Energy Adaptive Digital Ecosystems

by

Andreas Christoph Bergen
M.Sc., Computer Science, University of Victoria, 2013
B.Sc., Computer Science, University of Victoria, 2011
B.A., Political Science and History, University of Victoria, 2007

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Energy Adaptive Digital Ecosystems

by

Andreas Christoph Bergen
M.Sc., Computer Science, University of Victoria, 2013
B.Sc., Computer Science, University of Victoria, 2011
B.A., Political Science and History, University of Victoria, 2007

Supervisory Committee

Dr. H. A. Müller, Supervisor
(Department of Computer Science)

Dr. Y. Coady, Supervisor
(Department of Computer Science)

Dr. S. Neville, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Since the turn of the century, the proliferation of virtualization and cloud computing has led to an increase in data centres and consequently an increase in power consumption for computing. Today, approximately 2% of global energy consumption is attributed to data centres alone. As a result, optimizing power usage effectiveness in enterprise data centres has become a laudable goal and a critical requirement in IT operations all over the world. While a significant body of research exists to measure, monitor, and control the "greenness" level of hardware components, significant research is needed to relate hardware energy consumption to energy consumption stemming from (software) program execution. In this dissertation, we argue that the true energy cost of program execution must focus on the digital ecosystem within which a particular software program is executed. We investigate the interplay between energy consumption, task scheduling and execution decision making using dynamic runtime models of digital ecosystems based on the execution context of software.

Single instances of software applications are no longer confined to a single device or machine. Instead software commonly interacts with resources and services outside of its own hardware unit. The scope of this interaction defines the application's digital ecosystem. Smartphones interact with cloud resources; cloud resources include databases, specialized compute or storage clouds, specialized hardware and virtual machines (VMs). Combining processes of varying complexity with varying resource allocations produces different energy consumption levels. The challenge is to investigate the variability of software process orchestration based on a power consumption framework to accrue and optimize energy savings in digital ecosystems. The contributions of this dissertation include: *i*) an adaptive energy consumption framework; *ii*) self-adaptive energy management systems based on this framework; *iii*) deployment mechanisms for applications to use this framework; *iv*) models at runtime models for self-adaptive energy management systems. Our ultimate goal is to develop smart, self-adaptive, green computing techniques, such as adaptive job scheduling and resource provisioning, to reduce overall power consumption in data centres, on individual devices (e.g., mobile, desktop, laptop or server), and in digital ecosystems.

Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family, for their support throughout my time in graduate school,

Hausi Müller, for his mentorship, guidance and support,

Yvonne Coady, for enthusiasm, support, encouragement, patience and persistence.

UVic's EDC2 staff, without their support and willingness to allow us insights into their metrics none of this research would have been possible. Their collaboration allowed us to obtain very valuable real world data.

The hardest thing is to go to sleep at night, when there are so many urgent things needing to be done. A huge gap exists between what we know is possible with today's machines and what we have so far been able to finish.
Donald Knuth

DEDICATION

I dedicate this dissertation to my family, friends, colleagues and supervisors who have supported and encouraged me throughout this journey.

Chapter 1

Introduction and Motivation

Our ultimate goal is to develop smart, self-adaptive, green computing techniques, such as adaptive job scheduling and resource provisioning, to reduce overall power consumption in data centres, on individual devices (e.g., mobile, desktop, laptop or server), and in digital ecosystems. Accordingly, with this goal in mind we address and highlight our work from the aspect of experimental research as well as the collection of results with real hardware that addresses challenges in the field of Green Computing; further, our approach also situates this dissertation in the larger context of future research avenues focused on CPS, IoT and smart cities.

Green computing generally refers to "the practice of using computing resources more efficiently [or sustainably] while maintaining or increasing overall performance" [54]. This definition is intentionally very broad and includes topics such as Green IT, supply chain optimization, marketing green computing to customers, corporate emissions and many more areas well beyond the scope of a dissertation topic. For the purpose of this dissertation, we limit the meaning of green computing to "the practice of maximizing the efficient use of computing resources to minimize environmental impact" within data centres, a definition often adopted in Computer Science and Software Engineering research when discussing green computing [54]. Thus, for us, the term *efficient* is used in the context of reducing the duration of time and/or energy consumption for a given task. This definition and the findings of this dissertation apply to servers in data centres, mobile devices and digital ecosystems.

This chapter provides motivation for our research and discusses fundamental research challenges of the past, present and future in the field of green computing. This

field spans a wide and varied number of topics. For this dissertation, we focus on power profiles of data centre servers, desktops, laptops and mobile devices as well as their operations in digital ecosystems. The research in digital ecosystems closely relates to open problems and research challenges in the areas of Internet of Things (IoT) and smart Cyber Physical Systems (CPS).

The goal of our research is to understand how software systems consume energy in digital ecosystems. Our contributions include a framework to characterize and measure energy consumption, including run time models for energy optimization, canonical use cases, as well as repeatable micro and macro benchmarks.

We now provide a brief overview of the subject domain and situate our research and contributions in the context of previous research approaches in this field.

1.1 Motivating Software Energy Consumption and its Optimization

While the availability of large arrays of real hardware is a practical limitation to conduct experiments for our research, we can still tackle many open problems in the space of green computing head-on. The main, overarching open problem is simply how to reduce software and hardware related energy consumption and in turn energy costs. This problem applies to single standalone and federated data centres alike. With the continuous evolution and proliferation of digital ecosystems, the research focus also changes continuously. We aim to provide approaches and solutions to energy consumption management that can adapt with this continuous evolution and thus will apply to future digital ecosystems.

Cloud computing has successfully established itself as a paradigm in the computing and IT world. Compute and storage clouds are key to the exploding fields of data science and machine learning. The clouds' data centres provide necessary capacity in storage, memory, network bandwidth and compute power to store and analyze data. A large share of the energy demand of data centres is due to cooling servers; however, the actual power demands of servers are not to be underestimated and contribute significantly to the overall data centre's electricity consumption [12]. Raw compute power and the ability to store and access data, at a large scale, has a significant im-

pact on energy consumption of worldwide. As such, power consumption and efficacy of power usage in enterprise data centres is a critical research area with many open and challenging problems.

Power consumption in modern data centres has been a growing concern for enterprises and governments in recent years. Koomey argues that on a global scale, power consumption of data centres amounts to approximately 2% of energy demands [75]. With continually increasing sensing and actuating capabilities as well as data collection and digital content generation (e.g., smart cities), the use of cloud, edge and fog computing is expected to increase dramatically. Amazon engineer Hamilton emphasizes that power has become the most important factor for high-scale data centre operators [53]. He further reveals that the efforts to reduce energy consumption are made even at the costs of individual servers' performance—servers account for approximately 42% of energy consumption of data centres, 19% of which is direct consumption (i.e., server hardware) and 23% stems from indirect consumption (i.e., cooling) [53].

Energy demands of servers can vary significantly between idle states and full server utilization states [94]. Kansal and Zhao from Microsoft Research, in collaboration with Srikantaiah (Penn State), point out that even at very low utilization, below 10%, power demands of servers can be as high as 50% of the power draw which a server would incur at full utilization [119]. In other words, running a server at full CPU utilization can more than double the electricity consumption compared to its idle state. Table 1.1 illustrates this phenomenon. We see that the energy consumption of an idle server is approximately half that of a server at full utilization.

To understand the impact of energy consumption and optimization within data centres fully, one needs to be aware of the fact that a server's electricity demands varies significantly with the types of applications being run. We have found through our experiments that CPU intensive applications will alter the power consumption profile in a different way than memory intensive applications. Generally, different resource intensive applications will alter the power profiles of the servers on which these applications run in different ways. Understanding the causes of these differences in workload based energy consumption and heterogeneous hardware environments can be beneficial for energy optimization research. This dissertation aims to fill the gaps

Table 1.1: Energy consumption of servers located in the UVic EDC2 data centre confirm prior research findings

. Idle servers consume up to 50% of power.

CPU Utilization	Current Consumption (abs.)	Current Consumption (rel.)
"idle" (0-5%)	0.8 amps	1
25%	1.1 amps	1.37
50%	1.5 amps	1.87
75%	1.8 amps	2.25
100%	1.9 amps	2.37

in software energy optimization research as it relates to cloud data centres and an increasingly connected world of digital ecosystems.

Berl et al. identify many research challenges related to cloud computing and energy consumption [20]. These challenges can be extended to the Internet of Things (IoT), CPS and digital ecosystems which integrate cloud computing resources. These open challenges include: energy aware scheduling in multiprocessor systems; reducing memory energy consumption based on scheduling; the integration of scheduling tasks with data placement; energy constraint scheduling; economic criteria; policies and energy as criteria for job dispatch and scheduling; power minimization in wired and wireless networks; energy considerations for the entire topology of integrated components servicing applications and end users (including services and infrastructure enablers); and hardware consolidations [20].

Berl et al. identify four main categories in the intersection of cloud computing and energy consumption:

- reducing the software and hardware related energy cost of single or federated data centres that execute 'cloud' applications;[1]
- improving load balancing and hence QoS and performance of single and federated data centres;
- reducing energy consumption due to communications;
- saving Green House Gases (GHG) and CO_2 emissions resulting from data centres and networks so as to offer computing power that is environment protecting/conserving.[2]

This dissertation addresses all four main categories identified by Berl et al. [20] as we address these issues from the point of view of an integrated digital ecosystem.

More specifically, this dissertation provides contributions to the open challenges of: energy constraint scheduling, policies and energy as criteria for job dispatch and

[1] A cloud application is a software application which is (partially, or entirely,) deployed on cloud resources (e.g., VMs, specialized hardware, servers, CPUs).

[2] GHG and CO_2 are difficult to measure and are outside the scope of this dissertation and, thus, will not be addressed directly.

scheduling, power minimization in wired networks and energy considerations for the entire topology of integrated components servicing applications and end users. While we only address a selected set of the open problems within each of these domains, this dissertation contributes approaches and solutions to each of these fields.

In particular, the contributions enable energy constraint scheduling through automatic profiling of hardware and software components (cf. Section 5.3.1). Furthermore, we demonstrate that results from our profiling work are transferable to scheduling and placement of virtual machines in order to effect overall energy optimization (cf. Section 6.3). Moreover, this dissertation contributes to reducing energy consumption of software-intensive systems by providing methods and tools for energy optimization and demonstrating their effectiveness in data centres in particular and digital ecosystems in general.

1.2 Our Approach

This section provides a high level overview of our approach including the context within which this research applies. The focus on data centres is in part due to the availability and access to servers within a data centre at the University of Victoria. It would be interesting to extend this research to digital ecosystems consisting of a large number of mobile devices, wireless communications, sensors, actuators as well as other IoT and CPS components connected to the internet. However, measuring power consumption in such diverse ecosystems is challenging. Nevertheless, we hope our findings in regard to data centres and the specific energy consumption of software may prove interesting for emerging digital ecosystems.

1.2.1 Overview of our Approach

The main, overarching open problem in green computing is how to reduce software and hardware related energy consumption and energy costs. This problem applies to single standalone and federated data centres alike. There is no panacea to the problems of energy optimization.

Measuring power in a real data centre can be accomplished in many different ways. The most common measurement techniques are instrumentation of hardware

components, software analytics tools, or directly from the power sources feeding into individual servers or entire server racks.

Power consumption can be measured and monitored along the entire multi-level network hierarchy of an enterprise data centre—from individual core and memory units, to servers and racks, as well as server rack power distribution units (PDU) and uninterruptible power supplies (UPS). Power instrumentation schemes can employ direct measurement techniques by using various hardware sensors or modelling techniques to estimate power consumption. For example, many recent processor architectures incorporate power reducing mechanisms whereby the CPU frequency is throttled back to reduce power consumption. Moreover, CPUs can be put into selected states depending upon the load; when the CPU is idling, the frequency at which the CPU operates is adjusted dynamically as more power is consumed at higher frequencies (DVFS). Tools, such as the Unix PowerTop utility and Joulemeter,[3] monitor how programs use the CPU mode features and estimate power consumption accordingly. Many servers have built-in power dashboards to monitor vital system functions and estimate power and bandwidth consumptions.

For this dissertation, when dealing with data centres, we concentrate on energy that is measured coming into an entire rack. This method of power consumption measurement has many benefits. One practical benefit is that the data centre in our use case already provides monitoring of energy consumption at the level of server rack PDUs. Classifying a processor's workload depending on the observed power profile visible at a PDU is possible with this monitoring setup since all servers on one power circuit are under our direct control. Demonstrating that the PDU is a viable point to detect and classify software processes based on their energy consumption is one of the contributions of this dissertation.

Having direct control over the servers within each rack allows us to reduce the amount of noise in the measured data and subsequently we can generate more meaningful measurements on which we base our analysis. Having full control of the servers means that we control the idle times and particular workloads on these machines even

[3]M. Goraczko, A. Kansal, J. Liu, F. Zhao: Joulemeter—Computational Energy Measurement and Optimization, Microsoft Research, http://research.microsoft.com/en-us/projects/joulemeter/, retrieved, Jan 2014.

to the extent of placing processes on specific cores of the CPUs. A more detailed description of our setup is provided in Sections 4.2.1 and 4.2.2.

Furthermore, using the PDU as the source of energy measurements ensures that all measurements are a true representation of all energy consumption of a server. Hardware sensors within individual components of the server are often unreliable due to the tolerances with which they are built. Relying on multiple sensors inevitably increases uncertainty by reducing the accuracy of the measurement due to compounding factors. Software instrumentation either relies on individual hardware sensors or attempts to establish a *best effort* estimate derivative energy consumption from other factors. Even combining both hardware sensors and software monitoring does not provide a *holistic* view of the energy consumption of a server because neither all components nor all servers are instrumented. Using the PDU as the source of power consumption ensures that the energy consumption of all components within a server are accounted for and that our approach is applicable to all server types, regardless of their individual hardware instrumentation. Consequently, using the PDU is the correct choice for our purposes.

1.2.2 Evolution to Digital Ecosystems

Our findings from software energy measuremnts have a direct impact on digital ecosystems and our findings may drive future research questions in these fields. Digital ecosystems in the form of the Internet of Things (IoT) and smart Cyber Physical Systems (CPS) are not solely composed of servers in data centres. Over the past decade the proliferation of mobile devices and wireless components has accelerated and reached unprecedented levels. At the same time, sensing, actuating, communications and computational capacities have been introduced into an increasing number of devices, including machines (e.g., cars), households (e.g., smart fridge), and infrastructure (e.g., smart cities and smart buildings)—creating the IoT [48]. Digital ecosystems emerge from the integration of these components. Different applications require connectivity and resources from different locations and providers, thus creating unique, application specific digital ecosystems, such as Apple Home or Google maps. The digital ecosystems created by social networks, such as Facebook, Twitter or LinkedIn, consume enormous amounts of power. With the emergence of cognitive

computing, digital ecosystems around conversational agents and intelligent personal assistants with natural language interfaces are being built to interact with people, brands or services. Popular examples of such agents include Apple's Siri,[4] Microsoft's Cortana,[5] Google Assistant,[6] Amazon's Alexa[7] and Facebook's Jarvis.[8]

The interplay among the devices and applications composing digital ecosystems is complex. Single instances of software applications are no longer confined to a single device or machine. Instead software commonly interacts with resources and services outside of its own hardware unit.[9] The scope of this interaction defines the applications' digital ecosystems. Smartphones interact with cloud resources, cloud resources include databases, specialized compute or storage services, or specialized hardware. Components of the IoT, which includes sensors and actuators in the physical world, interact with cloud resources for storage, analytics and decision making processes. In such a connected and distributed environment, combining processes of varying complexity with varying resource allocations (e.g., sensors, actuators, mobile devices or servers) produces different energy consumption levels for each component. This level of complexity provides an opportunity to investigate generic and specific optimization strategies. Thus, we aim to minimize energy consumption in software applications.

To the extent that we are able to investigate energy consumption in mobile devices, we rely on software sensors and instrumentation. This presents a whole new range of challenges due to the manufacturing of the devices, the tolerances within the hardware components, the variation between parameters, and the sheer number of different devices. We use built-in software sensors which are gaining in popularity and are commonly deployed on new models.

[4]https://www.apple.com/ca/ios/siri/
[5]https://www.microsoft.com/en-ca/windows/cortana
[6]https://assistant.google.com/
[7]https://developer.amazon.com/alexa
[8]https://www.facebook.com/notes/mark-zuckerberg/building-jarvis/10154361492931634/
[9]In fact, interaction of a device with the outside through networked hardware is often a requirement even for the simplest devices and tasks. Consider when a laptop or smart phone loses the ability to connect to a network permanently, the device is said to be "bricked" (i.e., for all intents and purposes, the device is as useful as a brick).

1.2.3 Dimensions of Energy Consumption Research

Mobile devices and servers share many features—from CPUs to memory and networking equipment. Fig. 1.1 highlights the different fields of interest as they pertain to energy consumption research in the IT server world. Our research for this dissertation falls into the illustrated categories. Existing research in these categories constitutes a foundation for our research. In Fig. 1.1, we can identify the four main areas of energy consumption research: *Energy type, Performance/Energy context, IT Resource type* and *Performance type.*

Energy type is the broadest of these four fields and spans research and analysis from machine and source code analysis at the lowest level of abstraction to general themes of physical resources and their energy consumption (e.g., energy consumption of disks, networking, and entire servers). Our research falls into this category by directly addressing the energy impact that software components have on disks, networking resources, and other hardware components, such as memory and CPUs. We also measure the entire server rack's power and isolate individual servers. As such this research falls into the Energy type category as defined by Lago et al. [77]. Because we measure and effect energy consumption and optimization of virtual machines (VMs), CPU and memory components our work also makes contributions to the three other categories, with an emphasis on performance, rather than the resource type and energy context categories.

1.3 Evolution of Energy Optimization

To appreciate the contributions of this dissertation fully and to understand how this research fits with the existing canon of energy consumption and optimization research, this section provides overviews of the related work, previous efforts and challenges.

The evolution of green computing research can be broken down into several main streams as they relate to performance, sustainably and energy consumption. These streams are: VM consolidation and heuristics, networking components, hardware component, software metering and instrumentation, source code analysis, energy consumption and markets, forecasting and simulation.

VM consolidation and heuristics focus on minimizing operational costs within data

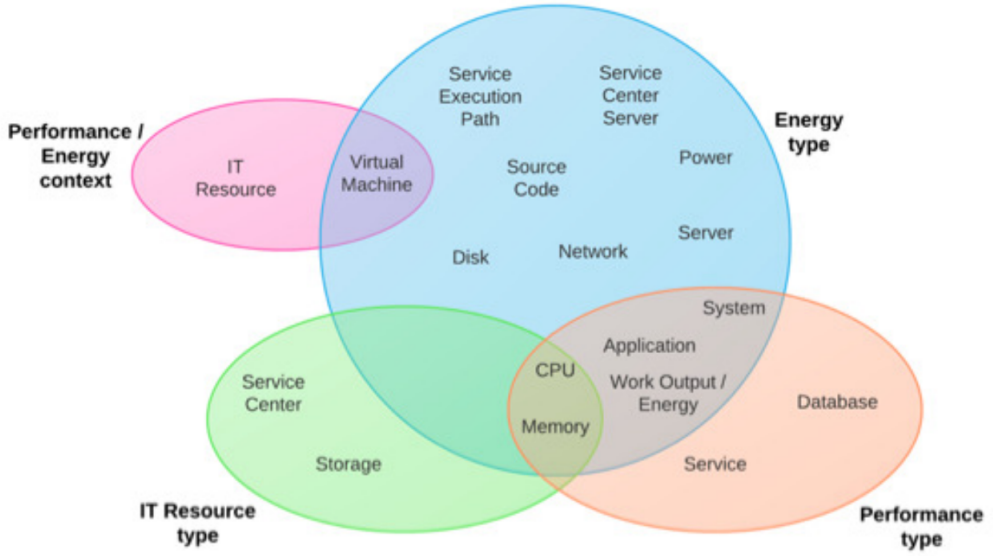

Figure 1.1: Fields and topics of energy consumption research—Courtesy of Lago et al. [77]

.

centres. This is achieved mainly through reducing the number of physical machines to which a VM can be deployed. The main avenues for energy savings arise due to optimization of resource utilization, networking components, and thermal states of physical servers and the related cooling [13]. Additionally, heuristics are involved to provide improved strategies for VM placement. Broadly speaking, heuristics for VM placement focus on reducing energy consumption, VM density per physical host, as well as workload optimization [86].

Heuristics are also a major research area for improving application algorithms on the software side, as well as Dynamic Frequency and Voltage Scaling (DFVS) on the hardware side. Assessment and evaluation of such heuristics often leads to the development of tool support in the form of simulation tools as well as measurement tools [14, 23, 29, 30].

Networking components, external to servers such as routers, and hardware component energy optimization is also a large field of research and commercial efforts. Improving the energy efficiency of networking components has been shown to have

beneficial impacts on large sections within data centres [91]. Previous work has shown that through optimization in networking, entire server blocks can be shut down in data centres [91]. This type of research also blends in with research efforts involving heuristics and physical components of data centres [135] as workload and network traffic must be taken into account for a larger scale view when optimizing individual components.

As highlighted in Section 1.2, source code analysis is an important research area within green computing. For example, assembly level instructions have been profiled for many years [93, 125, 126]. However, this type of low level analysis has recently fallen out of favour and has been surpassed in efficacy and popularity by higher function level analysis [32, 81, 117].

At a much higher level of abstraction, we can identify research efforts focusing on energy markets. This type of work spans from predicting and forecasting the future energy consumption of data centres to the large scale of energy transportation and source of energy generation [76, 109, 131]. While certainly interesting and relevant, this type of analysis is beyond the scope of this dissertation.

All these remaining challenges are addressed at various levels of abstraction. Yet, open problems remain at the levels of energy costs and consumption of wired and wireless networks and network components; scheduling, deployment, and migration algorithms and policies, sustainable programming language features; as well as sensing, storing and analyzing energy and performance metrics.

One aspect of forecasting energy demand, at a much finer grained level, has provided some interesting advances in machine learning techniques for energy consumption forecasting and software component placement within data centres [21, 70]. This type of work (i.e., machine learning and pattern recognition) is further elaborated in Section 3.1.

In summary, the field of green computing is composed of vast sub-fields with a rich history of research advances. Yet, many open problems and research challenges remain. Moreover, new challenges arise regularly as technological advances change

the components and composition of digital ecosystems and the software that runs on them.

1.4 Research Questions and Contributions

This section outlines the problem statements and research questions that this dissertation addresses. The research questions fall into two categories: *i*) *exploratory questions*; and *ii*) *design questions.*

1.4.1 Exploratory Questions

R1:

> What are the key resource usage factors (e.g., for CPU, memory or networks) that contribute to the energy consumption profile of software applications in digital ecosystems?

Contributions:

- Automated method for identifying a server's energy consumption per resource type and load.
- Energy consumption profiling tool for a data centre.

R2:

> What is the degree to which high-level adaptations at the application level, in terms of resource utilization and service deployment, can be utilized to optimize the overall energy consumption of the application's digital ecosystem?

Contributions:

- Application specific, automated and dynamic software modification to reduce energy consumption.
- Dynamic scheduling solution for software within data centres.
- Identification of potential energy savings specific to individual applications.

R3:

Does dynamic redirection of specific system calls and resource provisioning requests lead to energy optimization while not negatively impacting performance (i.e., either maintaining the same performance level or improving the performance level)?

Contributions:

- Automatic and dynamic changes to application level interactions with the underlying system to optimize energy consumption of software applications.

1.4.2 Design Questions

R4:

How can we obtain, store and manage relevant contextual information of software applications and their digital ecosystem?

Contributions:

- A contextual model to aggregate, store and analyze energy data within a new framework for data centres with the goal of improving energy consumption of software.

R5:

To what extent is dynamic run time modification of resource allocation manageable in a self-adaptive framework using models at run time (MART) and control theory approaches?

Contributions:

- Efficacy analysis of approaches presented in this dissertation.
- Identification of promising future avenues of research.

1.5 Dissertation Outline

Chapter 1 defines the problem and scope by situating this dissertation within the existing body of research and state of the art approaches. It also provides a high level overview of the motivating factors that inspired this research.

Chapter 2 provides a detailed explanation of the research questions, problem statement and approaches. Further, it introduces the reader to the contributions that this dissertation provides to the field.

Chapter 3 provides background information on existing and current state of the art research efforts in the field of green computing. This chapter focusses on works relevant to computer science and software engineering, and subsequently, narrowly defines green computing in order to establish a workable and appropriate scope for this research. The chapter also provides context by highlighting this dissertation's distinguishing features and contributions to the field.

Chapter 4 presents the research methodology including an initial hypothesis, discussion of the research questions, and the iterations required to evolve the hypotheses with the experiments and findings.

Chapter 5 provides insights into the experiments. Experiments focus on measurable energy consumption induced by software components and their particular use of hardware resources. The experiments also include dynamic models, as well as simulations, and scheduling strategies to effect energy optimization within cloud data centres.

Chapter 6 connects the experimental data with the models used in our setup. We provide insights into the collected data and how this data can be used by others.

Chapter 7 builds on the results of Chapters 5 and 6 to provide an evaluation of the impact of energy optimization of software components found in data centres. In evaluating the findings, we focus on two themes: (1) to what extent can energy consumption be measured and improved at the software level; and (2) what are the benefits in terms of energy consumption of such approaches.

Finally, Chapter 8 concludes the dissertation and illustrates potential avenues for future research emanating from this research.

Chapter 2

Research Statement and Approach

The chapter presents our research approach by relating the research questions to our hypotheses and experiments. For each research question, we introduce the criteria used to determine whether a research question has been answered successfully, either confirming or refuting the research question's hypothesis.

2.1 Problem Statement

Our research focusses on digital ecosystems as they exist in software applications through the interplay of resource utilization including network bandwidth, latency, storage and processing. Our primary motivation is to develop adaptive software infrastructure using self-adaptive systems and control technologies for the purpose of minimizing energy consumption without compromising system performance in digital ecosystems. In order to achieve this we need to be able to measure energy consumption of a wide range of software applications effectively on a wide range of devices. Once we are able to profile energy consumption of software we need to be able to connect software applications and their resource usage with energy efficient hardware on a case by case basis. Last, all this needs to be achieved dynamically in order to be scalable and applicable to future developments in digital ecosystems.

