operation & Performance: Taking Space.” [Online]. Available: <http://www.takingspace.org/airbeam-technical-specifications-operation-performance/>
- [116] O. Zaouak, A. B. Daoud, M. Fages, J.-L. Fanlo, and B. Aubert, “High performance cost effective miniature sensor for continuous network monitoring of h2s,” *CHEMICAL ENGINEERING*, vol. 30, 2012.
- [117] A. Antoni, M. Marjanovi, K. Pripui, and I. Podnar arko, “A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things,” *Future Generation Computer Systems*, vol. 56, pp. 607–622, Mar. 2016.
- [118] A. S. Akyurek and T. S. Rosing, “Optimal in-network packet aggregation policy for maximum information freshness,” in *2016 European Conference on Networks and Communications (EuCNC)*, Jun. 2016, pp. 89–93.
- [119] P. H. Azevêdo Filho, M. F. Caetano, and J. L. Bordim, “A packet aggregation mechanism for real time applications over wireless networks,” *International Journal of Networking and Computing*, vol. 2, no. 1, pp. 18–40, 2012.
- [120] “Air Quality Index (AQI) Basics.” [Online]. Available: <https://airnow.gov/index.cfm?action=aqibasics.aqi>
- [121] N. Nikzad, C. Ziftci, P. Zappi, N. Quick, P. Aghera, N. Verma, B. Demchak, K. Patrick, H. Shacham, T. S. Rosing, I. Krueger, W. Griswold, and S. Dasgupta, *CitiSense: Adaptive Services for Community-driven Behavioral and Environmental Monitoring to Induce Change*. Department of Computer Science and Engineering, University of California, San Diego, 2011.
- [122] P. Zappi, E. Bales, J. H. Park, W. Griswold, and T. . Rosing, “The CitiSense air quality monitoring mobile sensor node,” in *Proceedings of the 11th ACM/IEEE Conference on Information Processing in Sensor Networks, Beijing, China, 2012*.
- [123] J. M. Daisey, W. J. Angell, and M. G. Apte, “Indoor air quality, ventilation and health symptoms in schools: an analysis of existing information,” *Indoor air*, vol. 13, no. 1, pp. 53–64, 2003.
- [124] D. G. Shendell, R. Prill, W. J. Fisk, M. G. Apte, D. Blake, and D. Faulkner, “Associations between classroom co2 concentrations and student attendance in washington and idaho,” *Indoor air*, vol. 14, no. 5, pp. 333–341, 2004.
- [125] T. Labeodan, W. Zeiler, G. Boxem, and Y. Zhao, “Occupancy measurement in commercial office buildings for demand-driven control applicationsa survey and detection system evaluation,” *Energy and Buildings*, vol. 93, pp. 303–314, 2015.

- [126] N. Masson, R. Piedrahita, and M. Hannigan, “Quantification Method for Electrolytic Sensors in Long-Term Monitoring of Ambient Air Quality,” *Sensors*, vol. 15, no. 10, pp. 27 283–27 302, Oct. 2015. [Online]. Available: <http://www.mdpi.com/1424-8220/15/10/27283/>
- [127] “Particle Photon.” [Online]. Available: <https://docs.particle.io/datasheets/photon-datasheet/>
- [128] “Particle Electron.” [Online]. Available: <https://docs.particle.io/datasheets/electron-datasheet/>
- [129] “ICIS-Air Download Summary and Data Element Dictionary | ECHO | US EPA.” [Online]. Available: <https://echo.epa.gov/tools/data-downloads/icis-air-download-summary>
- [130] “California Environmental Health Tracking.” [Online]. Available: <http://cehtp.org/page/main>
- [131] A. M. Johansen, “SMCTC: Sequential monte carlo in C++,” *Journal of Statistical Software*, vol. 30, no. 6, pp. 1–41, 4 2009. [Online]. Available: <http://www.jstatsoft.org/v30/i06>
- [132] “Raspberry Pi 3 Model B.” [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- [133] “Intel Xeon Processor D Family.” [Online]. Available: <http://www.intel.com/content/www/us/en/products/processors/xeon/d-processors.html>
- [134] “Edge Gateway 5000 | Dell United States.” [Online]. Available: <http://www.dell.com/us/business/p/dell-edge-gateway-5000/pd>
- [135] “USB Battery Pack for Raspberry Pi - 10000mah - 2 x 5v outputs ID: 1566 - \$39.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” [Online]. Available: <https://www.adafruit.com/product/1566>
- [136] A. A, G. W, and L. F, “Change and anomaly detection framework for internet of things data streams,” Jun. 17 2016. [Online]. Available: <https://software.intel.com/en-us/articles/change-and-anomaly-detection-framework-for-internet-of-things-data-streams>
- [137] “Google aiy projects.” [Online]. Available: <https://aiyprojects.withgoogle.com>
- [138] “Ubuntu release 14.04 trusty tahr.” [Online]. Available: <http://releases.ubuntu.com/14.04/>

- [139] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiawicz, “The cloud is not enough: Saving iot from the cloud.” in *HotCloud*, 2015.
- [140] F. Hu, M. Jiang, L. Celentano, and Y. Xiao, “Robust medical ad hoc sensor networks (masn) with wavelet-based eeg data mining,” *Ad Hoc Networks*, vol. 6, no. 7, pp. 986–1012, 2008.
- [141] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Theory of Cryptography Conference*. Springer, 2006, pp. 265–284.
- [142] D. Kifer and A. Machanavajjhala, “Pufferfish: A framework for mathematical privacy definitions,” *ACM Transactions on Database Systems (TODS)*, vol. 39, no. 1, p. 3, 2014.