Application adaptation goals, such as maximizing throughput and minimizing energy consumption, require applications within digital ecosystems to be self-adaptive by following a model to control the adaptive process. Digital ecosystems are highly diverse and dependent on the context of the applications. Thus, a dynamic framework,

which adapts to the digital ecosystem, is needed to ensure adaptive behaviour not just of individual applications, but for all components working together in a digital ecosystem in order to achieve the selected application specific goals related to overall performance and energy consumption. For this research, we assume that digital ecosystems are scalable to many servers for the multitude of data center and mobile platforms. Scalability is a necessity in a world of rapidly evolving digital ecosystems including IoT and CPS.

One of the major limiting factors in sustainable or green computing experiments is invariably the availability of and access to real world digital ecosystems, including diverse hardware and software platforms as well as performance and energy consumption measures. This becomes relevant when applying one's research to any system of reasonable and realistic scale. Attaining full control of an entire data centre and running experimental research on its infrastructure and running applications is nearly impossible in practice for researchers in academia. The good news is that for our research we have full control of and access to approximately 7-9 servers in EDC2, the University of Victoria data centre. We have two server racks which host two different server types. This allows us to deploy our algorithms to reduce overall energy consumption on heterogeneous server hardware effectively. Using two different server types, we are able to extrapolate the effects of our algorithm and framework without the explicit need of simulations. Thus, our digital ecosystem is composed of 7-9 servers of types A[1] and B[2]), two Android smartphones[3] as well as three desktop computers situated within our research lab.[4]

In order to evaluate energy consumption as a metric used for energy and performance optimizations through scheduling decisions, we need access to high level monitoring tools. For our data centre servers, which are representative of modern data centres and clouds, this is possible because we have access to energy consumption metrics and tools used by the data centre's staff. We also have the ability to measure energy consumption in desktops and smartphones through built-in instru-

[1]Each of our servers is equipped with two Intel® Xeon® Processor X5650 running at 2.67GHz. Both processor have six cores with x2 hyperthreading and, thus, contributing a total of 24 logical processors. http://ark.intel.com/products/47922/Intel-Xeon-Processor-X5650-%2812M-Cache-2_66-GHz-6_40-GTs-Intel-QPI%29

[2]SAVI server

[3]Personal smartphones, Nexus 5, http://www.gsmarena.com/lg_nexus_5-5705.php

[4]Intel Duo Core

mentation, where applicable, or more commonly through external instrumentation.

On the software side, we must analyze what is under our control. We not only have control over the code at the programmer level, but because of WYSINWYX[5] [8], we can take advantage of available low level control available in the form of compiler switches and system call redirection. We have control over the exact machine instructions that are generated from any given piece of software (i.e., through the compiler or, at a more practical level, by capturing and redirecting system calls).[6]

Thus, to measure energy consumption, to effect change at scheduling and deployment strategies in data centres, and to alter software components dynamically at run-time, a complex body of contextual information needs to be managed. At this level of abstraction, the framework or controlling an application must be able to have access to a knowledge base of the energy consumption and, if chosen by the user, produce energy efficient code in agreement with the high-level goals. Energy efficiency ought to be optional, just like speed or size optimizations are optional in modern compilers. For this purpose, we need to develop a framework which builds on existing dynamic run-time models.

Energy consumption models have been used before, with limited success, at the level of software development [55]. These models were applied exclusively at system design time and, thus, provided a static solution which no longer satisfies the needs of modern agile systems [10, 73]. In the past, people have looked at energy consumption of individual machine instructions [114, 117, 126]. These models were often generated in advance on generic hardware and, through assumptions of general applicability and simulations, made available to the programmer in the form of estimated energy efficiency of the code being written. However, these are still in the early stages and often are not very good as they assume that the target hardware is homogenuous to (or close to) the one used for experiments. Software energy efficiency has been investigated to provide feedback to programmers, based on models, regarding the estimated energy consumption of their programs [138, 139]. Their application to large scale systems or, even small, modern applications which rely on a diverse set of dis-

[5]What You See Is Not What You eXecute

[6]We use redirection of system calls as a practical choice. Not only is this more feasible to automate, but it is also in line with current research discouraging machine level manipulation for energy consumption optimization.

tributed and heterogeneous resources, is still in its infancy.

We differ from this approach by addressing software energy optimization on real deployed software applications at runtime which impacts, both, the software itself, though system call redirections, as well as underlying hardware, through scheduling decisions. What saves energy: using efficient machine instructions that do not use a lot of energy, or ignoring the energy consumption of individual instructions and simply aiming at the highest performance (based on throughput, latency, or other measures) to optimize applications? Due to the increasing proliferation and rapid development in the IoT, we need to develop an understanding of the energy consumption optimization of entire digital ecosystems.

The subsequent sections discuss specific research questions addressing aspects of the above identified research challenges of dynamic digital ecosystem management and resource scheduling with the goal to minimize the overall energy consumption: *i*) the definition and specifications of an interconnected digital ecosystem on an application basis as well as the abilities and processes to alter the resource utilization patterns dynamically; *ii*) the run-time adaptations to resource scheduling and utilization with the goal of maintaining high-level performance and energy consumption goals; and *iii*) the ability to use low level modifications on individual components of an application's digital ecosystem to affect a component's performance and energy dynamically at run-time.

2.2 Research Questions

This section explores our research questions which fall into two categories as mentioned earlier: *i*) *exploratory questions*; and *ii*) *design questions* as outlined below.

2.2.1 Exploratory Questions

While our research focuses on energy consumption of software in data centers, we derived the exploratory questions by studying the state of the art research in the field of energy consumption. In this dissertation we investigate the connections between performance and energy optimizations for software applications running on servers

in data centres. We conducted the state of the art research with mobile applications and digital ecosystems, such as IoT and CPS, in mind.

> **R1**:
> What are the key resource usage factors of software applications that contribute to the energy consumption profile of interconnected IoT and CPS applications. What are the key resource usage factors (e.g., for CPU, memory or networks) that contribute to the energy consumption profile of software applications in digital ecosystems?

From our systematic literature review and our preliminary research, we conclude that the degree to which certain software aspects contribute to an application's energy consumption is not entirely clear in a dynamic environment. While it is generally known that certain resource utilizations by an application will generate a different energy consumption profile than another resource usage pattern, many questions remain open. There is an insufficient understanding as to the interplay between resource usage and its generalizability to energy consumption and the resource utilization patterns across multiple devices in a digital ecosystem. Moreover, there is no accepted research base line regarding which modifications of a software application's behaviour would lead to improved energy consumption metrics, such as performance metrics and energy/power consumption measurements.

To explore this question effectively, we need to develop a method by which we can determine the impact of a software's resource usage on energy consumption. Developing a tool and framework for profiling different server hardware and software components is an appropriate method to answer this question. Analytically speaking, we submit that to answer this question affirmatively we need to meet the following two criteria: *i*) we are able to determine the dominant resource (i.e., CPU, memory, disk, or network) used by a software application solely by analyzing its energy consumption profiles; *ii*) we can make a reasonably accurate forecast of this software's energy consumption profile on a different hardware platform. As a result, we can then pinpoint the exact servers in a data centre where software applications should run to minimize the overall energy consumption while satisfying performance requirements.

The current lack of fundamental research to answer this question effectively leads us to the next exploratory research question.

R2:
What is the degree to which high-level adaptations at the application level, in terms of resource utilization and service deployment, can be utilized to optimize the overall energy consumption of the application's digital ecosystem?

This question aims to determine the extent of possible energy consumption optimizations that are possible when high level goals of the digital ecosystem are managed by a control system at a higher level of abstraction. For example, how beneficial is it to postpone certain types of tasks in their scheduling to take advantage of anticipated changes in resource availability? Can certain computation, data storage or acquisition be offloaded to other parts of the digital ecosystem which are more energy efficient at these tasks in order to achieve net energy savings while maintaining overall performance goals?

An answer to this research question must give us the ability to make decisions on optimization strategies at runtime. Given a particular software applications and its characteristic resource usgae patterns, how much energy is saved by moving it to different hardware platforms and are the energy savings measurable and sufficient to warrant necessary monitoring and analytics?

Closely related is this kind of scheduling is the adaptation of certain application aspects at runtime. Runtime modifications of software are also attractive avenues of research as there is an impressive body of knowledge in this field. The corresponding research question is as follows:

R3:
Does dynamic redirection of specific system calls and resource provisioning requests lead to energy optimization while not negatively impacting performance (i.e., either maintaining the same performance level or improving the performance level)?

In other words, can we, within our experimental digital ecosystem at UVic, optimize software applications for energy consumption in such a way as to not reduce performance. This avenue of investigation focuses on changes to the software without considering changes to scheduling or deployment decisions. Considering extreme solutions, the impact and goals are clear: not running software at all yields energy

savings of +100%, but a performance change of –100% does not produce any tangible results. Thus, the goal is to find energy savings greater than 0% and, thus, reducing overall energy consumption, while determining and acceptable reduction in performance while still providing the expected results. In an ideal world, we achieve performance improvements and accrue energy savings. In some of our experiments, we managed to demonstrate this ideal world scenario of an increase in performance and a decrease in energy consumption simultaneously.

2.2.2 Design Questions

Design questions are commonly used to articulate non-empirical research in software engineering [45]. For this dissertation the design questions focus on an investigation to manage and improve self-adaptive models and feedback systems for the purpose of affecting the energy consumption of digital ecosystems.

The follwoing research questions frame how our findings can be applied to our ultimate goal of developing smart, self-adaptive, green computing techniques in data centres, on individual devices (e.g., mobile, desktop, laptop or server), and in digital ecosystems.

As such we aim to answer the following design research questions:

> **R4**:
> How can we obtain, store and manage relevant contextual information of software applications and their digital ecosystem?

This research question seeks to address the difficult problem of obtaining relevant contextual data at runtime. The problem arises because in a digital ecosystem various actors contribute to the complexity of the problem (e.g., servers, data centre monitoring infrastructure, software instrumentation, hardware instrumentation of desktops and smartphones). We address the problem of not only obtaining, but also storing the relevant information in a manner that lends itself to a management framework or model. As such we investigate not only what type of information is needed, but, due to the dynamic nature of the digital ecosystem, also at what frequency contextual information needs to be obtained.

> **R5**:
> To what extent is dynamic runtime modification of resource allocation

manageable in a self-adaptive framework using models at runtime (MART) and control theory approaches?

Managing digital ecosystems is a complex task. The number of parameters is large and the possible configuration combinations grow exponentially. Subsequently, a metric by which we can establish the successful answer to this research question is by judging the resulting system on two criteria: *i*) *system stability*—a state where changes to the system are unnecessary to achieve the system's goal; and *ii*) *optimization objectives*—reduction in overall energy consumption and no significant performance penalty using the runtime model to effect changes in the digital ecosystem.

2.3 Long Term Research Goal

The long term goal of this research is to investigate innovative methods to modify program execution and resource scheduling of applications dynamically within digital ecosystems in order to maintain or increase performance throughput while, at the same time, decreasing energy consumption.

Our short term goals for achieving high level goals related to energy consumption are: (1) to identify which types of applications benefit the most from dynamic resource scheduling within digital ecosystems; (2) to develop a framework to provide energy consumption data dynamically; and (3) provide runtime decision making support for applications to alter their resource usage dynamically. At the same time, the hope is that this dissertation can also serve as a *handbook* for software developers by providing useful insights, tips and practices which can increase an applications performance while reducing its energy footprint.

Chapter 3

Research Background

This chapter overviews the main research foundations pertinent to the research presented in this dissertation, including research in green computing, self-adaptive and self-managing systems as well as background on control theory, feedback loops and signal processing needed for our research.

We illustrate high level concepts of green computing and focus on the most relevant works as well as on the most recent and promising related research. Moreover, this chapter provides insights, critiques and unresolved issues in green computing within software engineering and computer science.

Research in green computing encompasses four main dimensions: hardware platforms, software applications, instrumentation and sensors, as well as models and simulations.

Within these dimensions, the goal is to reduce overall power consumption by developing methods, techniques, frameworks and tools and applying them to data centres, individual devices (i.e., mobile devices, tablet, desktop, laptop and server) and software applications' digital ecosystems. The ultimate goal is to develop smart, adaptive, dynamic green computing techniques, including adaptive job scheduling, resource provisioning, data migration and task allocation.

3.1 Green Computing

This section is contrasted with the relevant background work describing context and self adaptive systems which situates the dissertation in the larger applicable context and future research avenues. For the purpose of this dissertation, green computing covers data centre design, algorithmic efficiency, resource allocation, virtualization, mobile and wireless computing, and energy consumption. Within those fields we pay particular attention to hardware components found in data centres, desktop computers and laptops. Resources, such as networking capabilities and capacities, storage, and processing power, are at the forefront of this work as are the software applications utilizing these resources.

3.1.1 VM Consolidation and Heuristics

Virtualization is the key technological principle that has facilitated the growing popularity of cloud, edge and fog computing in recent years. Virtualization partitions the resources of a single server into multiple separated execution environments where each environment is isolated from others; therefore, multiple operating systems can run on the same hardware [11, 33, 92, 111].

Virtual machine (VM) consolidation, in theory and practice, is closely related to server consolidation. Server consolidation is commonly defined as "an approach to the efficient usage of computer server resources in order to reduce the total number of servers ... that an organization is using" [112]. VM consolidation, is crucial to achieve server consolidation. VM consolidation refers to concentrating VMs on a reduced number of servers. However, VM consolidation is just one consideration (i.e., to move as many VMs as possible onto a single server). There are many other crucial considerations to the operation of data centres and service providers, including availability, costs of memory units within each server (this affects VM density and availability), redundancy, risk management, scalability (scaling out vs. scaling up), SLO/SLA, and performance [78]. Each of these considerations, in turn, has many nuances that fall outside of the scope of this dissertation. However, it is important for the reader to understand that VM consolidation is more complex than simply packing VMs tightly onto servers. Moreover, Laverick points out that VM consolidation ratios are, in fact, not key considerations for data centres [78]. Similarly, VM migration methods also present a multitude of approaches and research avenues [89, 141]. In

their 2014 paper, Medina and Garcia provide a comprehensive overview of the most common techniques used [92].

We now discuss some of the more recent, prominent and important works addressing VM consolidation. Bobroff et al. [22] developed a dynamic VM consolidation algorithm with the goal to reduce both energy costs and SLA violations. Often, energy costs and SLA violations are in direct competition: "Power management directly threatens the independence and performance isolation properties of virtualization technologies" [99]. In order to reduce performance and SLA violations servers have fewer VMs scheduled than they are physically able to accommodate. However, this then requires more physical machines to host the VMs which in turn increases energy consumption and associated costs. Bobroff et al. developed an algorithm to simultaneously tackle both issues by addressing over-utilization and under-utilization.

Singh, like Bobroff, also an IBM engineer, continued to advance dynamic solutions to reducing SLA violations and improving costs in data centres. Singh et al.'s algorithm is inspired by multi-dimensional knapsack problems and focuses specifically on load balancing resource requirements across multiple resource levels (i.e., storage, network and CPU) to avoid overloaded "hotspots" in the data centre [116]. In this algorithm, each resource component forms one dimension of the vectors used to compute solutions to this type of multi-dimensional knapsack problem. Singh's system is reactive rather than predictive, which sets it apart from Bobroff's work.

Beloglazov et al. [13] employ resource consolidation to minimize operational costs. Energy savings are achieved by consolidating virtual machines based on resource utilization, networking topologies and the thermal state of compute nodes. Specifically, a simulation of heuristics is presented for resource re-allocation which resulted in energy savings. They discuss energy saving practices, such as improving application algorithms, energy efficient hardware, dynamic frequency and voltage scaling (DFVS) and virtualization. Several policies are proposed to mange VM placement. Challenges in policy implementation include the trade off between energy savings and quality of service requirements and minimization of global power consumption. Simulation results produced savings as high as 83%. Beloglazov et al. [14,23,29,30] employ resource provisioning algorithms in conjunction with tools like CloudSim for evaluation pur-

poses.

In the absence of having access to a real data centre, modelling and simulation is a popular method to evaluate resource scheduling algorithms in combination with energy performance. CloudSim [29, 30] is, arguably, the most popular of such tools in academic research. The approach of using simulation to evaluate energy and cost saving algorithms in cloud environments has been taken on by several researchers, such as Buyya et al. who find great value in the simulation based approach for practical reasons [28, 74, 134].

Liang et al. [86] use a heuristic approach to solve the virtual machine management system within their own data centre architecture. They implemented their test system on real hardware to achieve realistic results. Their experiment resulted in 27% savings in energy consumption. Minimization of energy costs using heuristics yields varying results and often depends on the particular implementation, assumptions and underlying architecture [27, 144].

3.1.2 Networking Components

At a slightly higher level of abstraction, Mazzucco and Mitrani [91] investigate the energy saving impact of distributing workloads in such a way as to facilitate shutting down blocks consisting of multiple servers. Their results showed that such a strategy is highly feasible under specific conditions and depends on the type of servers' network traffic. They concluded that, as long as the network traffic is not too bursty and the load is not too large, it is possible to benefit from such a strategy.

Similarly, taking the approach of fine grained optimizations and power savings, Widjaja et al. [135] show that depending on the type of network traffic, small sized switches can be deployed in order to save energy costs. The authors also mention the importance of having the ability to turn network switches off as a benefit to energy savings depending on the type of network traffic. However, their simulations explicitly state that quality of service (QoS) concerns are not considered and, thus, can skew the results. In experiments like these, measuring energy consumption relies on sensor data on individual hardware components. Savga et al. [130] provide an overview of

the state of the art of server hardware monitoring. In isolated experiments, data centre network components have been shown to benefit from energy consumption specific optimizations. In other words, unused switches would ideally consume no power [1, 12, 56, 133]. Moreover, Jiang et al. [65] present mathematical models for turning off excess capacities in networking equipment, but emphasize that heuristics are needed to improve the performance of their approach. Turning off not only network switches but also servers, or entire server islands in data centres is a result of resource consolidation and has far reaching impact beyond the energy consumption of servers as this approach also affects cooling measures [51, 79].

3.1.3 Software Metering and Instrumentation

Kansal et al. [67] provide a metering method named Joulemeter, which provides VMs with a low overhead power metering system similar in functionality to hardware monitoring for physical machines. They experimentally determine the accuracy of the meter by comparing the software reported energy consumption to the physical machine's energy consumption. The authors also address renewable energy sources and variable data centre pricing that Joulemeter can take advantage of.

Pathak and Aggarwal demonstrate that effective instrumentation of software to track energy consumption can take the form of monitoring the number of system calls to the operating system [2, 39, 106] . A system call is considered an action of the operating system to effect a change on the hardware leading to a change in energy consumption. Thus, system call changes in applications as new releases emerge are used to predict an application's energy consumption. Part of this dissertation is inspired by the work of Pathak and Aggarwal. As such, we profile energy consumption of system calls across multiple machines with the goal to modify specific system calls to provide improved energy consumption for the underlying software application.

Hähnel et al. [52] investigate a finer grained power measurement technique by taking advantage of kernel level sensor capability. Specifically, they use running average power limit (RAPL) sensors available for Intel processors. Their results proved promising in providing finer grained power measurement support for individual instrumented applications.

3.1.4 Instruction Level Analysis

Following this trend of function level monitoring, finer grained approaches have been proposed by Li and Tiwari [82, 127]. Energy consumption for individual assembly language instructions was once considered a promising avenue of understanding the overall energy consumption of software. Energy consumption for assembly language instructions has been profiled extensively for many years [93, 125, 126]. However, the rewards from such approaches are minimal as most instructions have common overhead and do not vary significantly on an individual level. Thus, there is no longer a need to investigate individual assembly level instructions [32, 114, 117]. Due to this consensus among researchers, we restrict ourselves to the level of system calls in our research.

3.1.5 Higher Level Instruction Analysis

At a more abstract level, Bunse's work [26] shows that energy consumption can be one metric by which an algorithm is chosen. Through the use of trend functions, he predicts energy consumption of several algorithm implementations. This approach conceivably lets users select a final algorithm based on chosen metrics (e.g., execution time or energy consumption). Similarly, Li et al. [81] show the impact that common programming practices have on energy consumption in the mobile market (e.g., smartphones, tablets and watches), but also highlight how slight variations to, or adaptation of, best practices can improve energy consumption.

Related to this, Corral et al. [40] focus on mobile devices and highlight that in this context, energy consumption can be approximated based on the execution time of individual applications, or code segments. However, we will see later that this is primarily due to the fact that applications running on mobile devices are usually restricted to single cores (cf. Chapter 5). Research on mobile devices' energy consumption is of great interest due to the economic implications. Applications which require disproportionate energy consumption often fare far worse in the market place, thus resulting in lower revenues for the developer. As such, energy consumption in mobile devices for individual applications as well as their variation over multiple version and release cycles is of interest to application developers as well as academic researcher [58, 59, 143].

Energy consumption in mobile devices is also of interest to software engineering researchers interested in best practices and coding standards as well as their impact on energy consumption [81]. It is highlighted that best practices, as they are understood in the software engineering community, do not always translate into energy savings of the software to which they are applied. In fact, many so called "best practices" have adverse effects on the energy consumption of applications running on mobile devices. Mobile device energy research has been reviewed comprehensively by Wang et al. [132].

3.1.6 Energy Consumption and Markets

At the large scale, Wang et al. [131] and Chiu et al. [38] discuss models incorporating energy grid loads and data centre demands. Strategies to migrate workloads dynamically based on utility calculations are proposed and simulated through the use of models. Both research works, however also point out the complexity of reaching an optimal state as migrating workloads based on power grid considerations can have adverse effects on the overall system. Minimizing energy costs in a global market can also profitably employ different energy sources and markets from which to obtain electricity. This approach is also prominent as there is a dynamic component to the availability and price of different energy sources over time [46, 109].

Chase et al. [34] use an economics based approach to model supply and demand using a greedy algorithm. They also describe an adaptive resource provisioning architecture useful for consolidating services. Their experiments indicated a 29% reduction in power consumption using their solutions.

Krintz et al. [76] model how applications affect power consumption. The idea is that the application has some control over the power they use. This research targets mobile devices using application level observations and power consumption prediction. Their approach avoids complex modelling by using the system as a black box as a whole. They use several benchmarks and measure the power dissipation along several key metrics.

3.1.7 Forecasting Energy Consumption

Tesauro et al. [124] have used machine learning approaches to maximize energy savings and performance. By training on data from a simple random-walk, they achieved high-quality management polices that outperformed the best available hand-crafted policy by over 10%. Another strategy employed by Berral for the same purpose, used machine learning for a back-filling scheduling approach [21]. Their work focuses on the continued satisfaction of SLA agreements with energy-aware scheduling. The tradeoff between energy consumption and performance is addressed by Kephart [70]. Two controllers are used, in a small scale setup, to optimize the tradeoff between energy savings and overall system performance using predefined utility functions. Using machine learning algorithms improved the coordinating without time-consuming drawbacks to optimize the system.

Duy [44] uses a green scheduling algorithm featuring an artificial neural network predictor for optimizing server power in the cloud to forecast energy demands. He uses neural networks (NNs) to predict load requirements to decide whether or not to shut down servers in the data centre. In other research, genetic algorithms are employed to determine load, energy savings and job scheduling in data centres [108]. Many machine learning approaches exist and compete with one another for best results as it is still not clear which approach is best suited for this domain.

3.1.8 Simulation versus Execution Environment

Another point in favour of our approach is the work highlighted by Teodorescu and Torrellas, who point out that even homogenous hardware is in fact heterogeneous due to production processes [123]. In their work the authors point out that, for example, cores vary in the amount of consumed power and supported frequencies despite being identical [123]. Thus, dynamic analysis tools for actual hardware are far superior in accuracy than simulations [46]. An additional point in favour of this dissertation's approach is the increasing trend in data centres and other computing paradigms to favour energy consumption savings over performance [146].

3.2 Software Engineering for Self-Adaptive and Self-Managing Systems

This section presents an overview of software engineering research and background for self-adaptive and self-managing systems (cf. SEAMS software engineering research community). The chapter also introduces machine learning research related to green computing and self-adaptive systems. Dynamic and, in the future, self-adaptive approaches have become a necessity in today's world of data centres for operational, planning and energy related aspects. As MacVittie suggests, the "static, inflexible data centre models of the past inhibit growth, promote inefficiency, and are fraught with operational risk" [88]. Dynamic, adaptive and self-adaptive problem solving approaches provide exciting opportunities and will unlock new potentials in data centres in the future. It is important to consider these works to gain an understanding of the larger scale applicability and potential future directions (IoT, CPS) of our findings which are derived from a currently existing data center.

3.2.1 Self-Adaptive and Self-Managing Systems

Self-adaptive systems (SAS) are systems that have the ability to modify their own behaviour at run-time. Often the degree of adaptation is predetermined (e.g., switch among a few modes). However, normally self-adaptive systems are not bound in their runtime adaptation and can thus deal with a certain amount of uncertainty in their environment. In this case, the self-adaptive system can alter its behaviour according to specific contextual information which can change over time in order to achieve the system's goals. [128] Adaptations are separated generally into high-level or low level goals, as well as short-term and long-term adaptations [102]. Furthermore, adaptations at runtime often occur as a system "drifts away from its design-time models [while] run-time models are kept in sync with the underlying system" [55]. Models at runtime are further subdivided into research pertaining to architectures, objectives, techniques and types of models; Szvetits and Zdun further categorize areas within these categories into a total of 27 specialized areas of research [122].

In 2001 IBM began to promote the vision of autonomic computing [61]. The seminal paper by Kephart and Chess pioneered the notion of self-managing systems, including the famous MAPE-K loop [71]. They featured four self-* (read *self-star*)

properties: *self-configuring, self-optimizing, self-healing* and *self-protecting*. These properties are key characteristics which set autonomic systems apart from other systems. More recently, more self-* properties have emerged and are gaining in relevance in different autonomic frameworks [25, 42] and IoT platforms [73, 104].

Kephart and Chess [63, 71] introduced the key building block for self-managing systems as depicted in Figure 3.1—an autonomic element (AE) consisting of two components: an *autonomic manager* and a *managed system* which form a feedback loop.

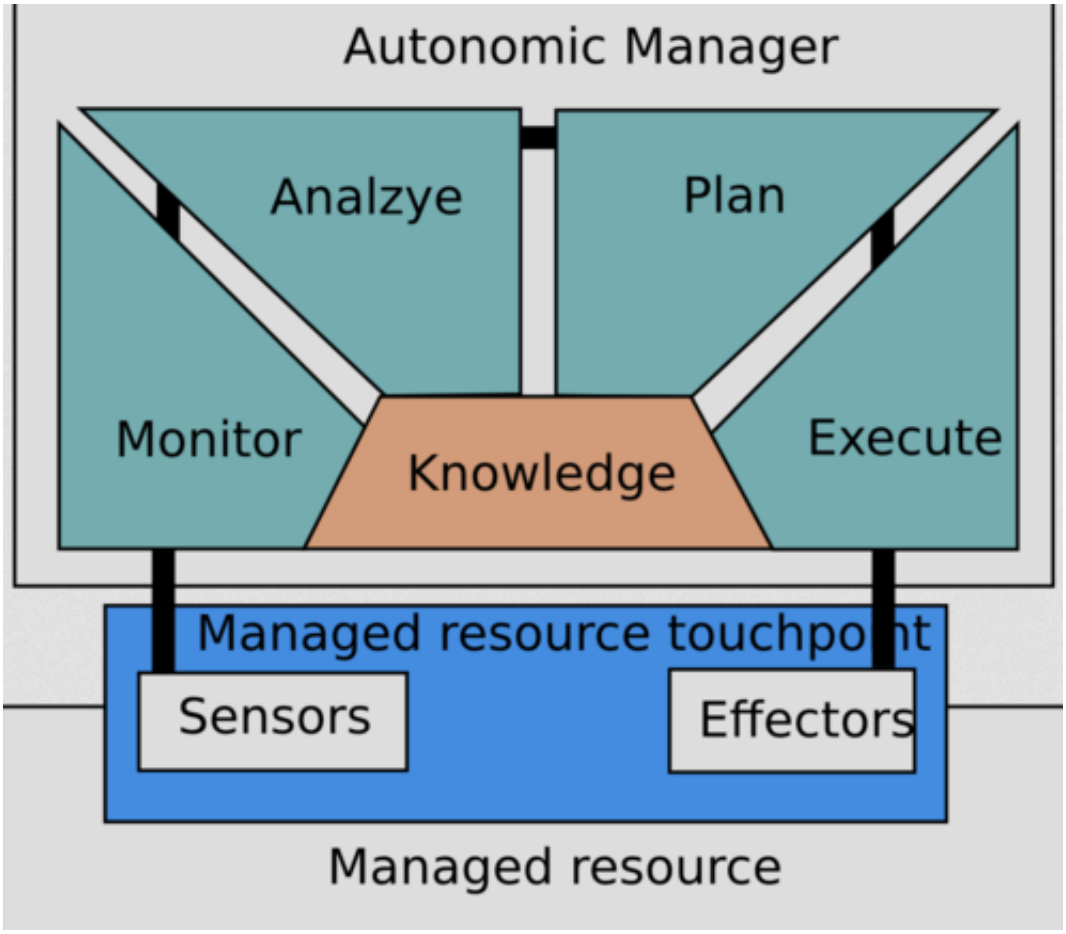

Figure 3.1: Autonomic element with managed system and autonomic manager manager featuring the MAPE-K loop with its standard components [63, 71]

The autonomic manager and managed system correspond to the controller and the process, respectively, in the control loop from traditional control theory. To separate concerns, they structured the autonomic manager into four distinct phases and

a common *knowledge base*: *monitor, analyzer, planner* and *executer*. This famous design, depicted in Figure 3.1, is now referred to as *MAPE-K* loop. This design has hugely influenced academic research directions as well as industrial realizations of self-adaptive and self-managing systems.Kephart and Chess [41, 62, 96–98, 128].

The MAPE-K loop contains four distinct, interconnected, phases. a) Monitor; b) Analyzer; c) Planner;and d) Executer.

The *Monitor* phase continuously collects sensor data about the environment within which the application is currently running. The filtered sensor data are then stored in the *Knowledge Base* for further processing and long term storage. The *Analyzer* module performs analysis tasks on the collected data and tries to recognize known patterns and trends in order to determine whether the high level goals are currently met, or will eventually be met in the future. The *Planner* module is called upon when the *Analyzer* has determined that goals are no longer being met. It is the *Planner* module's job to determine the best course of action and behaviour modifications in order to achieve the system's goals in the short term or long term. The *Executer* module is tasked to provide the required instructions for behavioural modification of the system to the low level components through effectors or actuators. At the level of hardware, interfacing with the real world, are *sensors* and *effectors*. Sensors provide information to the *Monitor* module, while effectors carry out instructions from the *Executer* phase in order to alter the system and its behaviour in accordance with its high level goals.

The autonomic element (AE) is the conceptual architecture for an individual feedback loop in self-adaptive and self-managing systems. Commonly, AEs are composed into three-level hierarchical architectures. At the lowest level, autonomic managers (AMs) control individual resources, such as disks and CPUs. Each AM in the middle level aims to achieve one of the self-* properties (e.g., self-configuring or self-healing). The top level orchestrates the middle managers and determines priorities on tradeoffs among the middle level managers when aming to achieve overall system goals.

This three-level architecture is further refined in Figure 3.2 which depicts IBM's original reference architecture which is referred to as *autonomic computing reference architecture (ACRA)* [63]. The ACRA hierarchy is composed of five basic layers.

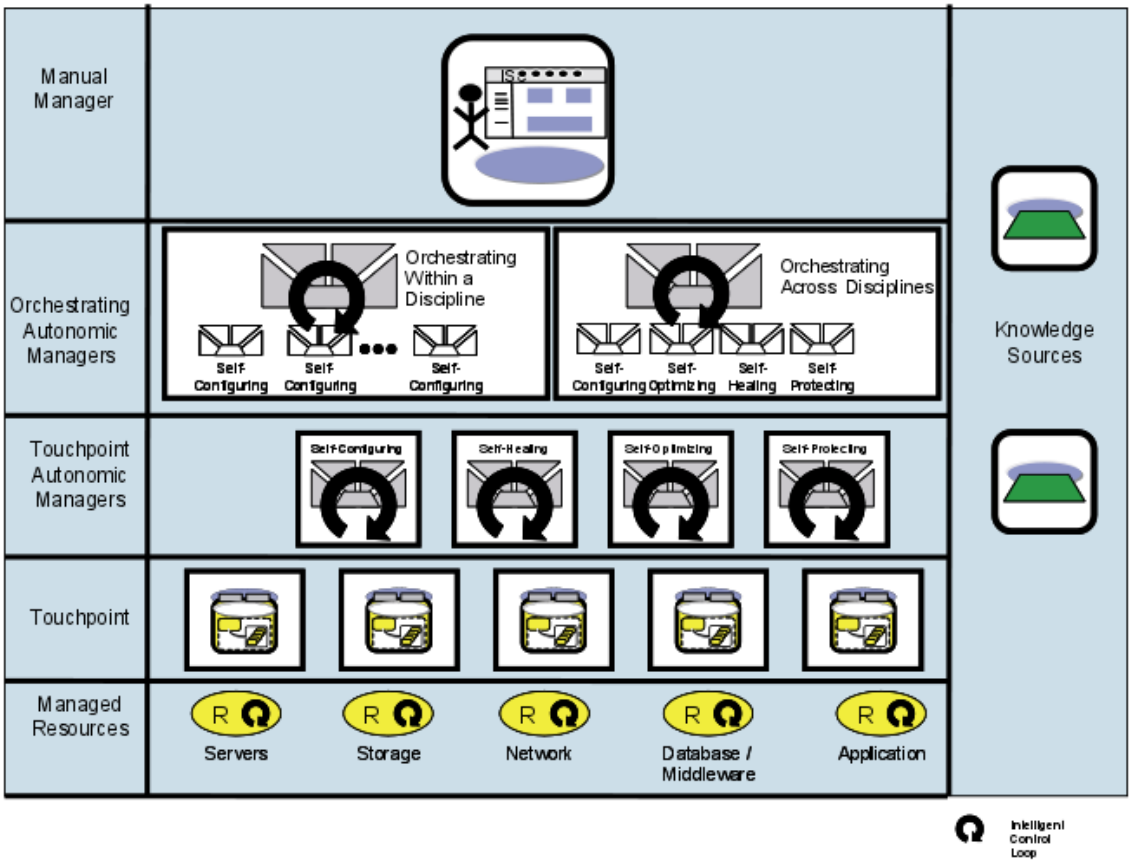

Figure 3.2: Autonomic computing reference architecture (ACRA) [63]

Conceptually at the bottom of the hierarchies one finds the *managed resources* (e.g., a light-bulb, a server, a network switch, a sprinkler system, or any other resource in an autonomic system). Interfacing with the managed resources are *touch points,* which allow for information and instruction flow between the managed resource and the AM. Touch points are controlled and receive instructions from the *touch point AMs.* Information is analyzed and plans for decision making are formed in accordance with relatively low level goals relevant to successfully managing the low-level resources of the system.

Above the touch point AMs are the *orchestrating AMs.* This level is composed of multiple interconnected AMs which oversee a multitude of touch point AMs. The *Monitor* modules of this level retrieve sensor data from a wide range of systems. High level goals are monitored and effected at this layer in the ACRA hierarchy. These high-level goals are supplied by the top layer—the *manual manager*, which is often

controlled and operated by a human.

Monitoring, controlling and managing context information at runtime is a crucial function of self-adaptive and self-managing systems. The availability of ACRA has allowed researchers and practitioners to expand the scale of autonomous systems by increasing automation and by continually reducing the human involvement towards high-level goal setting. An abundance of data and new data through a continuously increasing number of sensors (especially in the IoT domain) makes autonomous, large scale, systems a relevant research area. Lessons from such large scale autonomous systems apply to a variety of topics ranging from city planning to networks and data centres. Runtime analysis of context information has received great interest from the self-adaptive community and has been identified as an important research area in software engineering for adaptive systems [36, 103, 105, 120].

3.2.2 Data Science—Machine Learning and Statistical Analysis

Machine learning approaches, including Neural Networks (NN), Support Vector Machines (SVM), wavelets and Statistical Analysis, have found increasing application in load balancing as well as resource and job scheduling in data centres. Machine learning, in computer science, is a subfield of pattern recognition and can be defined as follows: "Machine learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions" [95]. Machine learning automates analytical model build and uses algorithms that iteratively learn from data. The goal is to find hiden insights without explicitly programming where to look in the big sea of data. The textbook on pattern recognition by Duda et al. [43] is a great introduction to traditional classifier training. Neural Networks (NNs) and Support Vector Machines (SVMs) are just a few of the many machine learning topics. Machine learning has myriad applications in forecasting, data analysis, bioinformatics and image analysis to name but a few [90].

Recently, Aksanli investigated scheduling tasks in data centres by dynamically forecasting the availability of green energy [3,4]. Jobs are scheduled or removed solely based on the, real and predicted, availability of green energy (i.e., energy produced from renewable sources). Aksanli does not seek to improve the energy efficiency of the

data centre's workload, but rather promotes the notion of scheduling data centre jobs when energy from green (renewable) sources is available regardless of the efficiency of the underlying workloads.

Duy [44] uses a green scheduling algorithm featuring an artificial neural network predictor for optimizing server power in the cloud to forecast energy demands. He uses NNs to predict load requirements in order to decide whether or not to shut down servers in the data centre. Similarly, Sharma uses genetic algorithms (GAs) and NNs in an effort to combine dynamic voltage and frequency scaling (DVFS) and GAs to minimize SLA violations [115]. Sharma also seeks to minimize energy consumption and provides a high-level overview of some energy saving methods' benefits and drawbacks. Energy consumption and data centres lend themselves to a variety of different NN approaches and variations. Foo employs evolutionary NNs due to their speed and accuracy in their studied environment [47]. Wu et al. illustrate the many different NNs currently being studied and deployed to place VMs in data centres [140]. They demonstrate that there are many promising avenues, which all which can yield successful results, depending on the exact underlying problem being studied.

While NNs are only recently being used in data centres and research related to scheduling and energy efficiency, they have been used extensively in other domains. More commonly, NNs are being used in research involving energy consumption outside the data centre realm. Many lessons and findings from those works can be gleaned to help conduct analytics to optimize data centre energy consumption.

Neto et al. highlight the value of NNs when forecasting and profiling energy consumption of entire buildings while comparing NNs and simulation based physical models [37, 100, 107]. Similarly, Amin-Naseri et al. show that NNs can be used successfully to forecast daily peak energy loads in large systems [5, 60, 121]. They argue that NNs perform well because traditional models are often overwhelmed by the complexity of external variables, such as weather or climate data. Their method allows the clustering of peak loads specific to the day of the entity that is profiled. Related to this is the work by Li et al. in their classification of energy consumption in energy-efficient buildings [83]. They focus on forecasting and detecting abnormal energy consumption in buildings by recognizing and analyzing data outliers.

In summary, NNs are increasingly used to predict, forecast and profile energy consumptions of myriad systems. In many areas of computer science, using NNs has become standard practice. Most research does not focus specifically on the methods used, but rather report on the accuracy and effectiveness of their methods. This is because it is not yet fully understood which machine learning approach is the most suitable for the domain of energy consumption in emerging digital ecosystems. Furthermore, the success of a particular NN method often relies on the configuration and fine-tuning of the NN rather than the exact NN method used.

3.3 Control Engineering and Feedback Loops

This section provides background on an engineering cornerstone—control theory and in particular feedback loops in software engineering applications. Feedback loops are everywhere, heavily used in engineering solutions, and are increasingly finding their way into software engineering applications (e.g., control of web services, robots, drones or driverless cars). Self-adaptive and self-managing systems, as discussed in Sec. 3.2, feature feedback loops at various levels of abstraction.

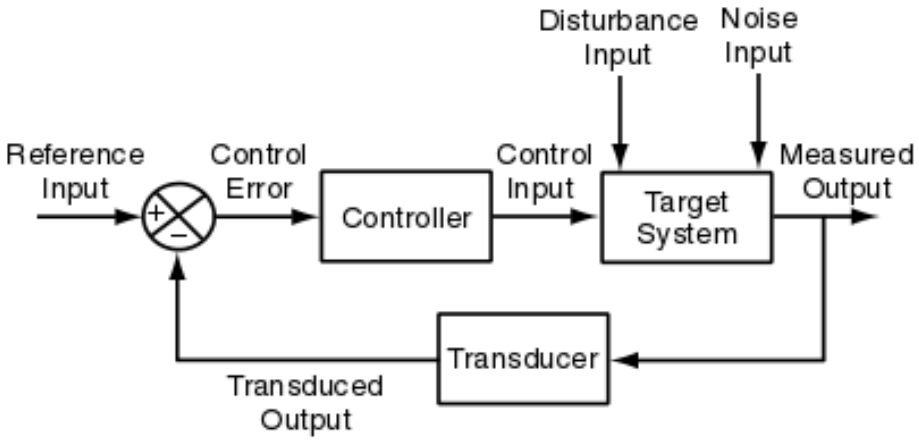

Figure 3.3: Classic control loop [57]

Stability and *robustness* are two desired properties of control systems. Control theory provides mathematical foundations to analyze these properties in control systems. The idea of a feedback loop is compare the measured input to the system's reference input. The basic components of a feedback control loop are depicted in

Figure 3.3. The control error is computed as the difference between the measured inputs and the measured outputs. If the target system does not provide the desired output, then the controller modifies the control input to alter the behaviour of the target system towards achieving the desired goal. Disturbance input, or noise, are often environment factors, which can alter the behaviour of the target system.

A control system must be robust, that is, the ability to function in environments where the input variables are different than those of the ideal model. This is because the inputs to the system do not necessarily conform to the envisioned inputs when the system was designed. Additionally, the model used by the controller may be linearized or simplified to ease calculations, yet must be able to function in an environment where many more variables affect the stability of the system.

The measured output is the behaviour of the system which can be measured as a result of its activity. In other words, the system measures some variables after it made modifications to itself with the goal of achieving stability. The reference input is simply the desired state the system should achieve based on a measurable parameter. By comparing the reference input (i.e., the desired state) to the output (i.e., the actual state) the system can compensate, through its own action of varying the inputs under its control (e.g., cruise control on a car: when the car approaches a hill, the car slows down due to the hill; as a result, the cruise control compensates by accelerating to maintain the desired speed). Most systems and control loops also feature a transducer, which is a component that translates the output measurements so that they can be compared to the reference inputs [57].

Control theory has produced many control schemes. Arguably the most common control scheme is a *PID (proportional, integral and derivative) controller.* Depending on the desired properties of the system, its controller will have proportional, integral and derivative terms, which are applied to the inputs and outputs in order to maneuver the system to its desired state [57]. Note that not all three PID components have to be present in a control system; omissions of one or two components is justified for certain applications.

The proportional component of a PID controller acts to move the system in the general direction of the desired correction (e.g., the car on speed control is slowing

down, the controller effects an acceleration increase). The integral component acts in such a way as to reject sharp/large changes in the error measurement (i.e., difference between the observed and reference values). This is useful to ensure that temporary steep changes do not cause incorrect or overreactions by the system (i.e., consider a failure to receive an output value in a running system, this absence should not cause the system to over react. The differential component dampens or smoothens the system's changes into a more controlled and gradual process to achieve stability. Equation 3.1 represents the general formula for a PID controller [136]. To configure a control system, engineers change or tune the gain terms (i.e., K_p, K_i and K_d) in this formula.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_D \frac{d}{dt} e(t) \tag{3.1}$$

In our research, we we aim to optimize energy consumption by letting a controller determine the ideal server on which a particular piece of software should run to minimize the consumption of energy. Aided by the fact that we have multiple types of servers in the data centre, we need to ensure that the controller models these server types with their different energy consumption features effectively.

For ease of design, simple control loops are always preferred. However, for large data centres and complex digital ecosystems (e.g., IoT and CPS), addressing all the needs and requirements with one control loop is challenging. To separate concerns, software engineers resort to multiple control loops which are often organized hierarchically, as for example in the autonomic computing reference architecture (ACRA) presented in Figure 3.2 above. In such a hierarchy, higher level controllers feed policies to lower level controllers and, thus, affect or change these lower level controllers, which is called *adaptive control.* Adaptive control is eminent when system requirements, contextual information or optimization criteria evolve. Upon dramatic changes in the environment of a digital ecosystem, the controller can no longer effect the desired outcomes in the managed system. As a result, the control model and algorithm have to evolve too.

Implementing control systems in software has the big advantage that the different components (i.e., MAPE-K components and the hierarchical components (AEs) for

adaptive control) can be realized separately instead of realizing the entire control system as one piece of hardware. Moreover, the controllers of lower levels in adaptive control systems can be readily modified—not simply by adjusting gains, but rather by modifying the models of lower level controllers (e.g., by changing variables, data structures, function calls and entire algorithms).

To facilitate dynamic controller models, a new research area has emerged in the software engineering community called *models at runtime (MARTs).* [15] Szvetits and Zdun have conducted a comprehensive systematic literature review of MARTS [122]. They identify four main areas within MARTs: a) *models,* b) *techniques,* c) *architectures* and d) *objectives.* In our research, the relevant areas of *monitoring*, *simulation*, and *adaptation* belong to the MART subgroup of *Objectives*. Control theory, and more specifically *autonomic control loops*, belong in the MART subgroup *Techniques.* Furthermore, this dissertation's use of a *model-aware framework* falls into the MART category *Architectures.* Models and MART play a central role in self-adaptive systems and will continue to grow in future digital ecosystems, such as IoT and CPS [80].
In our research, we develop MARTs for *energy-aware infrastructure* and in turn *energy-aware systems* and optimizing energy consumption and system performance in heterogeneous environments, including data centres and complex digital ecosystems. The key goal is to manage and optimize the trade-off between the non-functional requirements of energy consumption and system performance using adaptive control systems using MARTs. Models to address performance and efficiency often rely on queueing theory (for timing) [31, 49], Markov models, and runtime architecture models [6,7,122]. Energy models for different types of software running on different servers help us predict future energy consumption for different deployment configurations.

Over the years, the control community has defined several reference models for adaptive control. The prominent models are *Model reference adaptive control (MRAC)* [24, 35] as depicted in Figure 3.4 and *Model identification adaptive controller (MIAC)* [118] and as depicted in Figure 3.5. MRAC is also referred to as *Model reference adaptive system (MRAS)* [35]. All of these reference models are two-tier architectures for adaptive control; the two layers correspond to the bottom two layers of the IBM ACRA reference architecture. MRAC and MIAC differ in how the controller model in the top level is built. For MRAC, the top controller is defined at design time often by simulating of the underlying system. For MIAC, the top controller is computed at

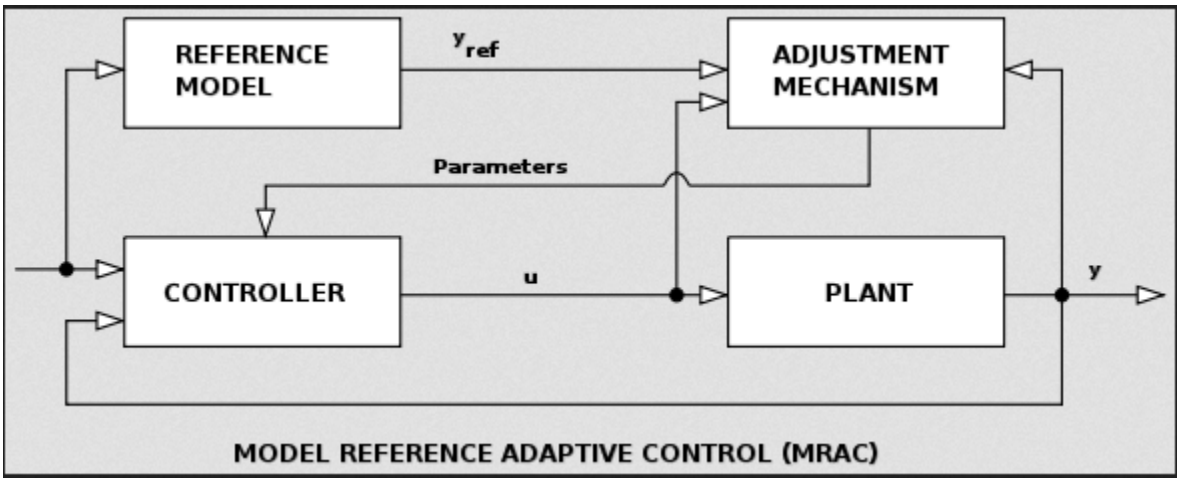

Figure 3.4: Reference architecture for model reference adaptive control (MRAC)

runtime by observing the behaviour of the managed system below and then *identifying its model.* Depending on the type of application, one model is more appropriate than the other to realize the desired level of adaptivity and system behaviour.

Today, the two layers of these adaptive control architectures are typically realized in software as autonomic elements and their controllers (i.e., models and algorithms) as autonomic managers (i.e., MAPE-K) as depicted in Figure 3.1. Thus, the top-level controller can adapt the individual software components that make up the controller of the bottom-level controller.

The software engineering community and the SEAMS community in particular have evolved the reference architectures for adaptive control emanating from the control communities further and defined several three-tier reference architectures for adaptive systems. One of the earliest model is the Figure 8 reference model by Oreizy et al. [102]. Kramer and Magee gleaned ideas from the AI and robotics communities to manage trade-offs in adaptable architectures, where the three tiers realize *component control*, *change management* and *goal management*. These three layers nicely correspond to the ACRA layers of *resource control*, *self-* management* and *human goal and trade-off management*.

Villegas et al. defined the three-tier DYNAMICO reference model as depicted in Figure 3.6 to manage control objectives and context relevance in self-adaptive software systems [129]. The three tiers represent *goals, adaptive or self-adaptive systems* and *managed adaptive infrastructure.* In DYNAMICO, separation of concerns goes

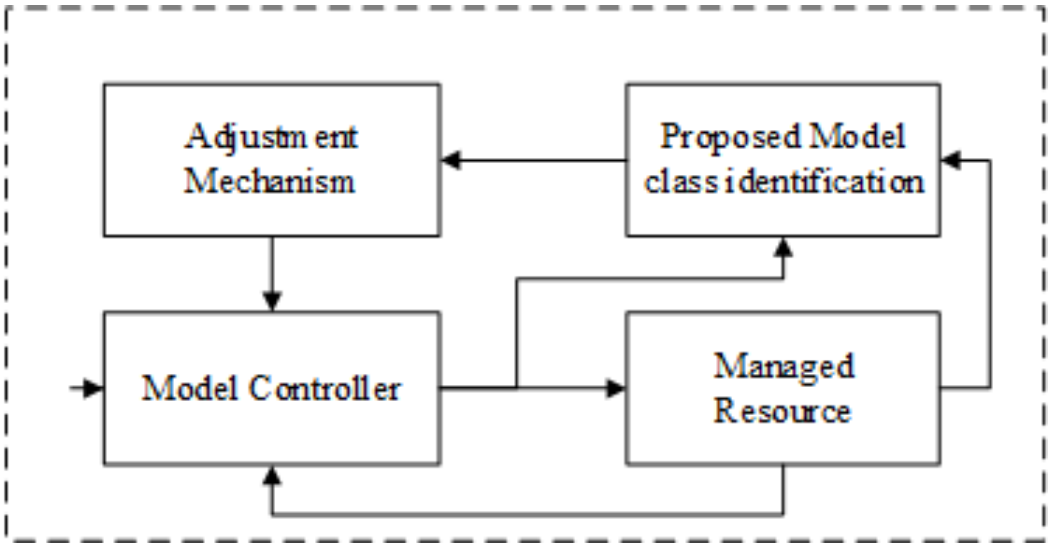

Figure 3.5: Reference architecture for model identification adaptive control (MIAC)

beyond the decoupling of adaptation mechanisms for managed systems. This architecture uses three different types of MAPE-K loops that interact among themselves to address changes at three levels: *control objectives, adaptation* and *monitoring*. DYNAMICO aims to address the need in self-adaptive systems to provision the controller with the ability to adapt to changing monitoring infrastructure, changing goals and the dynamic contexts. High-level goals, such as energy consumption, are specified and translated into actions through the planner and executor. The middle layer addresses the adaptation required for self-adaptive systems, following a more traditional approach as found in control theory. The bottom layer in DYNAMICO realizes the additional need for adapting the monitoring infrastructure. We use DYNAMICO as a foundation for our energy-aware model and framework.

3.4 Simulation, Signal Processing and Statistical Analysis

Due to the prohibitive costs of testing experimental scheduling research in a real data centre, scientists and engineers employ simulations. Simulations are a cost effective alternative to real hardware, networking, cooling, and facility expenses. Additionally, simulations allow the experimenter to extrapolate their approaches to realistic scales

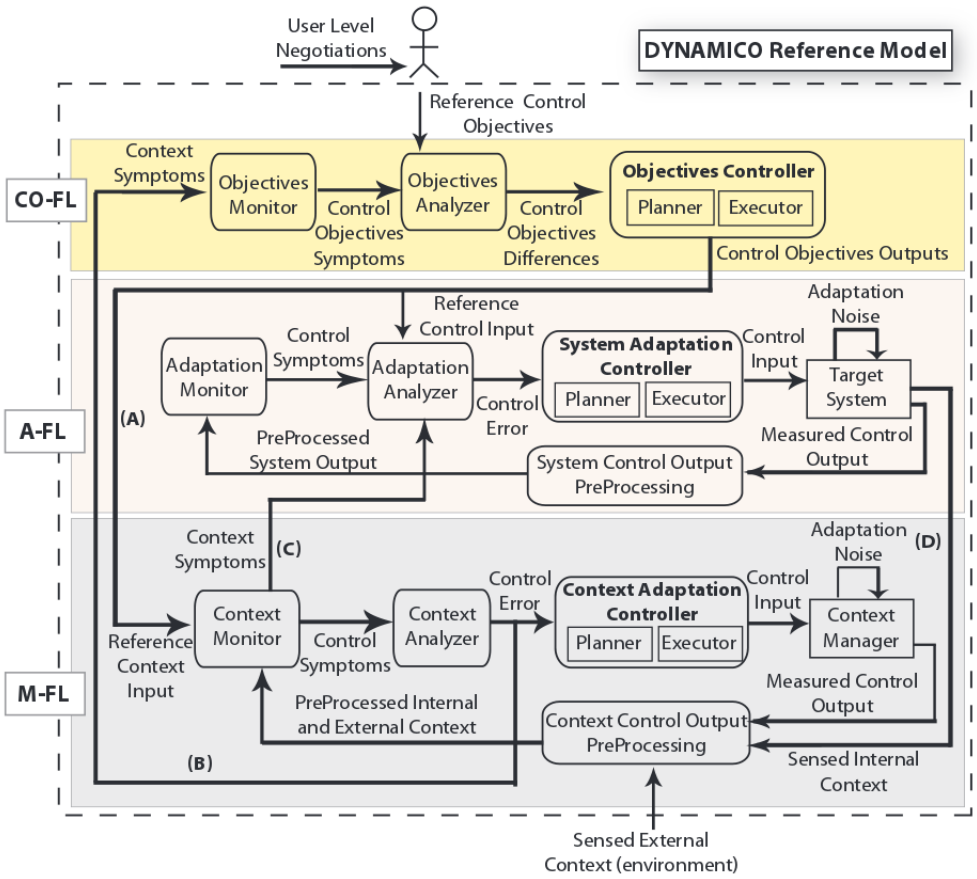

Figure 3.6: DYNAMICO reference model [129]

and, thus, evaluate their research more effectively.

A significant drawback to simulations, however, are the limitations in the variable environment conditions that can be simulated, or the fidelity of the system that is being modeled. As an experimenter, we are limited to only simulate the variables that the simulation software is provisioned for. This severely limits the ability to transfer the results directly to a real data centre simply because the real world is much more complex than can be expressed in simulations. Subsequently, simulations serve as an approximate representation of reality and all results must be considered carefully before drawing conclusions about real world data centres.

Signal processing plays another important role; it allows us to analyze the sensor data from hardware components. Signal processing lets us extract pertinent infor-

mation from seemingly noisy signals. The signals which we process may need to be preprocessed or transformed in order to amplify or filter the pertinent information first. We primarily use signal processing to detect patterns which will then be used by our machine learning component for classification.

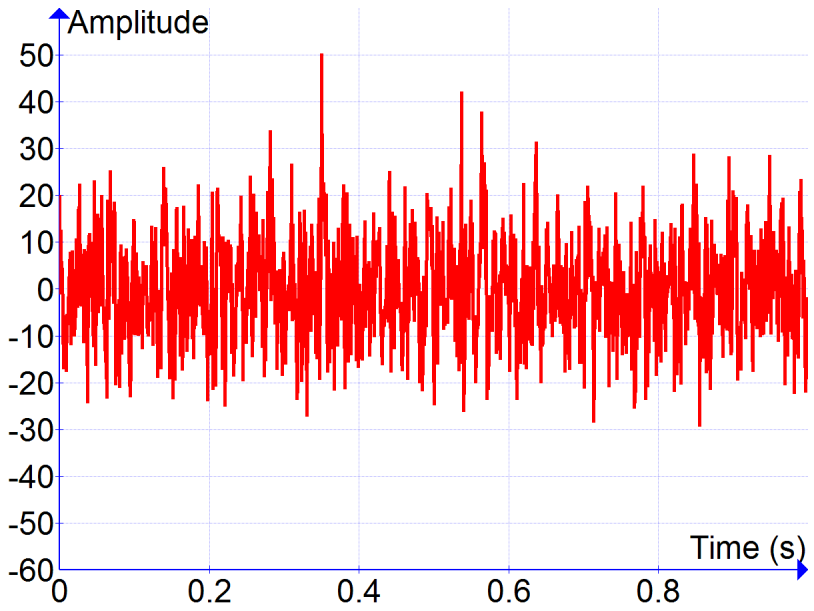

Figure 3.7: Artificially generated, noisy signal to illustrate the effects of Fourier Transform [137].

Signals are generally divided into continuous-time signal and discrete-time signals. Examples of continuous-time signals are voltage, current, temperature, or speed to name a few. The main feature of these types of signals is that as time changes in infinitesimally small increments, there is a unique value for the signal. Contrary to this are discrete-time signals. Examples of discrete-time signals are daily minimum/-maximum temperatures, lap times in races, or sampled continuous signals. The data we obtain from our data centre instrumentation is in the form of discrete-time signals. We obtain the total voltage, amperage, kW and kW/h at one minute intervals.

One such technique is demonstrated in Figure 3.7. Here a seemingly noisy signal has been recorded and is displayed in the figure. However, applying a specific signal processing technique referred to as *Fourier Transform,* Figure 3.8 shows that we can

extract and identify three distinct frequencies within this seemingly noisy signal.

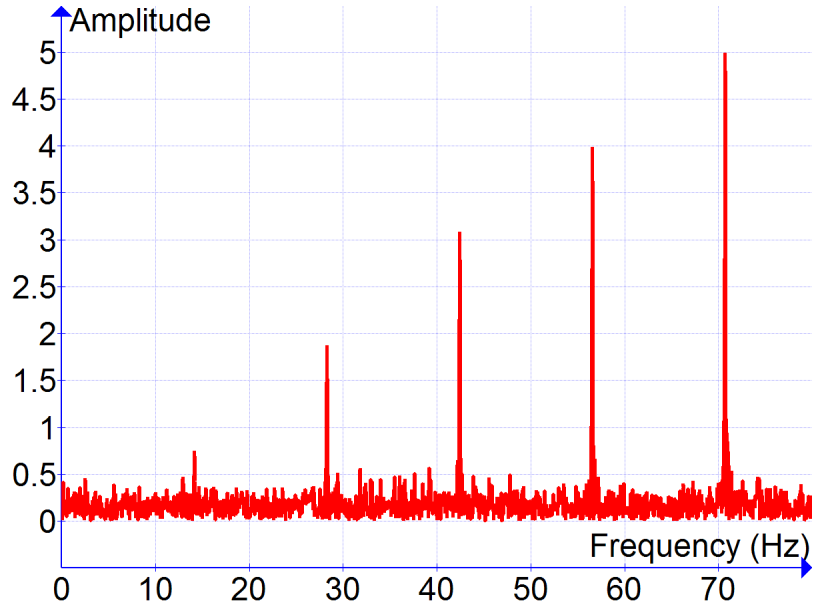

Figure 3.8: The result of the Fourier Transform applied to Figure 3.7 exhibiting distinct frequencies present in a seemingly noisy signal [137].

We employ such techniques in our research to identify the type of process and resource that is generating a given energy consumption pattern.

Statistical analysis helps us to make sense of a sea of data and come up with meaningful statements about the data collected in experiments. It supplements the techniques we use from signal processing and machine learning.

3.4.1 Simulating Energy Consumption in Data Centres

The most relevant simulation software for data centres in academia is *CloudSim* [30]. CloudSim allows the user to construct a basic model of a data centre. The Java based implementation can be extended and integrated in custom models.

While CloudSim continues to be improved upon and more features are added constantly, the functionality of CloudSim fits specific environments. Only recently, CloudSim added the functionality to model primitive energy consumption statistics.

Another related tool is *Cloudanalyst* [134], a tool based on CloudSim. This tool, as its name suggests, lets the user perform certain analyses on CloudSim recorded metrics. Cloudanalyst makes data extraction and analysis seemingly easier to perform than using CloudSim directly—albeit this depends on the experimenter using the tool.

We use simulation only in a limited fashion. Instead of simulations, we use real data, generated by real workloads and processes representative of real workloads, in a real data centre. This has the advantage that we are not limited in the complexity and number of variables composing reality that affect our measurements. Simulations can only change, affect and measure the features which are included in the simulation software. For us, using real data centres, our measurements are affected by a plethora of known and unknown variables (higher fidelity), all which need to be taken into account.

3.4.2 Signal Processing and Statistical Analysis

There is a common approach to data science involving signal processing and machine learning as depicted in Figure 3.8 The first stage preprocesses the data including acquisition of a signal, thresholding of the output and separating data useful for classification from the noise. The second step in the process is the feature extraction scheme, which computes a feature vector from a regular vector. A feature is a distinctive or characteristic measurement or structural component extracted from a segment of a pattern. Statistical characteristics and syntactic descriptions are the two major subdivisions of the conventional feature extraction modalities. A feature extraction scheme is meant to choose the features which are most important for the classification exercise. The final stage is signal classification, which can be solved by a variety of approaches, including linear analysis, nonlinear analysis, adaptive algorithms, clustering and neural networks.

Chapter 5 presents an introduction to the mathematical methods employed in our adaptive system for classification of a process' resource utilization based on its energy consumption pattern.

The most important step for the classification task is extracting a suitable set of features that has the capability to differentiate among different classes. Statistical

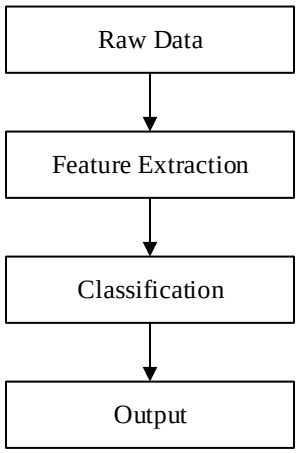

Figure 3.9: Classification process stages

analysis is a way to generate such representations which allow us to differentiate between classes of data within our data set. The method for feature extraction consists of decomposing the server's energy and power consumption data obtained while our experiments are run.

3.5 Chapter Summary

In this chapter introduced related work and background materials essential for our research. We covered recent advances in green computing from the high level analysis to low level energy measurements of individual machine instructions.

This chapter also highlighted some of the main features of self-adaptive systems. Section 3.2 connected green computing and self-adaptive systems as they are helpful in understanding and solving the problems we are addressing in our research.

We also gave a brief overview of control theory and feedback loops to the extent that they are needed in this dissertation. The exact details of these fields are fascinating. However, we often find ourselves in a position of employing user-friendly implementations of feedback systems and control loops when we use standard software tools such as Matlab.[1]

[1]http://www.mathworks.com/products/matlab

Finally, we provided a brief overview of the related signal processing concepts, which help us analyze the often massive amounts of data collected from our data centre and test servers. Similar to control and feedback systems, the most common use of signal processing comes in the form of user-friendly software packages. Here the deep internals of the systems are abstracted to a higher level with which we as the experimenter can interface to get results quickly and accurately.

Chapter 4

Research Methodology

This chapter discusses the methodology we use to develop and answer our research questions. We use a combination of case studies, artifact generation and prototyping as part of our experiments in order to investigate our research questions. Additionally, a systematic literature survey (cf. Chapter 3) provides the basis for focused research efforts on open questions in this field. Identifying particular open questions allows us to create targeted case studies from which we obtain data geared towards specifically addressing these open research questions.

Keele et al. describe systematic literature studies, the method by which to conduct them, and the benefits and drawbacks of the different methods [69]. Yin [142] and Easterbrook et al. [45] provide insight into the design and use of case studies. Runeson [113] posits that case studies and empirical studies have become an accepted method of conducting software engineering research. Thus, case studies are an approriate method to adress the research questions in our research.

Additionally, we conduct experiments to answer selected research questions. In particular, our experiments address research questions pertaining to the identification and isolation of resource types and their impact on energy consumption. Our findings are used to evaluate the efficacy of our models for improving energy consumption in digital ecosystems of interconnected applications.

4.1 Process and Hypothesis Formulation

This section describes the process we used in the development of our experiments, our research hypothesis, and our problem formulation guiding this research.

4.1.1 Initial Hypothesis

A simplified expression of our research hypothesis is that energy consumption of software varies depending on the hardware resources. Moreover, this difference can be exploited by scheduling tasks in such a way as to gain energy savings while satisfying performance criteria.

We developed our research hypothesis incrementally over several iterations. The initial idea was sparked and inspired by Hindle et al.'s research in energy consumption of mobile devices [58, 59]. They demonstrate that different versions of software appear to generate different energy consumption patterns on the same device. From this finding, the authors argue that bugs can be identified in consecutive versions of the software. Taking this work further, Hindle and Rasmussen were able to investigate the energy consumption of particular types of software running on devices [110]. In our approach, we follow more closely the work of Balasubramanian et al. who investigate energy consumption of different resources on mobile platforms [9]. Both approaches compare different versions of software on a single device over time.

In our research we explore the inverse of Hindle's and Balasubramanian's research (i.e., multiple versions of software on a single hardware platform). We explore energy consumption of single pieces of software on multiple devices using, nominally, identical hardware resources.

While mobile technologies and energy consumption were hot topics over the past decade, the availability and potential scalability of data centers is now of great interest in this line esearch. Having two fully instrumented server racks in a data center with heterogeneous server hardware at our finger tips greatly aided us in our decision to investigate energy consumption in data centers rather in the realm of mobile devices. However, findings and lessons from one realm are often transferable to the other, so the decision to chose servers instead of mobile devices is, with certain caveats, acceptable.

4.1.2 Hypothesis Refinement and Iterative Approach

The first challenge in our research was to determine whether energy consumption of software in data centers is measurable. To answer this, we contacted the University's data center staff. As a result, we were given access to their monitoring infrastructure and were able to observe the energy consumption of our servers in near real time. Once we had the ability to monitor our servers, we set out to determine (1) whether software energy consumption can be measured; (2) whether thresholds discernible measurements would occur; and (3) whether there were identifiable patterns that would link energy consumption to specific software processes.

This initial phase was exploratory and involved running extreme workloads on the servers while waiting to observe changes in energy consumption. Fortunately, our initial intuition, that software energy consumption is measurable and detectable, appeared to be correct through visual inspection of the data.

The second phase was to determine the detectability of energy consumption changes when using different hardware resources. For this, we followed a similar approach of introducing extreme workloads on the servers and identifying, visually, changes in energy consumption. This was done to establish whether this type of research is feasible and repeatable across different servers.

Once we were satisfied that the fundamental requirements (i.e., detectability of energy consumption changes based on resource usage over time), we developed structured, automated and repeatable test frameworks to produce energy consumption profiles. Later these automatically produced profiles were used to automate the tasks of identifying software processes based on energy consumption as well as their integration into our dynamic models to potentially affect resource allocations.

4.2 Data—Sources, Collection and Analysis

This section provides a brief overview of our data sources, data collection methods, and data analysis.

4.2.1 Data Sources

Data is crucial for evaluating the effectiveness of new or proposed methods. In software engineering, available sources of data range from mathematical models, simulation and emulation to direct measurements and tests. Obtaining data from mathematical models, simulation or emulations is generally a viable option in most cases. The benefit of this type of approach is that we are able to scale the system in, for example, simulations to near realistic magnitude. However, one downside, among many, is that we are only able to approximate the conditions and complexity of reality. The second source of data is to directly deal with real world hardware and software. A sizable data centre is nearly impossible to come by in academia.

The Enterprise Data Center (EDC)[1] at University of Victoria (UVic) provides server and data processing capacity. The EDC2 facility can accommodate up to 1.26 mega watts of power or up to 3,000 standard servers. This facility currently hosts many administrative servers and research computing facilities including high performance computing nodes such as WestGrid[2] and Future Internet nodes such as Global Environment for Network Innovations (GENI)[3] and NSERC Strategic Network for Smart Applications on Virtual Infrastructure (SAVI)[4]. The UVic EDC2 center is designed to back up systems for continuous operation 365 days a year through Uninterrupted Power Supply (UPS) and diesel power generation. Servers get power through a three-phase 208V power distribution system. Each rack is connected through two independent (on different phases) 30A breaker circuits so that servers with redundant power supplies can benefit from the power distribution redundancy.

Furthermore, we also make use of built in reporting software and sensors which are commonly found on modern devices (and smartphones) as well.

4.2.2 Data Collection

Power consumption can be measured and monitored along the entire multi-level hierarchy of an enterprise data center—from individual core and memory units, to servers and racks, as well as PDU and UPS units. Power instrumentation schemes can em-

[1]https://www.uvic.ca/campusplanning/completed-projects/enterprisedatacentre2/index.php
[2]https://www.westgrid.ca/
[3]http://www.geni.net/
[4]http://www.savinetwork.ca/

ploy direct measurement techniques by using various hardware sensors or modelling techniques to estimate power. For example, many recent processor architectures incorporate power reducing mechanisms, whereby the CPU frequency is throttled back to reduce power consumption. Moreover, CPUs can be put into selected states at run time depending upon their load. Furthermore, when the CPU is idling, the frequency at which the CPU operates is adjusted dynamically as more power is consumed at higher frequencies.

Tools, such as the Unix PowerTop utility and Joulemeter,[5] monitor how programs use the CPU mode features and estimate power consumption accordingly. Software designers use some of these tools when designing programs to reduce overall power usage of their programs, though usage is not widely adopted.

Many servers have built-in power dashboards to monitor vital system functions and to estimate power and bandwidth consumption. In the realm of cloud computing and resource scheduling, power-aware frameworks are rapidly emerging [145]. The fundamental premise is to provide power instrumentation and monitoring for software systems to optimize power consumption effectively and adaptively.

When building an enterprise data center today, power utilization, consumption and monitoring are at the forefront of the designers' minds.[6] Energy usage trends can be determined by calculating the power usage effectiveness (PUE) of the entire enterprise data center using UPS power consumption measurements. Arguably, the best view of data center power consumption comes from the PDUs (power distribution units) located within the racks. Modern rack-mounted PDUs feature integrated monitoring and control capabilities to enable continuous power monitoring. These PDUs are further configurable to send alerts and updates to centralized management agents for data analysis.

Our research focuses on server level power consumption to investigate how the compute, memory, disk, and network loads affect power usage of servers from a software standpoint. Data center utilization, in commercial settings, is typically only

[5]M. Goraczko, A. Kansal, J. Liu, F. Zhao: Joulemeter—Computational Energy Measurement and Optimization, Microsoft Research, http://research.microsoft.com/en-us/projects/joulemeter/

[6]B. Kleyman: Why monitoring data center power consumption is vital, TechTarget, http://searchdatacenter.techtarget.com

20-30% and, yet, at low (or idle) loads, the bare hardware can consume as much as 50% of the server power [12,66,94]. Thus, major power savings can be accrued if idle servers can be switched off in data centers.

Hence, we use the built-in instrumentation of our UVic EDC2 data center to obtain power measurements of the rack that hosts our own research servers for GENI and SAVI network nodes. This rack, which is connected to three circuits that power the servers in this rack, hosts five of our servers on one circuit while the other circuits are used to power some of our other servers. For this experiment our server pack is in one single two-phase circuit (i.e., no redundancy). Moreover, the data center provides the capabilities to report circuit power consumption. The power measurements include measurements and statistics on current consumption, real power usage and power factor measurements.

4.2.3 Data Analysis

Data Analysis is conducted in various ways. On a basic level, when appropriate, we make use of simple statistical methods (e.g., mean, median, average), assuming Gaussian distributions for the underlying data. In cases where data cannot be fitted into normal distributions, long tail distributions may be of use. Alternatively, since our research is grounded in using real hardware and real software, we can resort to employing top-level machine learning approaches to the data (or signal).

There is a common path for pattern recognition, which follows the structure laid out in Figure 3.9. The first stage is preprocessing the data. This includes acquisition of a signal, thresholding of the output and separating the data useful for classification from the noise.

The second step in the process is the feature extraction scheme which helps us distinguish between a feature vector and a regular vector. A feature is a distinctive or characteristic measurement, transform, structural component extracted from a segment of a pattern. Statistical characteristics and syntactic descriptions are the two major subdivisions of the conventional feature extraction modalities. A good feature extraction scheme is meant to choose the features or information which is the most important for the classification problem. The final stage is pattern classification. This

step in Figure 3.9 can be solved by a variety of approaches: linear analysis, nonlinear analysis, adaptive algorithms, clustering and neural networks.

The most important step for the classification task is extracting a suitable set of features that has the capability to differentiate among different classes. As discussed, statistical analysis is a way to generate such representations. The method for feature extraction consists of decomposing the server's energy and power consumption experimental data. Subsequent runs with different software configurations, different hardware, different operating system constraints then offer insights into the effectiveness of our methods to reduce energy consumption tailored specifically for the machines at hand. This has the advantage of increased accuracy and avoids the dangers of model risk, since our insights are dynamically generated and customized specifically to the hardware in the data center. At the same time, our approach is dynamic and can be applied to any server in the data center.

4.3 Dynamic Models at Runtime

This section introduces our reference model which is both based on the experimental results and benchmarks, as well as the interactions with hardware components such as servers and the intended scheduling model. We also introduce the models' effects on energy consumption as they are measured in our experimental setup in the UVic data cente. This topic is further explored in Chapter 7.

4.3.1 Model Design Process

In order to effect energy saving changes in a data centre dynamically and automatically, we need a reference model upon which a software architecture and implementation can be based. Our model is based heavily on ACRA (cf. Figure 3.2) with internal MAPE-K components [63] and as such makes use of its conceptual components: Monitor, Analyze, Plan and Execute. Models are the foundation for software architectures and high-level system descriptions. Likewise, according to Bass et al. in software engineering models are used to describe a decomposition of the different conceptual parts that comprise the underlying system and also describes their relationships [68].

The different conceptual components in our system are partitioned into the fol-

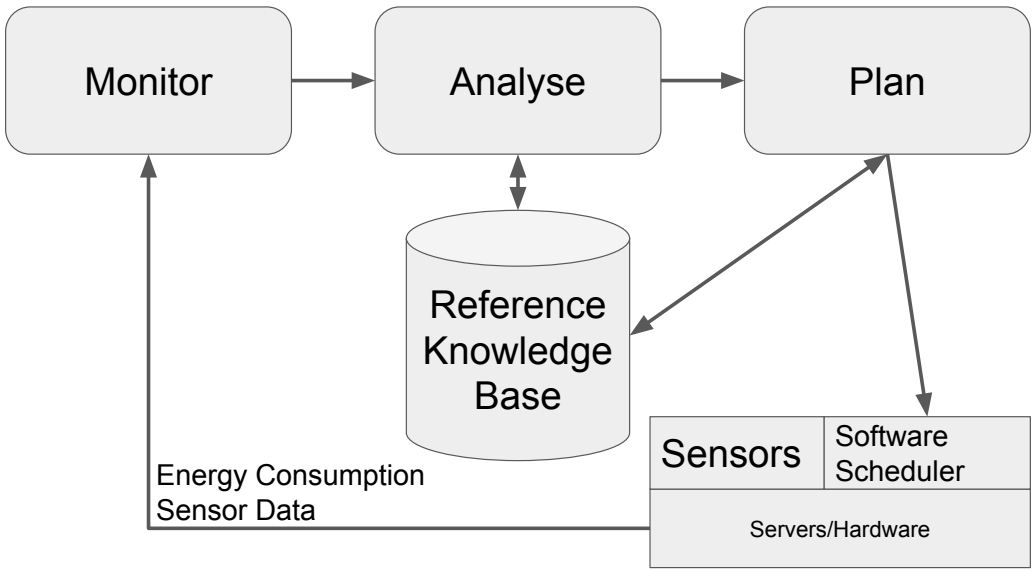

Figure 4.1: Our MAPE-K model to manage energy consumption and effect energy optimization in data center software

lowing distinct categories: *Servers, Resources within servers, contextual energy data, software applications* and the overall data centre.

Each of these components is a feedback loop and can be classified as a MAPE-K model. Figure 4.1 illustrates the composition of our reference model. In this model, sensor data (e.g., energy consumption) is delivered to the *Monitor* component of the model.

The *Monitor* component pre-processes the data and forwards it to the *Analysis* component. Alternatively, the monitor component can also function as a simple forwarding station without preprocessing. In that case the preprocessing has to occur at the *Analysis* state. At the *Analysis* component, the preprocessed data is incorporated and referenced with existing data points from the *Knowledge Base*. The purpose of this is to infer useful information about the energy consumption as well as resource usage of any processes that are connected to the underlying hardware servers. It is at this phase that power consumption and energy usage are correlated to resource usage of software applications. To do this, we employ Neural Networks (NNs) to help

with the dynamic identification of energy consumption and resource usage as detailed in Section 6.3 below. [19] This classification of processes, based primarily on energy consumption, aims to reduce power consumption in applications' digital ecosystems by way of making scheduling and job placement decisions primarily on the expected and measured changes in power consumption for particular components which interact with the applications (e.g., servers, mobile network infrastructure, networking equipment and mobile device resources).

Once a process has been identified as possessing a particular power profile and based on the nature of its resource requirements, adaptive job placement can be orchestrated within the application's digital ecosystem to optimize scheduling and resource utilization with the goal of direct energy savings. This is done at the *Planning* stage, which essentially controls the adaptation of the entire system.[7] The planning, or adaptation, stage determines whether to assign a particular hardware to a given software application. Initially, some data must be gathered on any software application. Only after data has been gathered, through the software applications contextual sensor data, can the planning phase make decisions about whether to migrate the software application to another server.

4.4 Limitations

The collection of data and its evaluation is, by necessity and time constraints, limited to a relatively small number of heterogeneous devices (i.e., the devices we can readily access). This small number of devices (i.e., two types of servers, two types of laptops, three types of desktops and two mobile devices) is a relatively small sample to test our framework for generalizability across platforms. However, since these devices differ to a large degree, an argument can be made to consider success with our limited resources to be extensible to a larger scale. This, however, is only possible if statistical rigour is observed and sufficiently large data samples are obtained to support our findings and derived claims.

Similarly, the evaluation of our models through the use of actual measurements is restricted by the allocated time frame within which to conduct the experiments.

[7]Note that by *ecosystem* we mean the entire data center, including software and hardware resources.

Statistical analysis and significance must be obtained for each case studies' components in order to ensure validity in the findings. However, due to the diverse nature of the experiments, the number of runs required to obtain statistical significance in the analysis, and thereby the required time, varied among experiments.

4.5 Chapter Summary

This chapter describes our research approach to investigating power consumption digital ecosystems. We highlight, and bring together, the components needed to answer our research questions effectively. We use literature surveys to position our work at the forefront with open questions in the realm of green computing and energy consumption. With these literature surveys, we combine case studies, experiments on real hardware with real software to guide our hypotheses development.

This chapter also introduces the reader to the infrastructure available to us at the University of Victoria on which we conduct our research. The data center is described in detail along with the servers in the server racks we can access readily.

Additionally, we position our model within the ACRA and MAPE-K reference models. We also introduce the reader to the methods we use when analyzing the data. Lastly, we present some of the challenges as well as limitations to our research approach.

Chapter 5

Experiments

This chapter provides a detailed description of the experimental setup for our experiments. We ran the experiments in a real data center using real hardware and software applications.

5.1 Our Approach to Energy Consumption Experiments

This dissertation focuses on digital ecosystems as they exist through the interplay of resource utilization of software applications stemming from network bandwidth, latency, storage and processing capabilities. Figure 5.1 illustrates our intended target for performance and energy optimization. The primary interest of this work is to focus on the application of self-adaptive systems, and context-aware resource scheduling within digital ecosystems for the purpose of conserving energy while completing the applications tasks without a incurring performance penalties. We now provide a brief overview of the experimental phases, milestones and challenges.

To investigate the interplay between energy consumption, hardware resource usage and adaptive scheduling and software applications thoroughly, we have chosen the following practical approach. First, we needed to determine whether different software applications exhibit measurables differences in their energy consumption. The ability to detect different energy consumption levels and associate them with different software applications forms the foundation of this work and, thus, was a necessary precondition. Our first experiments focused on determining whether this

hypothesis holds—we can indeed differentiate between different software processes by their energy consumption.

Once this first hypothesis had been preliminarily established as true for most cases, our next step included to scale the experiment horizontally. This meant that we essentially tested the hypothesis on different server hardware to ensure that it would hold in a data center which naturally is composed of multiple types of server hardware. Here it became evident that this task needed to be automated in order to defend the credibility and applicability of this research to the real world data center operators. Running single experiments once in a controlled lab setting can be done manually, however, moving this phase of the data collection into a real data center with multiple machines required automation. This automation setup did not exist and is one of the contributions of our research.

Consequently, we also worked on automating the classification of software applications on the basis of their energy consumption by using a number of machine learning techniques. Once satisfied with the results we continued to test the limits of this approach using real software processes. Following this process, we developed a method, through simulation, that would allow us to demonstrate the scalability of our approach in a large data center and to demonstrate the potential energy optimizations.

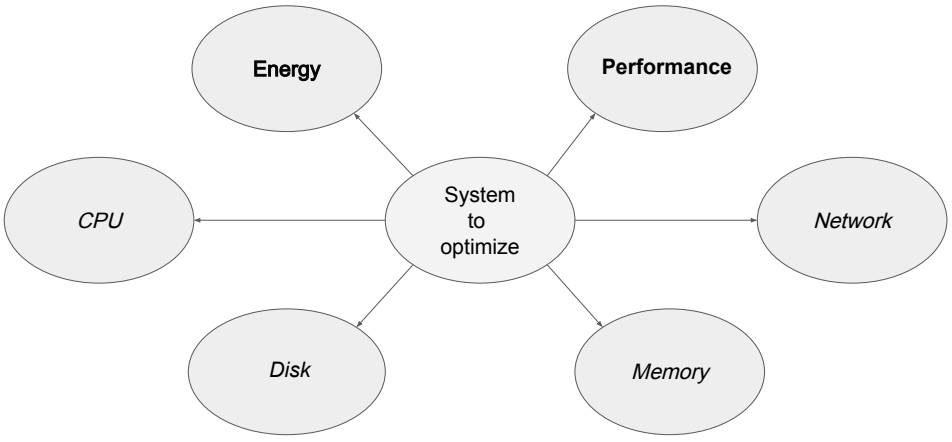

Figure 5.1: Optimizing energy use and performance in digital ecosystems

Establishing an automated classification method also led us to investigate whether software applications would have to be treated as so called "black boxes" which can only be a) run, or b) not run on a given machine. We determined that for certain types of software applications commonly available techniques can be employed in order to alter the resource consumption without affecting the application itself. As a result, we identified certain types of software applications where the choice is not just limited to a) run the application on machine A; or b) do not run the application on machine A. For those types of application, a third option exists, namely: alter the application's interactions with the underlying hardware in order to optimize it for machine A.

We identified that this third option exists for a segment of software applications that have particular resource requirements. This is possible because a software application interacts with the operating system in order to request, use and relinquish resources, such as memory, I/O or processor time. For a specific class of software applications, we were able to demonstrate that altering this communication between the software application and the operating system can have beneficial outcomes for the servers energy consumption while at the same time improve performance of the software application.

The software for our experiements targets specific resources within a server (e.g., memory, CPU, disk, network). We designed several experiments that would consume these specific resources for a given amount of time. The exact time each test software runs varies depending on the exact nature of the experiment. Having designed multiple test software programs we are able to combine and chain them together to better use the available servers and minimize the time between experiments. One such example of chaining together mutltiple pieces of test software can be seen in Figure 5.13. Here we chained together 3 different test software runs and present them in one figure. When chaining test software applications together it is important to ensure that the system had sufficient time to return to a stable state. A stable state ensures that the starting conditions for each experiment are identical and no external energy consumption influences the measurements (i.e., heat from previous experiments has dissipated and fans stopped spinning). Isolation is achieved by providing enough time between experiment runs. Additionally, we will use this type of figure frequently as it provides a large range of useful data in a concise presentation. Included in the presentation are ampere measurements for all 3 breakers, while we

focus on the top/red line of the graph as this is the breaker connected to the servers we used for the experiments. The figures also show the power consumption of the entire rack over time. All measurements are provided by the data center directly to us through visual dashboards and daily data dumps.

5.2 Balanced Energy Usage and Performance Optimization

Our approach now allows us to balance energy usage and performance. On the one hand, we are able to treat energy constraints as yet another metric for constraint based scheduling decisions. On the other hand, using the findings of our findings, software tasks and processes can now be scheduled within data centers based on their energy requirements in addition to their nominal resource requirements (e.g., CPU or memory). This is made possible because the impact of a scheduling decision is fully understood in our system before the decision is even made. This allows us to evaluate the effectiveness of a scheduling decision at runtime and leads to energy optimizations as well as (sometimes) performance optimizations.

Furthermore, this also leads to improved load balancing within the data center as hardware can now be allocated with a different metric in mind.

The following sections describe each aspect of our approach in detail and present our findings.

5.3 Experiments—Description and Process

The experiments and the processes used to obtain data points for the evaluation of our approach include profiling of hardware components, such as disk, memory, CPU and network components, in their interactions with software applications. These components are found in data center servers, laptop and desktop computers, as well as mobile devices.

5.3.1 Benchmarking and Automatic Profiling

In order to increase the usefulness, usability and impact of this work for the larger community, we completely automated the foundational process of benchmarking individual servers for energy consumption. Automation ensures repeatability and consistency among various server implementations while obtaining a similar set of data relevant for benchmarking each server's energy consumption characteristics.

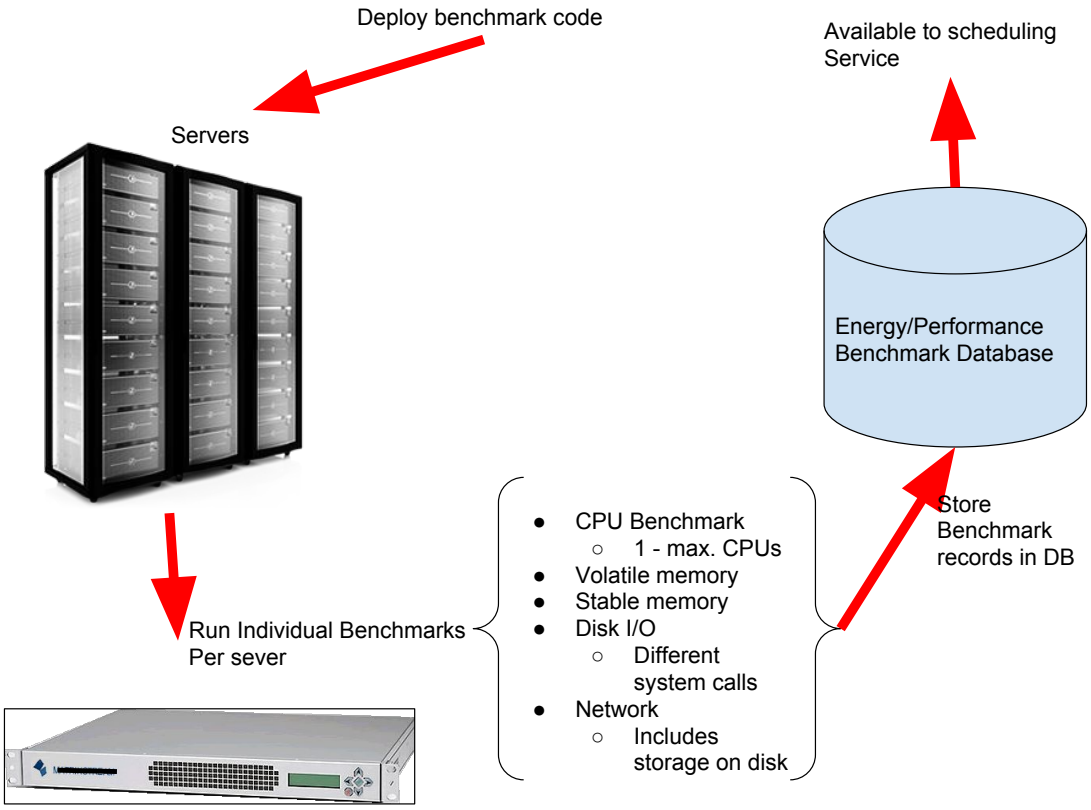

Figure 5.2: Deployment and execution cycle of automated benchmarking tool

Figure 5.2 describes the program flow of our benchmarking tool. This cycle should be run on hardware that has not been profiled previously (e.g., a server that has been newly placed in the data centre). Additionally, it may also be beneficial to run this benchmarking tool repeatedly on existing hardware. Benchmarking an existing server repeatedly can potentially be used as an early warning system for imminent hardware

failures or foreign, malicious software components that have infected the machine.

The process to benchmark and schedule a new piece of hardware is as follows. First the benchmarking tool is deployed to the data centre. From there the script (cf. Appendix A.5) is distributed automatically to individual servers. Once the script is deployed it is automatically triggered to run via `cron jobs`.[1] While the program runs, it performs a sequence of different and unique benchmarks that allow us to identify energy usage patterns in different software applications based on specific resource requirements. The benchmark performs tests on all hardware aspects, such as CPU, storage, network and memory. CPU tests fully load a minimum of one CPU and a maximum of all available CPUs (cf. Appendixes A.5 and A.6). The maximum of CPUs is defined as the maximally available number of cores on a server's processor chip set. Following the CPU benchmark, the tool runs various memory tests (cf. Appendixes A.7, A.8 and A.9). In each memory test, a different amount of memory is requested. For volatile memory tests, the contents of the memory are altered constantly while the CPU utilization is maintained at a low and stable level in order to eliminate contamination of the data points. Stable memory tests on the other hand allocate memory once and then do not alter it. This variety of tests is needed to obtain a profile of the possible energy consumption levels for different tasks. The script also performs disk I/O tests using a multitude of system calls to effect disk I/O operations (cf. Appendixes A.1, A.2, A.3 and A.4). Finally, network I/O operations are intended to be benchmarked for energy consumption. Here the data obtained over the network is either destroyed immediately, by sending it to `/dev/null`, or is written to the disk to avoid cross contamination with other data points, such as storage and CPU benchmarks.

Figure 5.3 illustrates the relationship and hierarchy between the different testing components. This is a simplified version of the call graph, the complete version can be found in the appendix of this document. The entire benchmarking process is controlled and automated via a simple software script. Once this bash script is moved to the server, it will call a custom Python program, which will run the CPU, memory and storage benchmarks. The network benchmark is controlled separately through another Bash script. We decided to use `wget`,[2] because it is readily available as a tool which fulfills all our needs to transfer files across a network. Thus, no extra custom

[1] https://en.wikipedia.org/wiki/Cron
[2] https://en.wikipedia.org/wiki/Wget

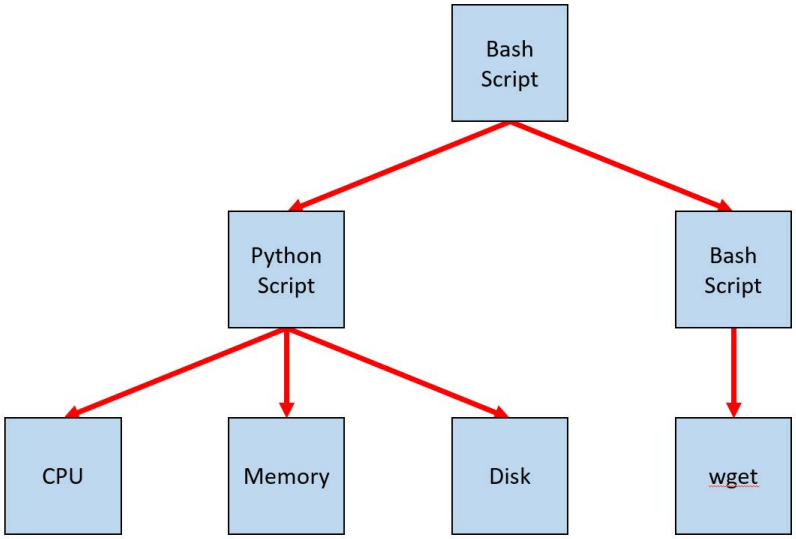

Figure 5.3: Simplified hierarchy of scripts used for benchmarking on Linux machines. Each leaf node (CPU, Memory, Disk, wget) can be composed of multiple individual benchmarks.

work needed to be done to encapsulate this task.

However, even though we took great care in ensuring that individual resource components of servers are tested separately, they all are in use at any given time on a server. None of these benchmarking operations completely isolates each resource component (i.e., CPU, memory, disk and network are all in use simultaneously most of the time). However, each test is designed in such a way to minimize the effect of the non-benchmarked component (e.g., in a disk benchmark test, CPU and memory usage are minimized, while network traffic stemming from the test is eliminated). Minimizing non-benchmarked components in such a way is necessary and sufficient to allow our machine learning applications to distinguish resource allocations successfully based on their energy consumptions.

5.3.2 Disk Benchmarks

Our experimental setup for the disk I/O benchmarks is inspired by the fact that Big Data applications read and write large amounts of data from hard drives at any given time. Here too, altering the method in which the disk is accessed can be measured at the PDU level. For this experiment we implement four ways in which a file can be

read and written to a hard drive. Two metrics are of interest here, *i*) the amount of data transferred in a given time period; and *ii*) the energy consumption per data transfer. The four methods tested involve the C standard library functions *i*) *fgets* and *fputs* *ii*) *fgetc* and *fputc* *iii*) *read* and *write* with a large buffer size *iv*) *read* and *write* with a smaller buffer size.

The functions *fgets/fputs* and *fgetc/fputc* were chosen because they are the common ways to copy files in C. For all our experiments, we used the *gcc* compiler version 4.9.2 (Debian 4.9.2-10).[3] The *read/write* functions are the underlying system calls for disk operations. Figures 5.4, 5.5, 5.6, 5.7 are snippets of sample programs that copy data files from one location on a hard drive to another. The complete code can be found in the appendix. These programs can be used to profile energy consumption of disk operations in our setup.

```
while( ( ch = fgetc(source) ) != EOF )
   fputc(ch, target);
```

Figure 5.4: Sample code used to copy file with *fgetc/fputc*

Compiler options are routinely used to affect performance (throughput, speed, or memory footprint) of applications. We used the compiler options to determine whether there is an inherent benefit to common compiler options in terms of energy efficiency differences of the resulting executables. As such, we used two compile flags in our two experiments. First, no compiler options for optimization (i.e., -O0). Then

[3]https://en.wikipedia.org/wiki/GNU_Compiler_Collection

```
while( ( ch = fgets(line, SIZE-1,  source) ) != NULL ) {
   line[SIZE-1]='\0';
   fputs(line, target);
}
```

Figure 5.5: Sample code used to copy file with *fputs/fgets*

```
int SIZE = 20000;
while((n = read(f_in, line, SIZE))  != 0 ) {
   write(f_out, line, n);
}
```

Figure 5.6: Sample code used to copy file with *read/write* large buffer

```
int SIZE = 8000;
while((n = read(f_in, line, SIZE))  != 0 ) {
   write(f_out, line, n);
}
```

Figure 5.7: Sample code used to copy file with read/write small buffer

we repeated the experiment with optimization flags for speed and size turned on (i.e., -O3).

Compiler optimizations are useful to study as, currently, no optimization exists specifically to reduce the energy consumption of the resulting code. Energy consumption is, if at all, only a secondary concern to priorities, such as compilation time, speed of the resulting executable, and space/size of the resulting executable. Often it is assumed that energy consumption and execution time are directly linked. This is usually true, except in the case of using multiple hardware resources as well as when using heterogeneous hardware resources.

On the one hand, the optimization flag -O0 was chosen as it is the default flag which only aims at reducing compile time and inserting expected symbols to produce the expected output from debugging [50]. The optimization flag -O3, on the other hand, turns on "all optimizations specified by -O2" and 12 (twelve) further optimizations. [50] Note that -O2 includes all optimizations except those involving space-speed tradeoffs.

We chose four popular implementations mentioned earlier to read and write a 2GB file. The different ways to access the disk with different system/function calls and buffer sizes are *a) fgets/fputs*, *b) fgetc/fputc*, *c) read/write with buffer size set at 20,000 bytes d) read/write with buffer size set at 8,000 bytes*. We used each of these implementations to copy the 2GB test file as often as possible in a given time frame. A file of this size allows us to run each test for a sufficient amount of time to produce a measurable change in the energy monitoring data. Furthermore, this was also the largest file size that could be copied with all implementations.

Using Figure 5.8, we observe energy conserving benefits for particular implementations of disk I/O when the applications are run for fixed amounts of time. We ran each application for the same length of time, to copy data as often as possible. Figure 5.8 shows the kW usage of 30 sample points taken for each of the implementations during a fixed period of time. We ran each implementation for 30 minutes to copy the file as often as possible. Thus, the 30 sample points correspond to the energy usage averaged every minute of the 30 minute experiment. The red line indicates the median sample value, while the thick box covers data points from the 15^{th} to 85^{th}

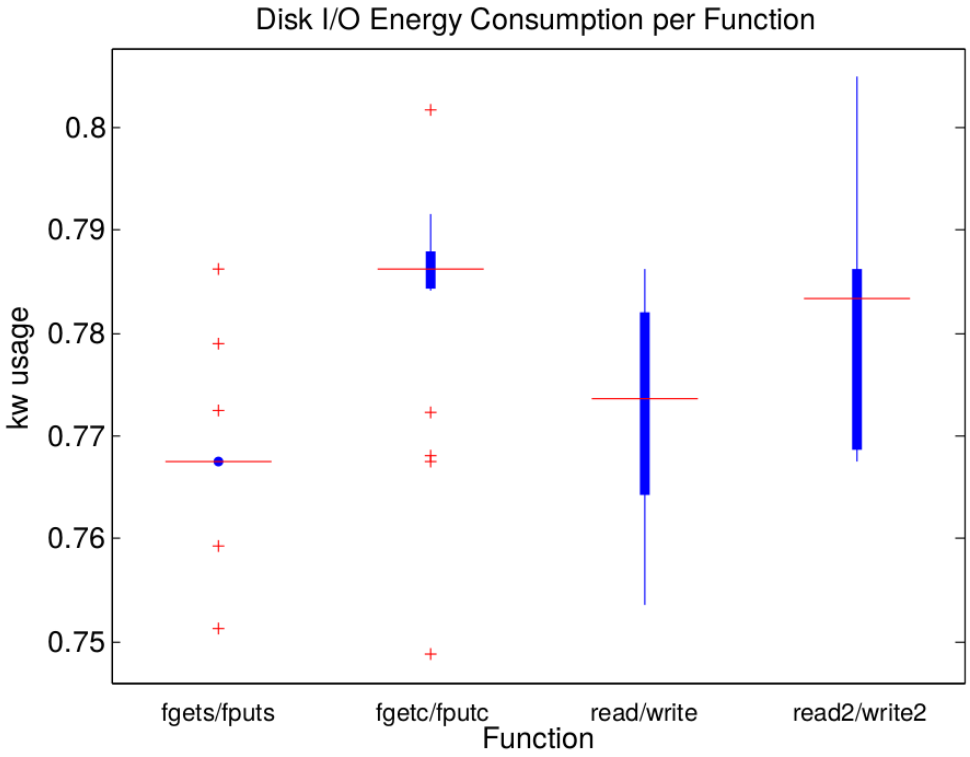

Figure 5.8: Energy consumption for different implementations of disk I/O for a fixed amount of time. No optimization flags were used at compile time to produce the executables. Each function was executed for a fixed amount of time, capturing data points as the benchmark completes.

percentile. The thin box covers the remainder of the data excluding any outliers (if they exist).

We observe that different implementations produce different energy usage patterns while using an identical number of CPU resources. Differences in energy consumption appear directly related to the underlying system calls. Using the tool *strace*[4] to capture system calls made by our applications as they access the disk, we determined that the only significant differences are parameters of system functions *write* and *read*, as well as the number of occurrences of these system calls. All our implementations resulted exclusively in system calls using *write* and *read*, however *fgets/fputs* and *fgetc/fputc* used the default 4,096 bytes as the buffer size to access the disk. System calls resulting of implementations using *read/write* used the buffer sizes we specified,

[4]https://en.wikipedia.org/wiki/Strace

namely 20,000 and 8,000 bytes (arbitrarily chosen to investigate their relative impact).

Both traditional user level function calls (i.e., *fputs /fgets* and *fputc/fgetc*) result in identical system calls. The number of these system calls varies between implementations which is the cause of the difference in the energy consumption profile. In both cases, the buffer size of each system call corresponds to the page size of this particular system (defined by operating system parameters). However, *read/write* functions at the user level are left unaltered by the compiler; the increased buffer size is maintained and results in fewer system calls.

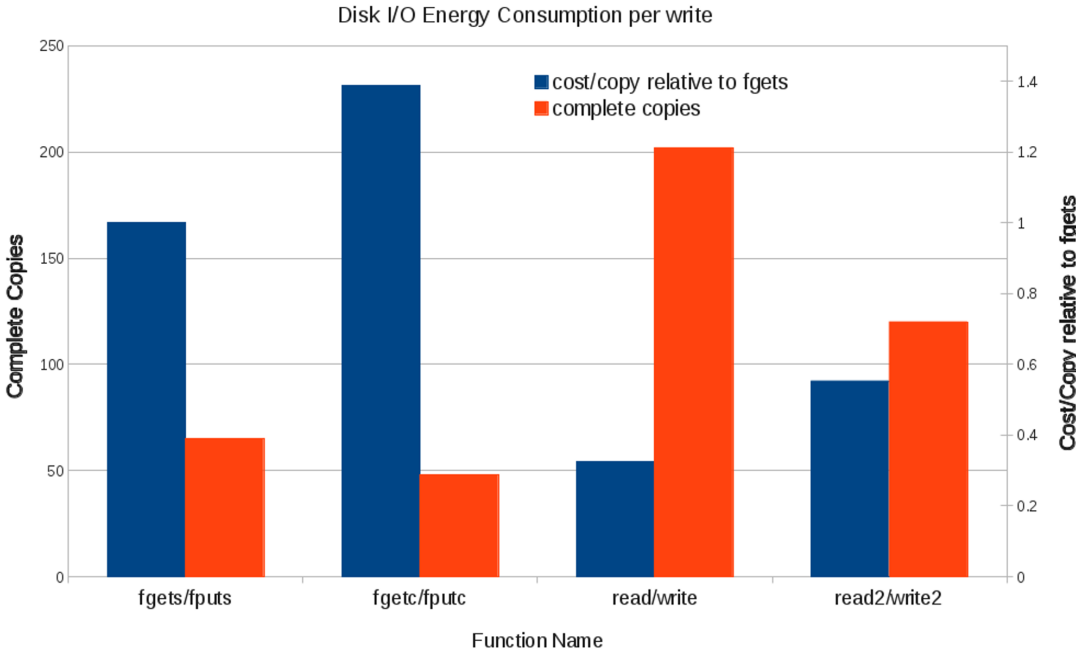

Figure 5.9: Complete copies per experiment and costs per *write* relative to *fgets/fputs*. *Read/Write* achieve more completed copies and are cheaper per *write*.

Figure 5.9 highlights the overall performance of each implementation by capturing the overall number of file copies that were achieved within a fixed amount of time. This figure also relates each implementation's performance to its energy cost per disk I/O operation.

At a high level, this figure indicates two things. First, looking at the orange (right) column of each pair, we see that both *read/write* pairs outperformed their

counterparts using traditional user level functions. Unsurprisingly, the function with the larger buffer size was able to copy more files than either of its competing implementations.

Second, the blue (left) columns of this figure indicate the total energy consumption of the implementation over a period of 30 minutes. We observe that the *read/write* pair used less energy in the same period of time than its competitors. This is likely due to the fact that fewer disk access operations were performed. Compiler options, including optimizations, do not take advantage of this readily available improvement in performance, and neither are potential energy optimizations carried out.

Yet, there is potential for further energy consumption optimizations of software components heavily relying on disk access. Specifically, looking at Figure 5.9 in detail, we see that when using *fgets/fputs* we copied the 2GB file 58 times in the allotted time frame. This figure serves as the baseline to assess other implementations.

Using *fgetc/fputc*, we completed fewer file copies in the same time frame and yet used more energy per complete file write than *fgets/fputs*. In other words, we used more energy and did less work with this implementation. This occurs despite the fact that both implementations use the exact same system calls to achieve the task. The frequency of these calls varies between implementations, thus resulting in different energy consumption profiles.

Using *read/write* functions with a relatively large buffer size of 20,000 bytes per I/O operation outperforms our standard implementation in terms of completed file copies by a factor of more than 300%. Its energy consumption per write, compared to the standard benchmark using an *fgets/fputs* implementation, is relatively low at only 30%. A similar result is observed for the same implementation (using *read/write*) modified to use a different buffer size of 8,000 bytes. Completed reads and writes of the file are lower than the implementation with a larger buffer size and at the same time the energy consumption per write is higher. This is due to the smaller buffer size and the resulting larger amount of required system calls to transfer a file of a given size.

The compiler does not differentiate between the implementations using *fgets/f-*

puts and *fgetc/fputc*. Both implementations result in similar executables producing identical system calls which affect disk I/O operations. However, our implementation using lower level functions *read/write* were left unaltered by the compiler and produced system calls with parameters that resulted in energy savings as well as performance increases.

5.3.3 Memory Benchmarks

Applications with heavy memory requirements, as the dominant resource, produce a different power profile than CPU-intensive or storage intensive applications. In this section we highlight two different applications which demand different amounts of memory. The impact of these applications from an energy consumption standpoint varies greatly. Similar to the theme in previous sections, here too, energy measurements are obtained from at the PDU level.

Figure 5.10 showcases the impact of a program that requires a maximum of approximately 10GB of memory. This program was constrained to use only one CPU. The power profile of this application demonstrates a comparatively stable consumption throughout the duration of the program's run—a mathematical function written in Python. It ran single threaded on one CPU for a duration of 18 minutes. Running this program for such a sustained time with relatively large memory requirements did, however, only produce a marginal increase in power consumption compared to CPU-intensive applications. The increase in current averages 0.2 ampere for the duration of the program's lifetime and is very stable. The resulting energy increase is very close to the energy consumption increase obtained from running one core at 100% with very little memory usage. Memory usage, when it is barely changing and the contents of memory are also kept relatively unchanged for the lifetime of the program, contributes minimally to overall energy consumption.

This is juxtaposed with the power profile shown in Figure 5.11. The application that generated this power profile is similar to the one of Figure 5.10. It also requires large amounts of memory as the main resource constraint (i.e., approximately 30GB of memory); additionally, the program was limited to two cores running at full utilization. The program ran for over 24 hours and had varying memory requirements throughout. Based on the observations from Section 5.3.4 the large spikes cannot be caused by running two cores at 100% utilization and, thus, must have resulted

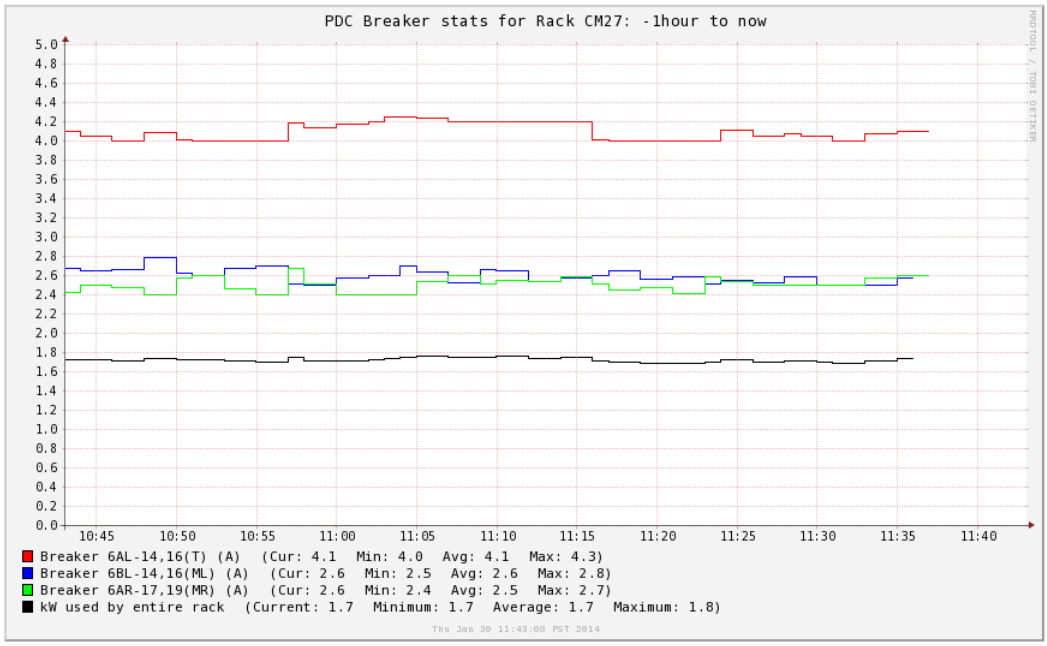

Figure 5.10: Power profile of a process dominated by memory resource utilization.

from differing memory requirements. Additionally, we find that energy consumption increases dramatically for memory usage when memory locations are changed or updated frequently, as is the case in Figure 5.11.

While this program shows a lot more variation in its power profile than the previous memory intensive application, it too exhibits long periods of stable higher than normal power consumption. This means, while there is seemingly unpredictable power consumption behavior in these high memory applications, they do lend themselves to adaptive scheduling and resource provisioning.

Through our benchmarks, we have established that memory intensive applications, such as CPU intensive applications, exhibit a visible power profile with observable changes in the server's power consumption. Requiring approximately 10GB of memory in our server configuration resulted in a marginal increase in power consumption (cf. Figure 5.10). This means that from a power consumption point of view, starting multiple processes with relatively small to medium memory footprints can be more energy friendly than booting up a new machine for such a process. We have also shown that the contribution of memory to the energy footprint depends heavily on

the frequency and quantity of updates to memory locations rather than purely on the size of allocated memory.

Moreover, we have shown that memory intensive processes produce periods of stable energy use if memory usage remains fixed (cf. Figure 5.10). This characteristic is observable even in the power profile of Figure 5.11 and lends itself to scheduling decisions. If the goal is to remain below a certain value for energy consumption, we propose that CPU or appropriate memory intensive applications can be run on the same server without exceeding desired energy consumption thresholds by using the power profiles as decision making guidelines. This relies on two of our confirmed findings: (i) power profiles are uniquely characterizable for CPU and memory intensive applications; (ii) there is a measurable relation between resource consumption and energy consumption; this relation can be profiled for a given hardware configuration and varies depending on the underlying hardware.

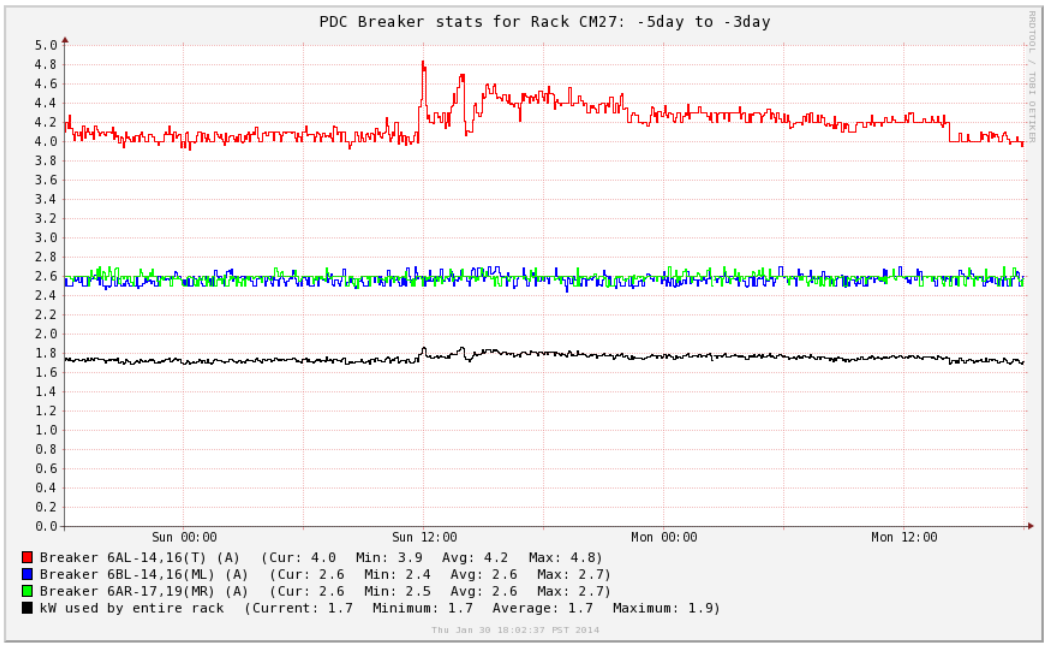

Figure 5.11: Power profile of a process dominated by large memory resource utilization. Memory utilization accounts for more energy consumption near the beginning of the benchmark when data is frequently loaded and changed in memory.

5.3.4 CPU Benchmarks

Our interest is in exploring the relationship between energy consumption and execution time as it relates to a multi-core system. Figure 5.12 illustrates the energy consumption relative to the server's idle state for the number of CPUs at full utilization. Two CPUs at 100% utilization effectively provide twice the execution time as a single CPU at 100%. The boxplot illustrates the result of 300 unique data points per CPU utilization test with the indicated number of cores at 100% utilization. The red line in the graph represents the median value of kW measurements at any given sample point. The boxplot captures 70% of the data in its core component and the remaining 30% in the graph's whiskers. Outliers are represented as a red plus.

Figure 5.12 quantifies the linear relationship between adding additional cores, and the increased available execution time, with the energy usage of a server. We observe a steady increase of energy consumption as cores are activated from 2 to 12 cores in this server. Adding cores in this range will result in a linear increase of energy consumption. However, this linear increase does not continue beyond the point where exactly half of the available cores are fully used. Once 12 cores are active, adding more cores does not incur the expected increase in energy consumption anymore. The growth in energy cost while adding additional work capacity is minor as can be seen when 12 to 20 cores are active.

The change in slope gradient of energy consumption, occurs exactly when the number of active cores is greater than half the number of overall available cores since in our system, we have two cores located on each CPU. This profiling work was automated, non-intrusive and utilized only existing monitoring infrastructure in the data center. Custom profiling, such as this, now allows us to draw conclusions about the available energy optimization strategies for this particular server and can be repeated easily for other servers in the a data center, thus adding to the utility of our approach.

Adding additional cores at 100% utilization, after the cost-to-energy breakpoint has been reached (e.g., at 12 cores), incurs minor additional costs to the operator of the data center. Therefore, one can establish an optimal cost-to-energy consumption zone for each server which is specific to each server's hardware. At this optimal cost-to-energy consumption zone, any additional utilization will only minimally increase cost or energy consumption (e.g., placing more processes on the same server will min-

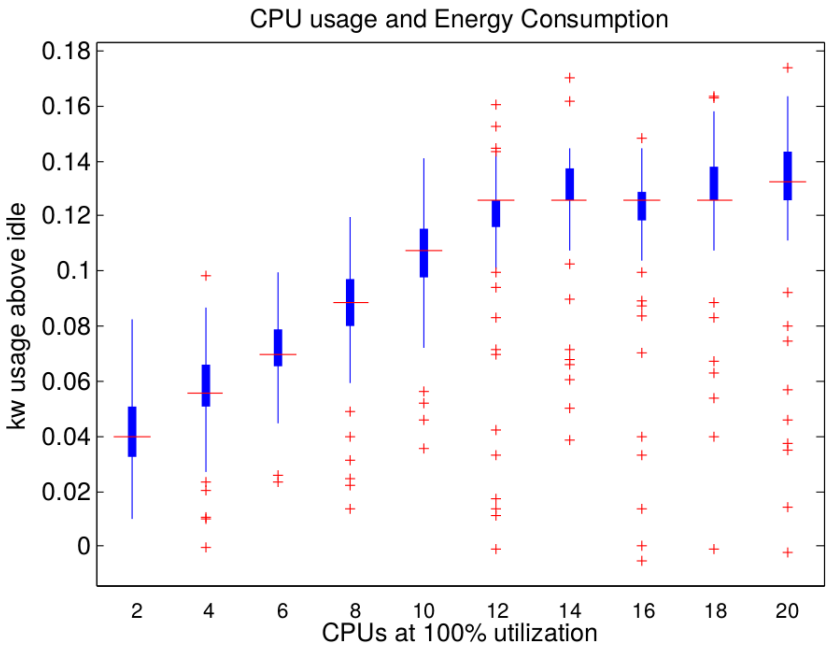

Figure 5.12: Energy consumption over 300 sample points for varying CPU utilization between 2 and 20 cores. Energy consumption increases linearly until half the cores are fully utilized. Each set of cores is utilized consistenly for a fixed amount of time.

imally increase costs while revenue may continue to increase linearly, as can be seen once more than 12 cores are active in Figure 5.12).

Our findings are based on data that was obtained by measuring the entire server system at once, thereby capturing every aspect of operation during each benchmark. This is a valuable approach as the alternative, obtaining measurements at isolated points in the system (e.g., just at the CPU or just at the server's fans), leaves out other components which are not instrumented, but might have adverse effects on energy consumption (e.g., consider increased fan activity as more cores are active). Instead our measurement encompasses the energy consumption of the entire server as it is measured at the PDU level of the server rack inside the data center. Hence, it is a true measure of the entire server's energy consumption over time.

Both, time and CPU utilization play important roles in optimization strategies for CPU intensive applications. Figure 5.13 aims to answer the question whether jobs with more resources take less time and, simultaneously, provide energy savings. To

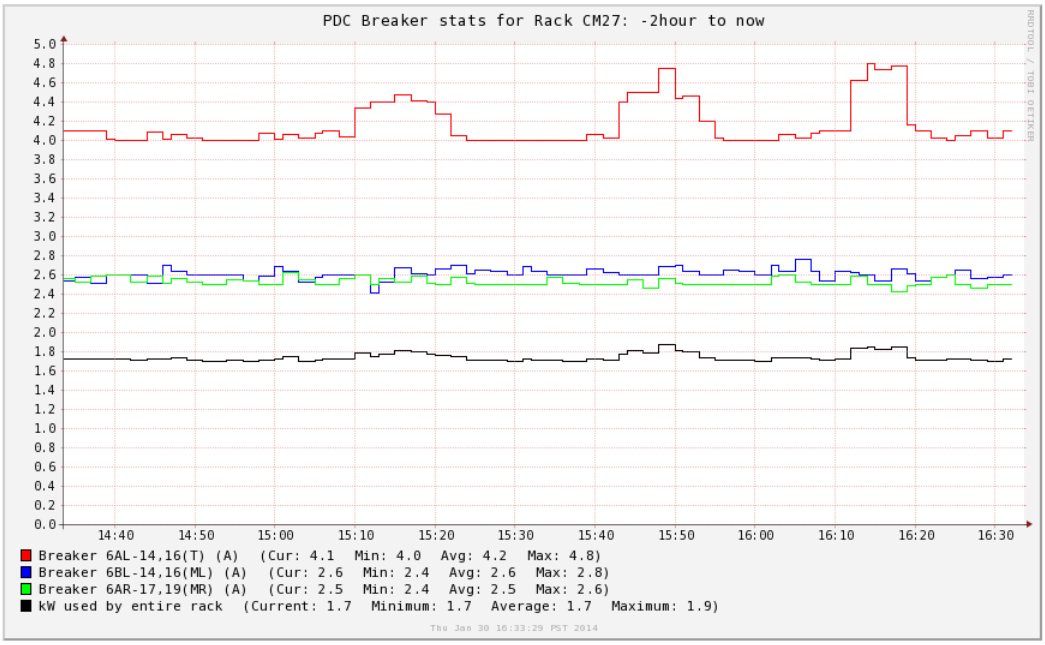

Figure 5.13: Power profile of CPU-intensive benchmarks running at eight cores with eight threads, eight cores with 16 threads, and 16 cores with 16 threads for the same workload.

answer this question, we first ran a CPU-intensive benchmark on eight cores with eight threads. Completion of this process took approximately 11 minutes and serves as the benchmark to which the remaining configurations are compared. During this time, the current draw increased by 0.3 amps. This was followed by an idle period to return the system back to a stable state. Following this stable state the number of cores for the identical workload was kept at eight threads, while the number of cores was increased to 16. This did result in a greater ampere spike and maximum increase. At the same time however, the duration for which the job ran was not affected significantly. Inspecting the last visible increase in the power profile, this rise corresponds to the same job being provisioned with 16 cores and 16 threads.

Figure 5.13 shows that when the number of resources is doubled and the total time is reduced by half, then the power demand remains the same. This is visible in the peaks of the first and third power profile peaks. The second power profile peak however, indicates that the increase in available computational resources must be managed smartly. In our use case, increasing the number of cores available to a process did neither speed up the computation, nor did it save energy. Quite to the

contrary, increasing the number of available resources in such a way that the software cannot utilize it efficiently—making 16 cores available to a process that only has eight threads did increase the total power consumption. This is visible not only in the top line of the graph, but also in the corresponding chart at the bottom which represents kW consumption of the entire server rack.

Using this finding, energy wasting can be avoided. Two separate pieces of knowledge need to be available to create energy savings effectively as a result of our benchmark: (i) knowledge about the job (i.e., eight threads, CPU-intensive); (ii) power profiles for CPU intensive jobs on a particular hardware. Additionally, our power profile in Figure 5.13 established a mapping between power consumption, available resources (i.e., 8 and 16 cores) and additional information about the process (i.e., number of threads). Thus, in this situation where a CPU intensive, eight thread process on 16 cores is run, power consumption can be improved without negatively impacting the performance of the job precisely because we have power profiles of our hardware/server configurations. To reduce power consumption, the allocated resources can be safely reduced to eight cores. This strategy maintains performance levels and improves power consumption.

To obtain all these findings, energy consumption measurements at the PDU level were sufficient. There is no need to use instrumentation within a server, which would provide only a fraction of the relevant information and introduce additional, compounding sources of errors. The PDU provides a complete view of the energy consumption of CPU dominant software applications.

Our CPU benchmark is designed to identify not only the characteristics of CPU intensive applications on a server by server basis. Our benchmark can also provide insights into possible modifications that can be applied at the software applications level. Tackling code modifications at the software applications' level allows us to take advantage of certain hardware characteristics at run time that may increase performance and energy consumption.

5.3.5 Network Benchmarks with Limitations

We envisoned networking benchmarks for our research to provide a complete view of all available resources. However, as often is the case in experimental setups, limitations are encountered that allow for only limited data collection. The University of Victoria enforces packet/traffic shaping rules that apply to all network traffic in and out of the university [16].

Network bandwidth is a finite resource, only allowing a maximum fixed amount of data to be transferred at any given time. Traffic shaping, in most cases is used prudently to allow research and business activities to proceed while limiting unrelated activities such as streaming movies. In the case of UVic for example, that means that email traffic will, in ideal circumstances, never be limited by too many people watching Netflix.

Unfortunately, for us this means that the network traffic we use to send data between nodes and across campus cannot be used to effect a large enough and measurable change in energy consumption on the servers we can readily access. The servers we have access to are located within the UVic network and are, thus, subject to its network traffic rules.

We do expect to see similar differentiating characteristics on different servers as we observed in the other resources we were able to measure. Network traffic, when isolated does not rely heavily on CPUs. Furthermore, a plethora of network cards and hardware exists with different performance characteristics and most likely different energy characteristics. While we were not able to measure this effectively, we are confident that our hypotheses carry through to network I/O. Depending on the software application, network traffic is closely tied to memory and/or storage. As such, a combination of those resource and energy usage characteristics may come into play. In the meantime, more research is needed, in a setup where this can be explored fully.

5.4 Discussion

The long term goal of this research is to investigate innovative methods to modify program execution and resource scheduling of applications dynamically within digital ecosystems to maintain or increase performance while at the same time optimize energy usage. As it stands today, we have to be aware of—and acknowledge—internal and external threats to validity of our experiments and the conclusions we draw from them.

One limitation related to this long term goal stems from the fact that we only have a limited number of different types of severs available in two different server racks. This definitely limits our ability to demonstrate the scalability of our approach for a realistic, large scale data center.

Furthermore, our measurements are limited to the instrumentation provided by the data center. While this does not necessarily impact the quality of metrics, it does impact the frequency with which individual data points are recorded. In the future, a more frequent measurement interval, below the current one minute mark, may be desirable.

Moreover, we are also limited in the hardware resource types which we profile, namely CPU, memory, and storage on servers. This is due to our reliance of existing state of the art monitoring equipment available in the UVic data center.

5.5 Chapter Summary

This chapter highlighted our experimental approach to investigate the relationship between performance characteristics, resource usage, and energy consumption of software applications in modern data centers. We have developed and deployed various techniques in order to benchmark and discern the relationships between performance and energy consumption. We have done so spanning multiple servers, and multiple hardware components within these servers. In this chapter, we also discussed the specific contributions that we make to the field of green computing. We published excerpts of our experimental work and descriptions previously [17, 19] though extensions and further work have been added in this dissertation. From our experiments,

it is relatively clear that the PDU of a server rack can indeed serve as a source of contextual energy consumption data for certain use cases. We have demonstrated that there is an initial indication that even the particular resource usage and resource type can be identified through measurements at the PDU level. Furthermore, we have demonstrated that our adaptive model has multiple data points available from our measurements and preprocessing of data on which scheduling decisions for software applications can be made. The appendix of this document contains source code and scripts that we created and used in order to perform this research. They should serve anyone interested in replicating or continuing this work further.

Chapter 5 also provide insights into the methods used throughout this work. We explain the data collection, experiment setup and execution as well as provide detailed software samples used to profile the servers available to us. The information provided, along with the source code of the benchmarking and orchestration scripts, serves as a foundation for interested readers to replicate our experiments on different server hardware. Moreover, we connect our experimental work with the machine learning approaches, which together form the basis of our dynamic and adaptive model to reduce energy consumption of software applications in data centers.

Chapter 6

Effects on Energy Consumption

This chapter presents two closely related and interconnected aspects of our research. First, we discusss the energy and performance data we obtained through our experiments. By doing so, we can highlight the connection between performance, resource usage and energy consumption. This allows us to demonstrate how one would use the available information and benchmarking information obtained from our research to schedule software applications more optimally in accordance with performance and energy consumption. Second, we introduce the reader to our work of automated data analysis using machine learning. We employ machine learning techniques to identify and classify software applications' resource usage based solely on their energy consumption patterns. These patterns are hardware specific and create a form of lookup database, which can help in software deployment decision making.

6.1 Energy versus Performance—An Overview

For our experiments, we have access to only two different server racks with two different server types. However, even though the server types are nominally identical (i.e., same architecture, disks, memory provisioning), measurable differences in their energy consumption have been identified. These measurable differences lead to an interesting refinement in our research approach. Not only do servers with different hardware components have different energy consumption patterns for identical software, but also servers with identical hardware have measurably different energy patterns (variations exist due to manufacturing processes). As mundane as this may

appear, it is actually quite significant. Generally, identical hardware is modeled as having identical energy consumption, yet we demonstrate that this is clearly not the case.

More specifically, for our experiments this means that any migration of software jobs between the different servers will inevitably induce noticeable changes in energy consumption as well as overall performance. By using the existing energy monitoring setup provided by UVic EDC2 and our custom benchmarks developed in the course of this research, we are able to quantify these differences through actual data.

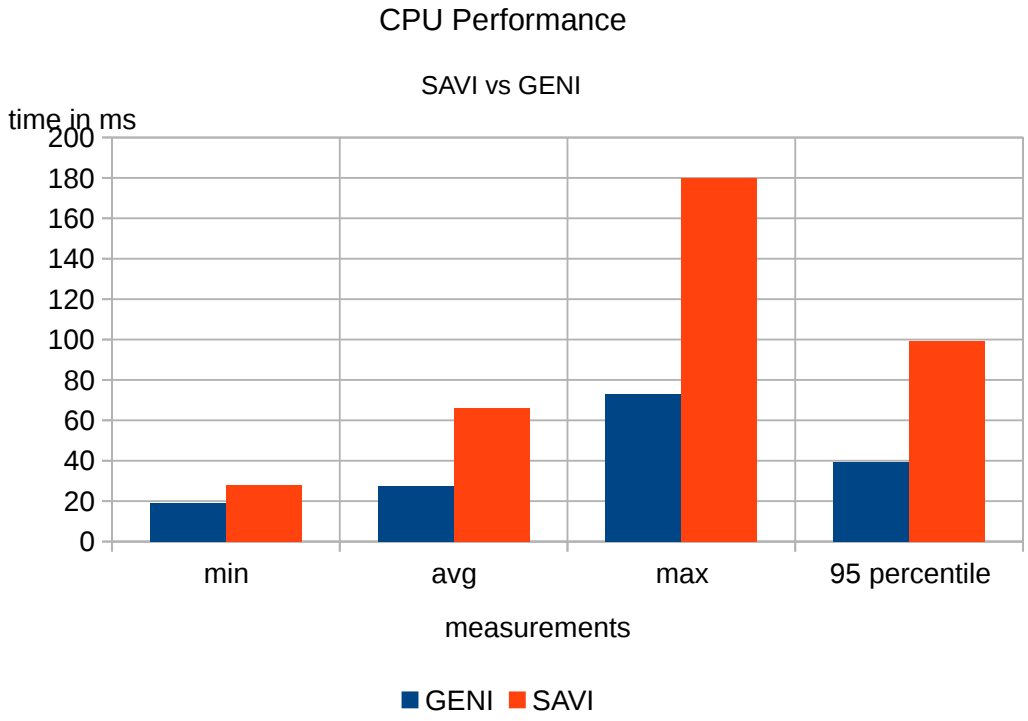

Figure 6.1: Performance of CPU-intensive reference job on SAVI versus GENI servers. A less performant server in terms of job duration (SAVI), which consistently consumes less energy for the same job.

Figure 6.1 illustrates the performance comparison when migrating CPU-intensive jobs from our servers assigned to the GENI cluster to the SAVI cluster. We compared the servers located on the SAVI rack with those located on the GENI rack. The SAVI server requires less energy for a given job than the reference GENI server

in Figure 6.1, however its performance is slower and, thus, a job takes more time.

More specifically in Figure 6.1, we can see that the performance (i.e., the time it takes for a given CPU-intensive reference job to complete) is shorter and, therefore, better on the GENI cluster. On the GENI cluster running our CPU-heavy benchmark with ten threads, limited to ten specific cores on the CPU took `27.5` seconds to complete. This included completing `10,000` request events in parallel. We observe that the distribution of the runs is much more tightly clustered around the mean on the GENI cluster as well. Albeit not shown in this figure, the immediate energy consumption is lower and, thus, better on the SAVI cluster. The SAVI cluster uses less energy at any given moment because less work is performed. This is due to the fact that the SAVI cluster is only able to run on one core, despite mapping eight CPUs to its hardware. Overall, the time it takes to complete the job on SAVI is much longer. On SAVI the benchmark completes in a total of 658 seconds, while on GENI, in comparison, the entire test suite completes in 275 seconds. An interesting question arises when one takes into account the combination of performance and energy consumption in order to determine which configuration is better for any given situation (i.e., it is feasible to consider that increased energy consumption is preferable over shorter execution times in some cases). Migrating to a slower server can in fact be useful, especially when it saves energy. Consider the case where a software job is run over night, starting at midnight. If the output of this job is not consumed until the morning, when the workers/analysts return, then there is no benefit if this job finishes within one hour and consumes a lot of energy compared to finishing within six hours and consumes overall less energy. Context specific trade offs can potentially become a viable scheduling and deployment strategy. Figure 6.2 provides the complete set of metrics for this benchmark comparing the SAVI and GENI servers.

6.2 Scheduling Strategies for Energy Consumption

Based on our findings regarding energy consumption and performance of the available resource usage categories, we are able to assist in the scheduling decision making. Figure 6.3 helps in illustrating the underlying logic of this process. In Figure 6.3 the performance is displayed on the horizontal X axis. The further right a data point is, the greater is the performance output of the underlying server. As more work is

```
Run for Geni and SAVI
GENI
Maximum prime number checked in CPU test: 90000
est execution summary:
    total time:                          27.5506s
    total number of events:              10000
    total time taken by event execution: 275.3595
    per-request statistics:
         min:                            19.12ms
         avg:                            27.54ms
         max:                            72.79ms
         approx.  95 percentile:         39.13ms
Threads fairness:
    events (avg/stddev):           1000.0000/45.45
    execution time (avg/stddev):   27.5359/0.01
SAVI
Maximum prime number checked in CPU test: 90000
est execution summary:
    total time:                          65.8791s
    total number of events:              10000
    total time taken by event execution: 658.3961
    per-request statistics:
        min:                             28.13ms
        avg:                             65.84ms
        max:                             179.85ms
        approx.  95 percentile:          99.20ms
Threads fairness:
    events (avg/stddev):           1000.0000/206.64
    execution time (avg/stddev):   65.8396/0.03
```

Figure 6.2: Detailed statistics gathered running experiments to compare the performance of the SAVI and GENI servers

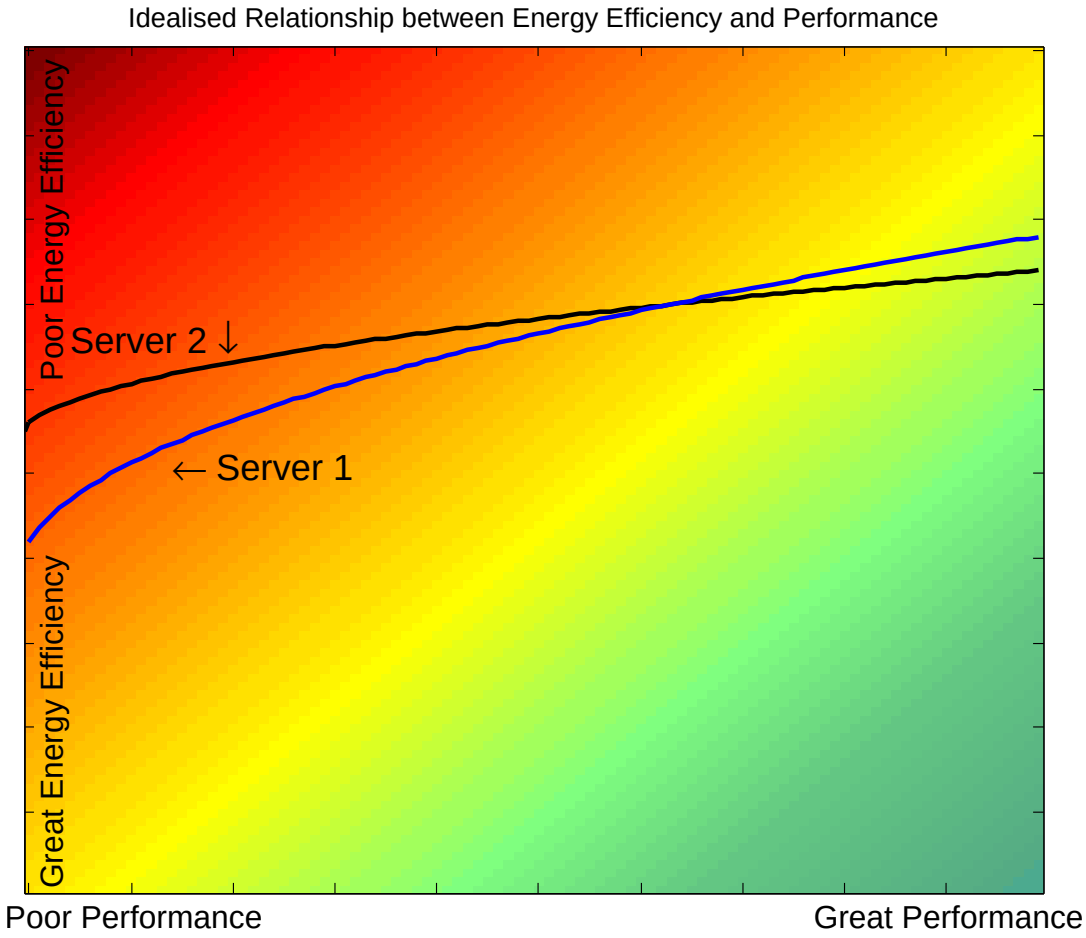

Figure 6.3: Idealized illustration of the relation between energy consumption and performance—while abstracted, the illustration is based on our actual findings

done on a particular server, the greater the performance. Energy usage is marked on the vertical axis. The more energy a server uses, the further up in the figure the data point will be. Combining these two factors we can identify, in an idealized hypothetical scenario involving two servers, which server is the more optimal machine to schedule work on. The ideal point is in the bottom right corner with maximum performance and zero energy consumption, which is impossible to achieve. The idealized scenario highlighted in Figure 6.3 is based closer on actual observations in which, generally, more performance is associated with more energy consumption.

Server 1, in Figure 6.3 uses a certain base amount of energy even when it is idle—thus, the performance of overall work done is low. Conversely, Server 2 uses slightly

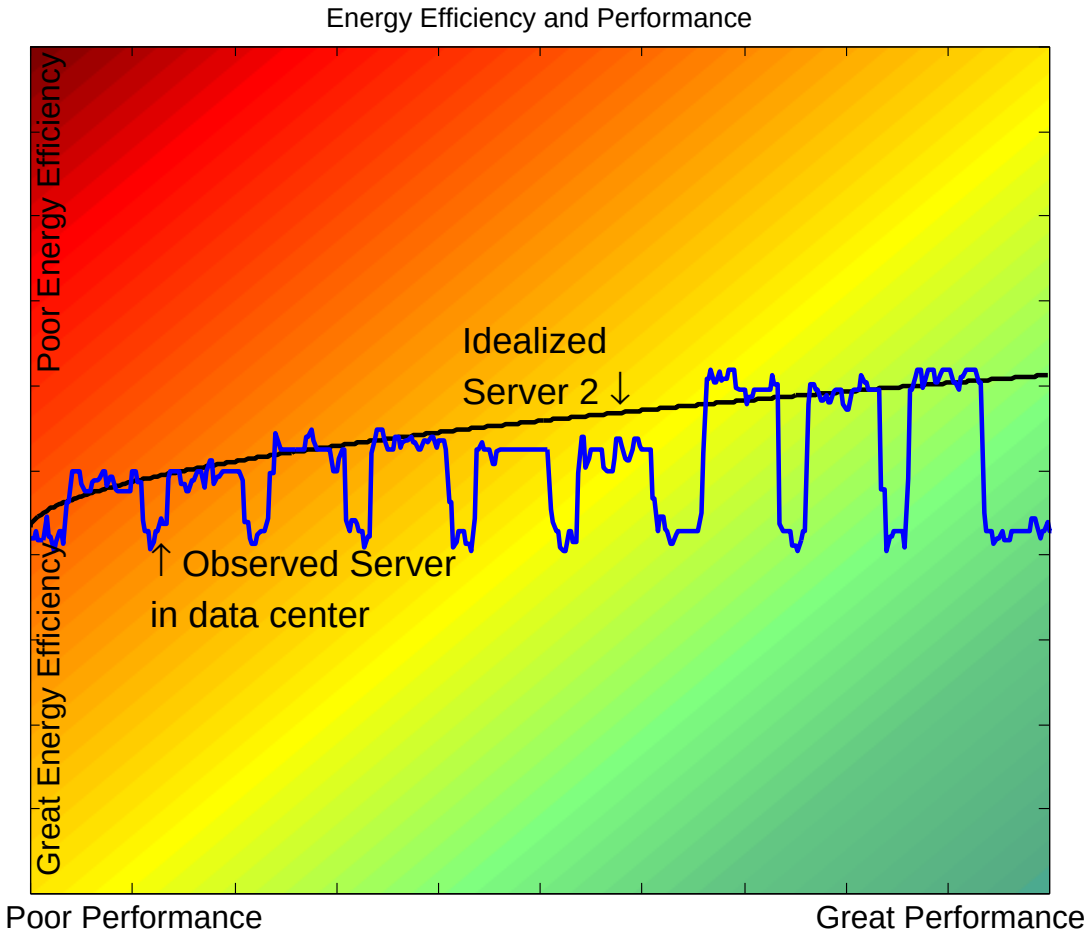

Figure 6.4: Idealised illustration of relation between energy consumption and performance overlayed with actual measurements in data center using server *GeniRack02*. We overlay actual measured timeseries results with the idealized curve.

more energy when idle. This discrepancy in idle energy consumption has been illustrated before by our own work and that of others [17,119]. Moreover, in data centers, 70 percent of servers are typically in an idle state at any given time according to Liu et al. [85]. Using our idealized illustration scenario, when the load is increased on these idealized servers, Figure 6.3 illustrates that not all servers behave the same in terms of provided performance and energy consumption. While server 1 starts out at lower energy consumption at idle than server 2, with increased load the energy usage per unit of performance tips in favour of server 2. Using this notion as a basis, we can identify ideal regions in this figure where both criteria, performance and energy usage, are relatively optimal compared to other available servers (i.e., in the case of

Figure 6.3 there are only 2 servers to choose from).

Figure 6.3 represents the idealized scenario. Figure 6.4 overlays the idealized plot with the actual measurements obtained from one of our servers in the form of a time series trace. As we traverse the figure to the right, we notice that as the workload increases, so does the energy consumption. The drops in energy consumption represent times when the system was allowed to return to the idle state between experiments, they have been included in this figure to illustrate that the system does indeed return to an idle state and the energy consumption normalized during those periods. This is important to ensure that a rise in energy consumption over time is not a result of lingering side effects, such as increased heat and cooling mechanisms as the experiment progresses.

We also observe that as the workload increases and performance—measured as work done—increases, the energy consumption does not always increase at the same rate. This is noticeable as more CPUs are brought online and confirms our earlier findings. Ultimately, the data peaks in the *observed server* in our data center is of interest as we can see that it follows roughly an idealized curve where increasing performance usually, but not always leads to increased energy consumption.

6.2.1 Dynamic Strategies and Scheduling Policies

Figure 6.5 illustrates the available data which can be used to provide an accurate point of view of the different server types at any point in their workload state correlated to their energy efficiency.

To support the decision making process on where a particular software application should run, we need to take multiple facets into account. Figure 6.5 showcases the multi-dimensional, multi-factor data pool that is available for the decision making process. The figure distinguishes between the available server types and their anticipated as well as measured energy-to-performance ratios. Since our model is based upon identifying the primary resource type usage of a software application, an application will be classified as one of the four resource types visible in Figure 6.5. Further this association occurs for each available server (in our case: two types of servers, but in real data centers there are many more). In order to locate the ideal placement for the software application, we aim to find a server where both, performance and energy

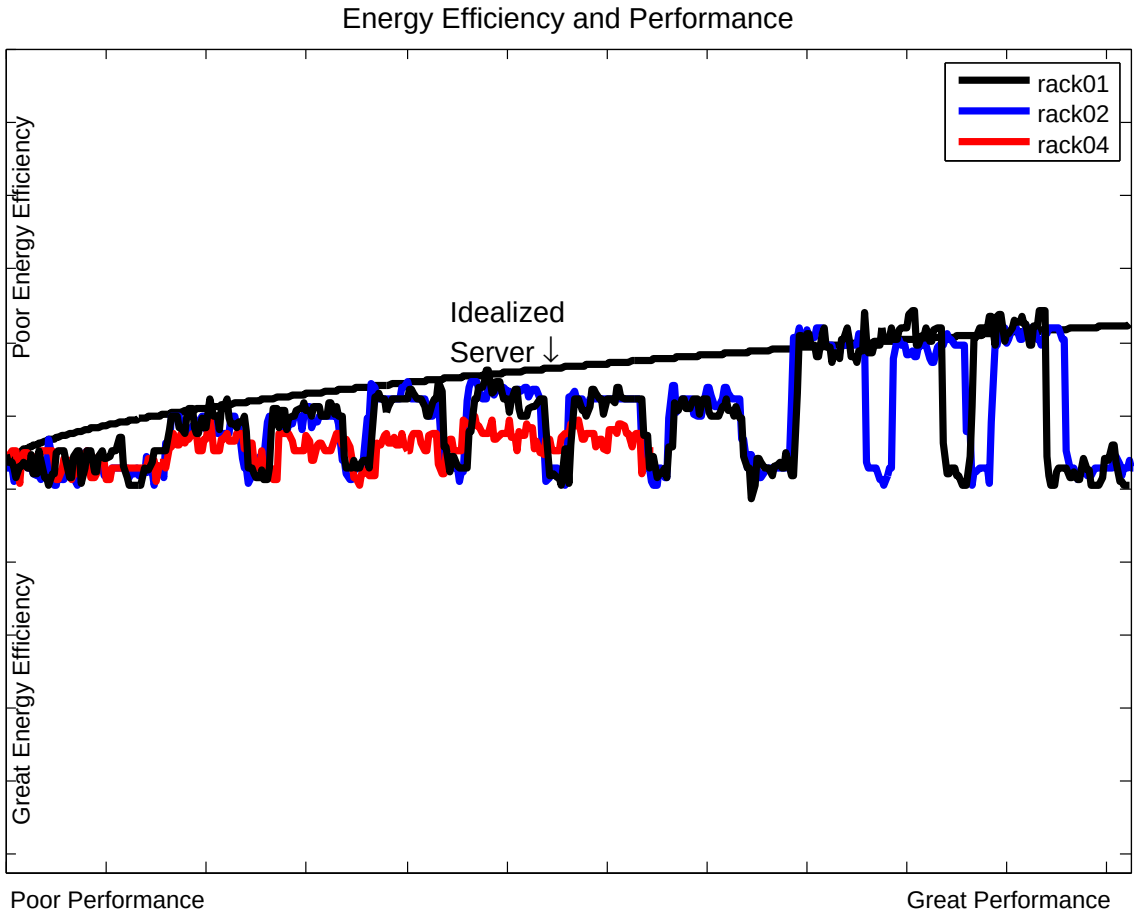

Figure 6.5: *GeniRack01, GeniRack02 and GeniRack04* CPU performance relative to energy consumption. We overlay actual measured timeseries results with the idealized curve.

savings, criteria are satisfied.

Figure 6.6 illustrates the different levels of performance and energy consumption available on our servers when considering CPUs. In this figure, we see data for three servers, namely `CPU226`, `CPU227` and `CPU228`. We measured the energy and performance of each of these machines in an isolated setting and combined their data in this figure for illustrative purposes. The red lines on the graph represent a desired level of performance as well as energy consumption. This creates two interpretations of the bottom right and bottom left quadrant that is formed by the intersecting red lines. On the one hand, the intersection of the red lines marks the point where the minimum performance and the maximum allowable energy consumption are reached.

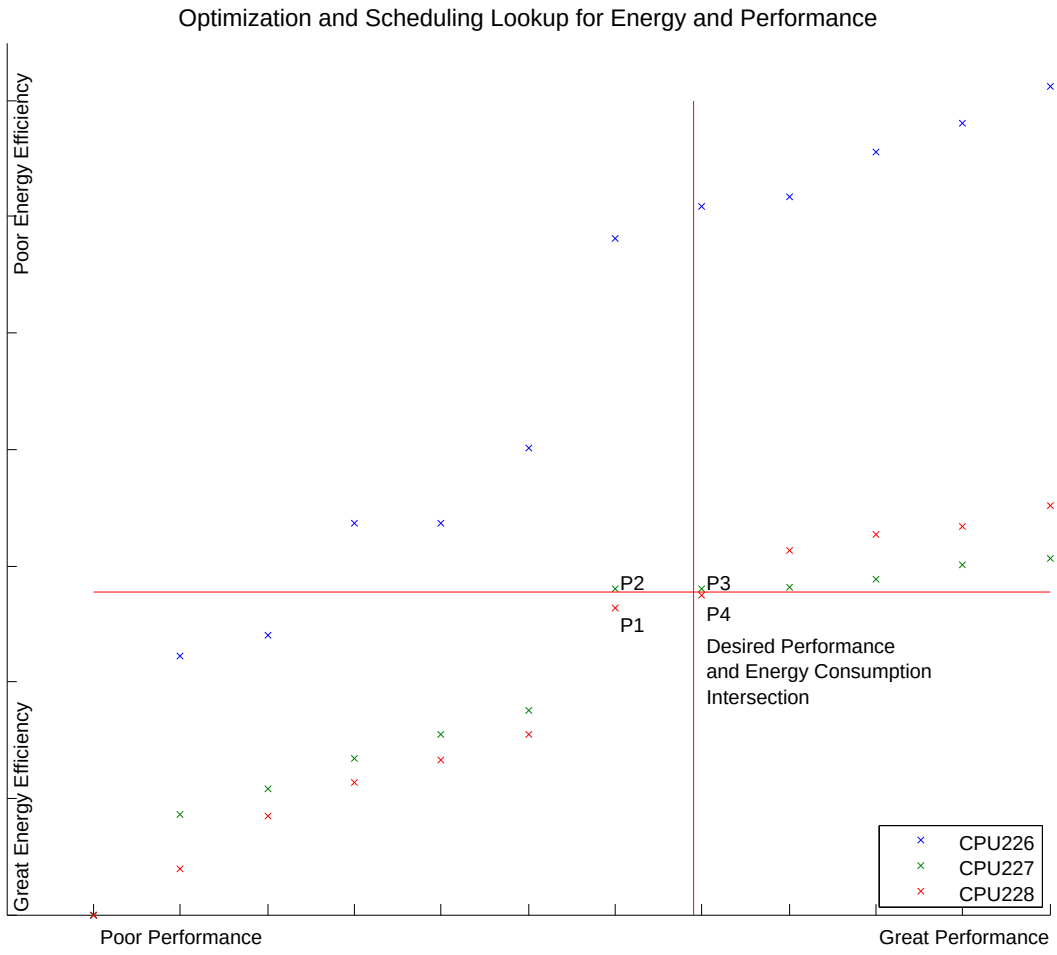

Figure 6.6: Comparison of CPU levels of multiple servers relative to multiple servers. The south-east corner of the plot is the ideal job placement as it maximizes performance and minimizes energy consumption.

On the other hand, the intersection of the red lined marks the point where the maximum performance is reached that a customer is willing to pay for and, still, the maximum allowable energy consumption. A server which meets the requirements for this particular job and service level requirements will be in the bottom right quadrant if we specify a minimum acceptable performance. Alternatively, a server will be in the bottom left quadrant if we specify a maximum amount of performance for which a customer is willing to pay for (i.e., more compute power comes at a higher cost). The points `P1, P2, P3` and `P4` are all in proximity of the intersection of energy consumption and performance specified by this example. The intersection is chosen for illustrative purposes and represents real values in terms of CPU performance com-

bined with an energy requirement. We can see that `P1` and `P4` meet the energy requirement, because they are below the horizontal line. `P3` and `P4` both meet the performance requirement. However, only `P4` meets the minimum performance and energy requirement. This means that `CPU228` at `P4` provides the desired minimum CPU performance and, at the same time, uses less energy than the maximum allowable amount. If, however, the customer is only willing to pay up to this level of performance then `P1` would be chosen on `CPU228`. Even though the points `P1, P2, P3` and `P4` are in relative proximity to each other, which server and load is ultimately selected depends on the customers' criteria for selecting performance and energy consumption.

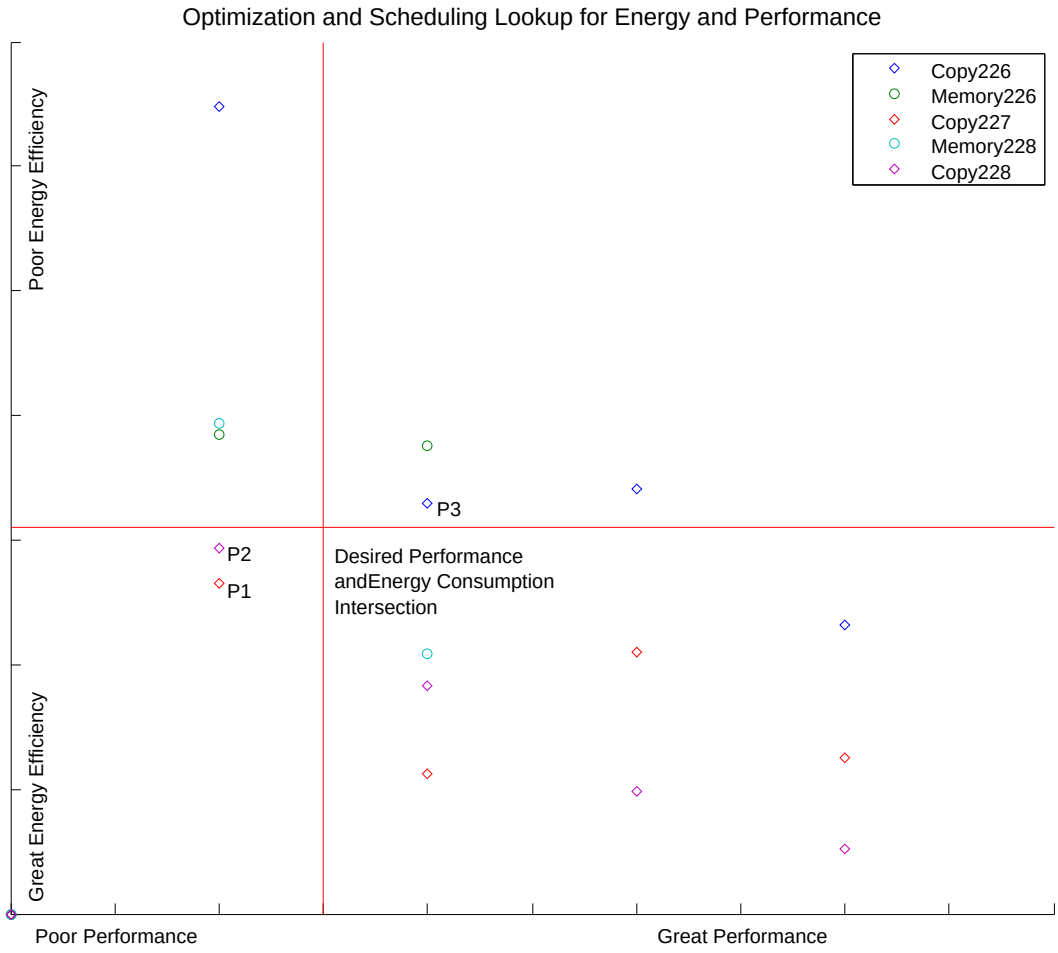

Figure 6.7: Comparison memory and disk bandwidth levels of multiple servers relative to their energy consumption. The south-east corner of the plot is the ideal job placement as it maximizes performance and minimizes energy consumption.

Figure 6.7 illustrates the different levels of performance and energy consumption available on our servers. For this figure, we have chosen to combine memory and disk operations as these two are often related. As in Figure 6.6 the red lines represent the intersection of two desired service levels pertaining to performance and energy consumption. The intersection of the red lines marks the point where the acceptable number of disk I/O operations is performed while not exceeding the maximum allowable energy consumption for this task. The points `P1, P2` and `P3` are all in proximity of the intersection of energy consumption and performance for disk I/O. The intersection is chosen for illustrative purposes and represents real values in terms of I/O operations per second and data transfer requirements, all combined with an energy requirement. We can see that `P1` and `P2` meet the energy requirement, because they are below the horizontal line. `P3` meets the performance requirement if we consider the vertical red line to mark the minimum acceptable performance. In this case the customer would have to make one of two choices. One, abandoning the energy cap requirement as `P3` is above the maximum specified energy consumption. Two, the customer can consider accepting less performance in order to stay below the energy cap. In this case the closest points would become `P1` and `P2`. With the different interpretation of the vertical red line, now as the maximum performance a customer is willing to pay for, the points `P1` and `P2`. are available. `P1` and `P2` have the same performance. Since `P1` uses less energy than `P2,` our system would select `P1` as it is the better fit. Depending on the interpretation and presentation of the red intersecting lines in terms of performance only, either the data center operator, the customer, or a dynamic scheduling utility would be in a position where one of the two criteria (i.e., performance or energy consumption) have to be compromised. If energy consumption is a flexible criterion in this scenario, we could choose `P3` as the machine to run our disk I/O intensive job. Yet, if energy savings are more important than performance we could now choose between either `P1` and `P2`.

We can see from Figure 6.8 that for a given performance requirement (e.g., CPU, disk or memory), we can identify multiple servers which satisfy both performance and energy efficiency concerns. This figure contains data points for CPU, memory and disk to illustrate the multitude of selections available. This figure demonstrates the varied information regarding performance and energy consumption that is available from even just a small number of servers. If needed, in a multi tenant scenario, an application can be moved between any of the servers identified as satisfying the re-

quirements in performance, energy consumption or both. Furthermore, as the load on these servers changes, some machines may be added, while others may be removed from the pool of available machines for further allocations.

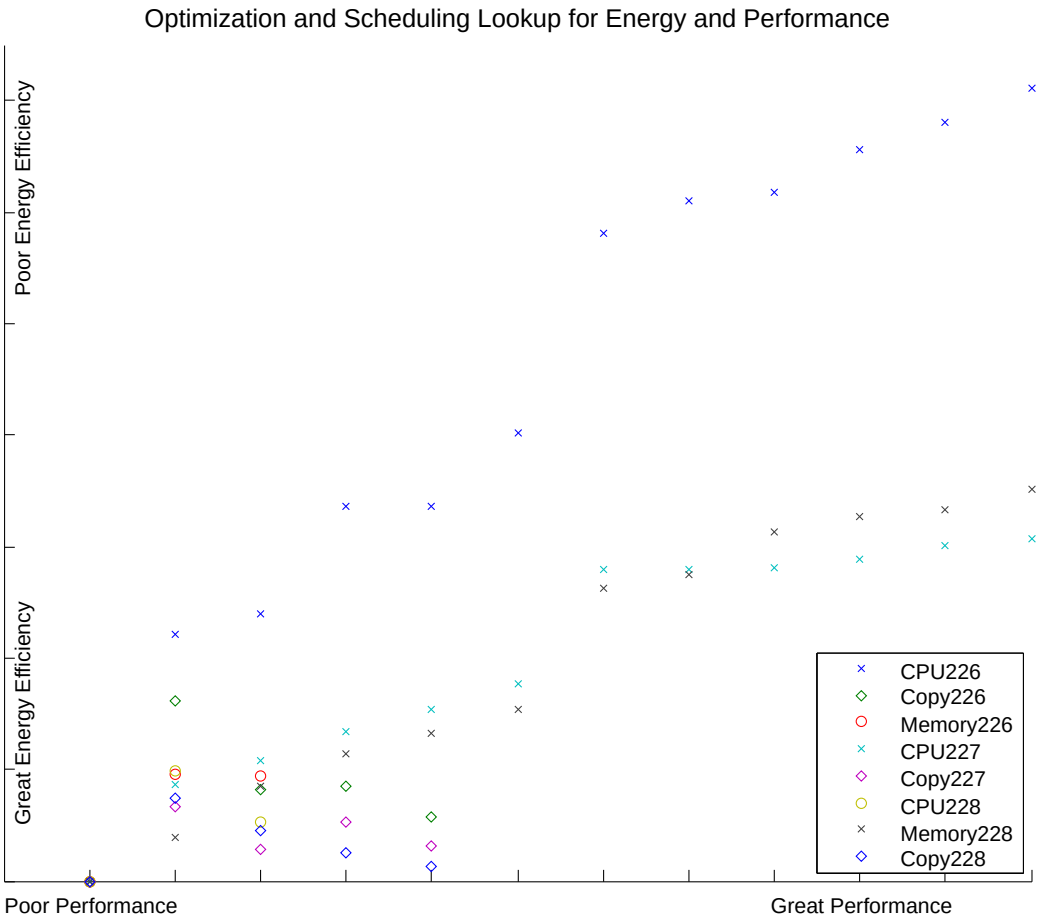

Figure 6.8: Comparison of CPU, memory and disk bandwidth levels of multiple servers. The south-east corner of the plot is the ideal job placement as it maximizes performance and minimizes energy consumption.

With additional server types available, more measured data points would be available, consequently, the data between the two points is interpolated to provide an estimate indication of resource and energy utilization at levels that falls between the measured benchmark data points (e.g., energy consumption of 1.5 fully utilized CPUs on Server 1 can only be estimated as data points only exist for 1 and 2 fully utilized CPUs, not 1.5). Figure 6.8 shows the available servers and their underlying energy and performance characteristics. Using this knowledge of how a server's energy de-

mand will vary when resource usage changes, allows a scheduling system to determine on which server a software application should run on based on the priority of the criteria which are: performance of the dominant resource type and/or energy efficiency.

For example, an application that solely prioritizes performance in CPU would pick the GENI server (if resources are still available), as we can see from Figure 6.1. However, an application that deems the SAVI performance metric to be acceptable and prioritizes energy efficiency would select SAVI servers. Accepting that these two servers can function as extremes on their respective spectrum, we can further argue that other server types provide measured data points between the SAVI and GENI data points. This scenario allows scheduling decisions to act with more flexibility and performs better regarding a best fit for performance and energy efficiency when deciding which server to use for a given software application.

6.3 Machine Learning and Energy Consumption

Knowing the energy consumption of a software application, as shown in Section 6.2, is a great position to be in. However, to achieve this and the ability to associate energy consumption with the resource use of a piece of software is challenging. It requires either monitoring access to the operating system (OS) running on a server, or, as in our case, access to the energy consumption data of the server. Previously, we demonstrated that we are able to identify the underlying resource type of several software applications purely by investigating its energy consumption patterns. [19] To classify software applications based on their energy consumption, we employ machine learning techniques. Multiple machine learning types exist. For our experiments we focused on two different types: Support Vector Machines (SVM) and Neural Networks (NN). This decision was based on the suitability and popularity of these two methods at the current time.

Ultimately, NN was chosen over SVM. SVM relies, in its most naive form on data that is separable. In other words, each SVM must be able to classify the data into two distinct classes. Since we required multiple classes this required us to use multiple passes since our SVM were not able to deal efficiently with the large number of different classes simultaneously. Consequently, NN were selected, even though SVM,

NN and possibly other methods would theoretically yield the same accuracy.

There is a common path for pattern recognition which follows the structure laid out in Figure 3.9. The first stage is preprocessing the data. This includes acquisition of a signal, thresholding of the output and separating data useful for classification from the noise (filtering).

The second step in this process is the feature extraction scheme, which is meant to determine a feature vector from a regular vector. A feature is a distinctive or characteristic measurement, transform, structural component extracted from a segment of a pattern. Statistical characteristics and syntactic descriptions are the two major subdivisions of the conventional feature extraction modalities. A good feature extraction scheme is meant to choose the features or information, which is most important for the classification. The final stage is signal classification. This step in Figure 6.9 can be solved by a variety of approaches: *linear analysis, nonlinear analysis, adaptive algorithms, clustering* and *neural networks.*
‘

This Section 6.3 presents a short review of mathematical methods employed in our adaptive system for classification of a process' resource utilization based on its energy consumption pattern. We published a paper on these results the proceedings of SEAMS 2015 [19].

The most important step for the classification task is extracting a suitable set of features that has the capability to differentiate among different classes. Classes are the different items or objects one wishes to differentiate between. In our case, the classes are represented by the exact resource amount used when a given energy consumption is measured (e.g., energy consumption X maps to a specific number of CPUs being active, a memory intensive job, a network or disk I/O job). As discussed in the introduction, statistical analysis is a way to generate such representations. The method for feature extraction consists of decomposing the server's energy and power consumption data obtained while our experiments are run.

Then, the obtained coefficients are used to construct a matrix of dimension $N \times K$, where K is the number of data points and N is the number of coefficients for each data point. The features (columns) are treated individually, dependent on their ca-

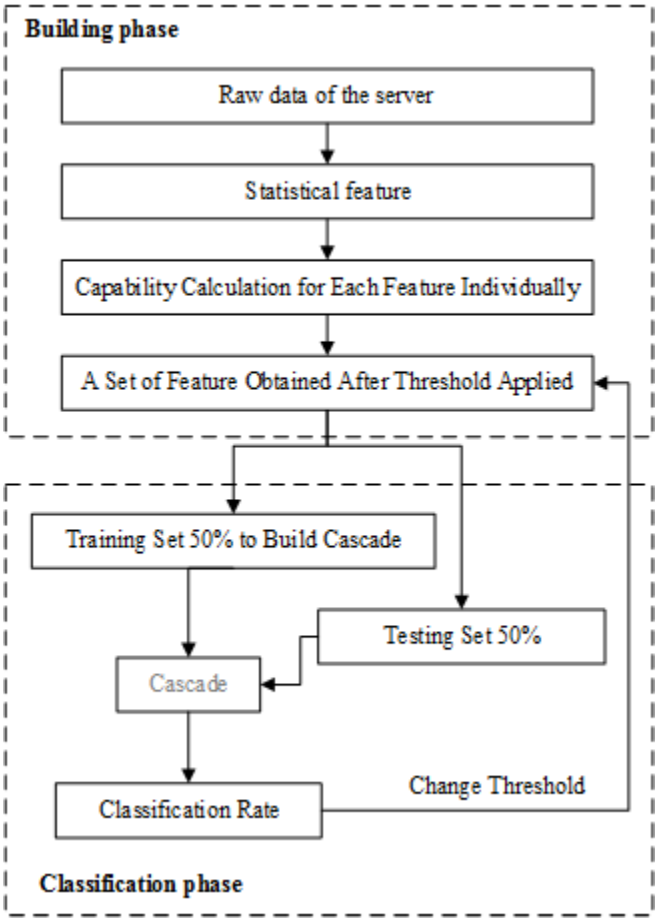

Figure 6.9: Classification System

pability to separate different classes. We refer to this capability as C_T. Subsequently, the detected features are ranked in descending order according to the value of C_T.

We calculate the capability of features as follows. A and B refer to the two classes. For each feature, we compute the means μ_a and μ_b, and the standard deviations σ_a and σ_b. Then, the capability of the feature is calculated as described by Liu [84].

$$C_T = \frac{\mu_a - \mu_b}{\sqrt{(\frac{\sigma_a^2}{n_a}) + (\frac{\sigma_b^2}{n_b})}} \tag{6.1}$$

We use Formula 6.1 to assess the capability of a feature to separate different classes, where n_a and n_b are the number of samples in classes A and B, respectively. Moreover, a threshold value is applied over the score of the features C_T. The most

significant features are kept according to the applied threshold value.

The training and testing stages follow the feature extraction. These stages are important for the accuracy of the overall system.

In the classification stage, we divided the data set into two equal-sized groups: *training* and *testing.* The training group is used to build a cascading classifier. Conversely, We use the testing group to calculate the performance of the classifier after the initial training phase. To optimize the number of features with the maximum classification accuracy rate, the threshold value is dynamically changed and the classification is performed again using the new feature set. This process is repeated until a classifier is found with the maximum performance and with the minimum number of coefficients. We then use 5-fold cross validation to the optimized coefficients in order to establish the validity of the classifier to avoid high bias or high variance (i.e., under-fitting or over-fitting the data). Figure 6.9 summarizes our method.

6.3.1 Conducting Experiments

Data is obtained from the PDU servicing five servers on a single rack. Our experiments run and affect the energy consumption of only one server while the others are kept idle in order to profile the energy usage pattern of each particular resource usage. We control these five servers even to the degree where we can place processes on particular cores of the CPU to ensure accurate resource allocation during each test.

Figure 6.10 illustrates the information that is obtained from the data center system monitoring tools. The red line indicates the energy consumption of a single PDU servicing five of our servers. Since one of these servers is used, while the other four servers on the same power circuit are kept idle, the visible change in energy consumption as we circulate through the tests represents the change produced by a single server.

These data were previously investigated and labelled. The data set is selected due to the various cases it includes. It is also used in other research which describes the data center and measurement setup in greater detail [17]. The energy consumption data are classified into *Idle Process, 2 CPU, 4 CPU, 6 CPU, 8 CPU, 10 CPU, 12*

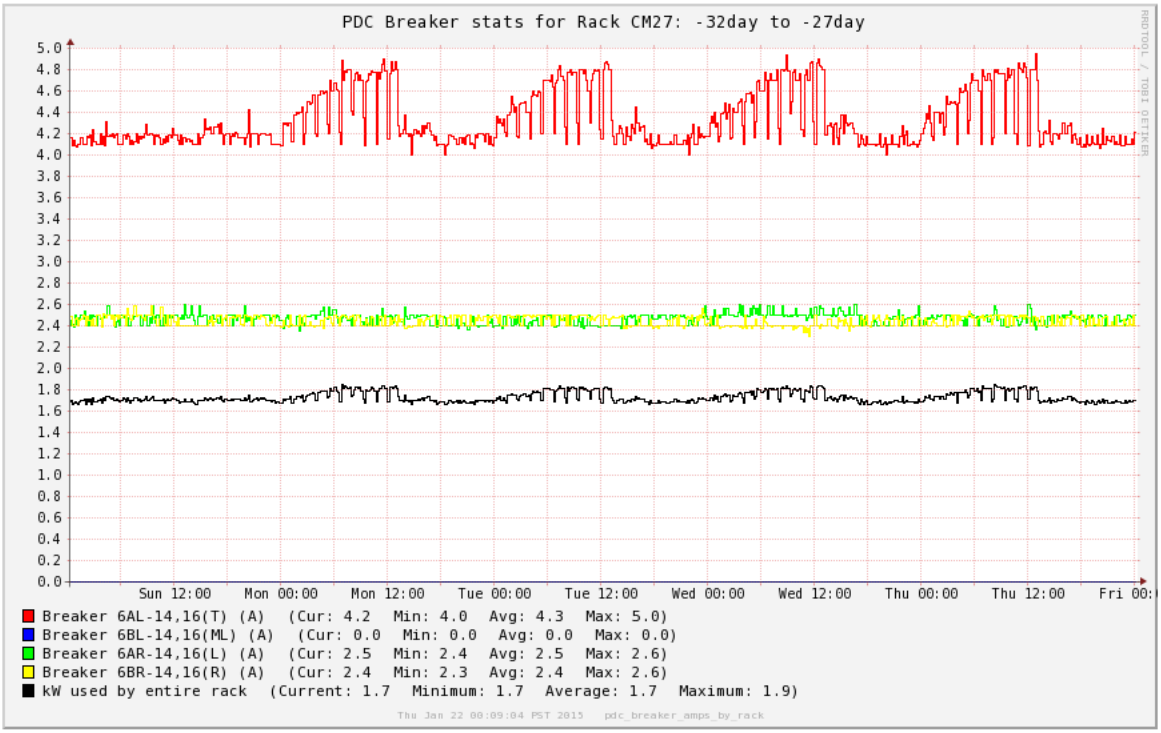

Figure 6.10: Experimental data over a period of four days. We run benchmarks to test 2 to 20 CPUs at full utilization for a fixed amount of time as well as our memory usage benchmark. The experiment is run repeatedly to confirm consistency and to confirm assumptions regarding measured energy consumption (e.g., the other servers on the rack are not contributing to energy consumption changes).

CPU, 14 CPU, 16 CPU, 18 CPU, 20 CPU, Copy Process, Small Copy Process, and Memory Volatile Process.

Once the data are cropped, statistical analysis methods are applied and the obtained vectors are used to built the $K \times N$ matrix. Then, a ranked list of features is build based on the criteria used to assess the capability of every feature for separating two labelled classes. Then, a dynamic threshold is applied to extract the most significant features. The energy consumption data decomposed into training and testing data. The training data are used to build the cascade classifier, while the testing data are used to calculate the classification accuracy rate. This method is repeated until the algorithm reaches the best classification rate with the minimum number of coefficients.

Table 6.1: Classification accuracy rates obtained through 5-fold cross-validation.

Cross-validation	Idle Process	2 CPU	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	18 CPU	20 CPU	Copy Process	Copy Small	Memory Volatile
50% no threshold	94.68	96.29	98.86	98.22	96.43	97.30	92.32	92.07	93.53	93.85	93.96	94.68	96.69	95.49
50% threshold	91.59	94.91	97.55	96.91	94.94	95.21	90.16	90.13	90.65	91.51	93.09	90.62	93.76	94.86
70% no threshold	91.07	96.25	98.00	97.40	96.16	96.27	92.5	92.44	92.3	92.99	94.21	93.77	96.2	95.39
70% threshold	90.55	95.21	97.22	96.7	95.22	95.13	91.43	92.29	90.71	91.52	93.15	91.19	93.8	94.71
80% no threshold	94.08	96.19	98.01	96.37	96.16	95.77	91.84	91.39	92.67	92.31	93.39	93.78	95.76	95.46
80% threshold	90.98	94.77	97.20	95.70	95.14	95.16	89.52	90.08	91.63	90.15	93.58	91.34	93.9	94.94
90% no threshold	91.37	96.96	97.35	96.29	94.8	94.93	92.35	93.22	92.31	92.84	93.79	93.72	95.68	95.47
90% threshold	89.16	95.52	96.92	96.04	94.25	94.84	91.3	93.25	91.43	90.78	93.02	91.49	93.77	95.32

6.3.2 Experimental Results and Discussion

This section provides a discussion of our experiments, classification method and findings.

Our method can be described as follows. First, we extract statistical features to the energy consumption data set. This operates on the raw, unmodified data. Following this we use the obtained information with the proposed method for feature extraction. After suitable features have been identified by the algorithm, we proceed to construct cascading NN and the classification accuracy rate is calculated. Figure 6.11 illustrates the performance of the cascade classifier for different classes used in our setup.

To validate the results, the 5-fold cross validation method is applied at the optimized threshold point. Classification accuracy rates are calculated using the obtained coefficients from the applied threshold. The result of the cross validation is presented in Table 6.1, which presents the cross-validation accuracy of each method per class. We present two rows for each training/test split. The table presents the identification accuracy in the cross-validation test phase for the labelled training set size. First we present the data when there is no threshold change applied to the training phase. The first row of the pair is the accuracy obtained in the testing phase for the associated training set size with prior feature extraction but without dynamic threshold changes. The second row represents the same type of data (accuracy in testing phase), except this time the training phase was preceded by feature extraction using statistical analysis which modified the threshold for improved results.

The table shows that using more than 50% training data does not lead to increased accuracy and thus is unnecessary. Increasing the training data to a size too large beyond the optimal point leads to overfitting of the data and does not allow for

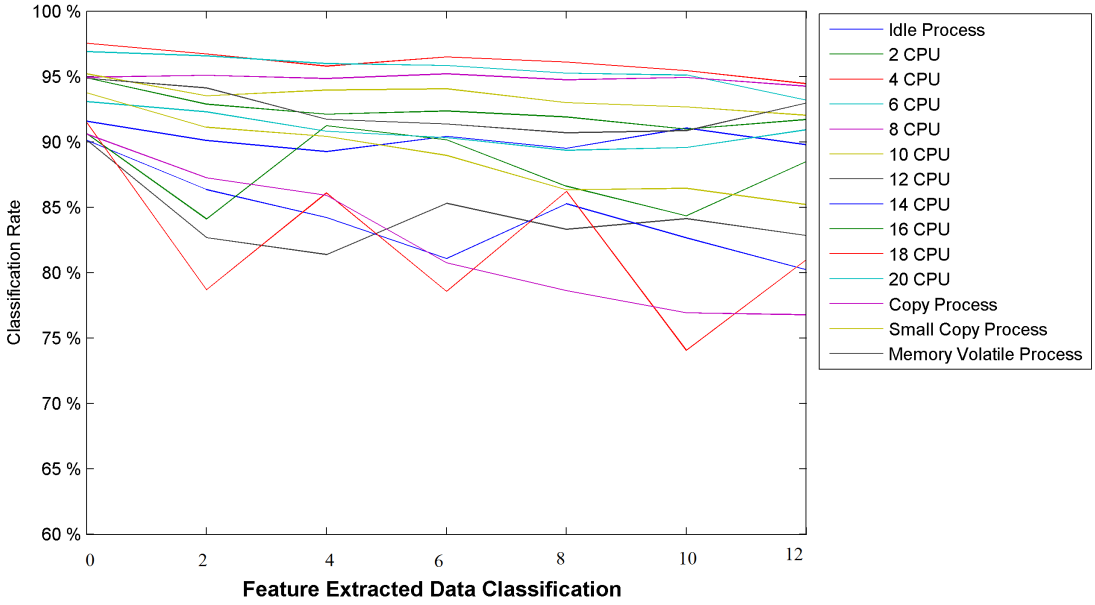

Figure 6.11: Classification accuracy rates for all classes corresponding to the number of features with different thresholds.

any claims about the actual quality of the classifiers.

The proposed method proves a good capability to classify successfully between different classes. The number of features decreases drastically with accepted classification rates. A feature extraction method for finding the most significant features is proposed and implemented to detect and classify energy usage patterns. The method is based on a ranking the feature according to its capability to distinguish between different classes. A Cascade classifier is constructed using 50% of data set and the remaining 50% is used to calculate the classification rate. The classification accuracy rate achieved by our method and is more than 90% accurate. For 14 different classes, increasing the number of features resulted in a decrease in correct classification as shown in Figure 6.11. The obtained results show the importance of the feature extraction step in developing energy usage and resource utilization patterns.

6.3.3 Adaptive Energy Consumption

We provide a description of the proposed model and the role it takes within an adaptive runtime reference model. We also highlight future avenues of research.

6.3.4 Motivation for Adaptive Solutions

Modern enterprise data centers currently contribute between 1.5% and 2% to global energy consumption and this figure is expected to increase even further in the future [72, 75, 101]. As such, energy consumption in data centers is a top level concern for the construction and operations of data centers. While the energy consumption of individual servers is considerably less than that required for HVAC measures in a data center, their effect, however, cannot be underestimated. Many research avenues propose energy savings via improved scheduling decisions at the server level using varying metrics.

Beloglazov et al. [13] consider resource utilization, network topologies and thermal states of physical machines to effect VM consolidations. Similarly, migration of services to the point of shutting down servers is investigated by Mazzucco and Mitrani [91].

We differ from these works by proposing process migration strategies at a finer grained information level. Unlike these works, which aim to consolidate processes on individual machines, we aim to find the best server for a process within the data center. This ensures that the process runs without resource constraints while, at the same time, minimizing energy usage.

6.3.5 Realizing Energy Adaptation in Data Centers

The ability to identify resource utilization of a single server accurately based on energy consumption allows for dynamic scheduling decisions to be made. Once a particular energy usage pattern is observed, our mechanism can identify the resource utilization that generated this pattern. Consider that process P generates energy consumption data E_1 over a period of time. Through our classification model, we are able to iden-

tify this energy usage pattern as belonging to a particular class of resource utilization U_1.

Furthermore, by having profiling knowledge about other servers in the data center, we are able to forecast the energy usage E_2 of the same resource utilization U_1. This allows us to make decisions based on the energy consumption difference. If a process is forecast to use less energy on another server while being given access to the same resource provisioning, it should be moved to that server.

The more fine grained this profiling and forecasting knowledge is, the more advantageous it will be for adaptive scheduling decisions. Conceptually, an MRAC model approach, as depicted in Figure 3.4, would be appropriate. Existing reference models are used to profile and predict process types and resource utilization to infer improved scheduling policies. However, this requires a priori information about the process that can potentially run on the server. This may or may not be possible. Therefore, we propose to integrate our classification approach into an MIAC model, as depicted in Figure 3.5, which continually identifies a new up-to-date reference model.

Using an MIAC model with our approach yields certain benefits. While the reference model is running to perform scheduling decisions, it can build and update new reference models for a variety of processes.

Having profiled the process P, for example, an MIAC model can classify this process and assign it to a certain resource usage group. Then this process can be migrated to different servers to generate energy usage profiles dynamically for this specific process on different server types. In this way, a knowledge base is dynamically generated, which can be integrated into the MIAC model and subsequently used to determine on which server any process of similar resource usage should run for improved energy consumption.

6.3.6 MIAC Deployment Architecture

Considering Figure 3.5 our adaptive feature extraction and classification engine is placed in the component labelled *Proposed Model Class Identification*. Input to this

component is proposed to take the form of energy usage data in the same format as it was used for training our classifiers described earlier. Then, if this usage can be attributed to a known class, a further comparison to the knowledge base occurs. Here the model requests information about whether other servers have previously been profiled for this type of process and if so did this process' resource utilization result in different energy usage patterns on different hardware. Using this knowledge, the process is either migrated to another server with available hardware and, most importantly, is expected to use less energy for this class of process.

Alternatively, if no such information exists, the process can be migrated autonomously to other servers which fulfil the processes' resource requirements to establish energy usage patterns dynamically. This approach would lead to the creation of a process specific knowledge base encapsulating energy consumption and resource utilization.

Most processes are not stable in their resource utilization and often have bursty resource usage. Bursts are associated with a change in resource utilization (e.g., a burst may manifest itself through increased CPU usage, or larger memory footprint or changing network traffic). While such a burst occurs, our model will no longer consider the process to be part of its original class. Once this bursty process is identified as belonging, even temporarily, to a different class, the model can determine whether it should be migrated to guarantee energy efficient operations while satisfying resource guarantees.

6.4 Chapter Summary

In this chapter, we showned that we are able to identify servers for which the performance and energy criteria meet a software job's requirements. We demonstrated that, with the help of our tool and data, one can chose the best machine on which to run a particular software application. This is possible for CPU, memory and disk I/O as demonstrated in Section 6.2.

Our work on machine learning helps us identify which type of application is running based purely on the underlying energy consumption. For this purpose, we tested

and identified machine learning methods and evaluated their accuracy for classifying energy usage patterns on data obtained from the enterprise data center EDC2. The accuracy of our approach is in excess of 90% for training set sizes using no more than 50% of the data.

Of the methods used, we determined via an experimental approach that feature extraction prior to training improves accuracy in classification. The method for feature extraction chosen is statistical analysis. We make this choice due to the data's similarity to other domains where *statistical analysis* methods have been found to perform best.

Finally, we can integrate our classification engine into the controller of an adaptive reference model. Due to the nature of our highly dynamic classification approach, we propose an MIAC based model to profile, identify and migrate individual processes adaptively within the data center for maximal energy savings.

Chapter 7

Valuation and Discussion of Results

This chapter discusses the findings obtained during the process of this research, including the experiments and case studies along with their results presented in previous chapters. We discuss selected implications of these findings as well as requisite context to accrue energy savings when running software applications.

7.1 Case Studies

This section provides a discussion of the results from our case studies and experiments. The experiments, results and findings are but one aspect of the contributions of this dissertation. The other area of contributions falls into the area of software engineering, and more precisely context management, smart systems and self-adaptive software. The findings from our case studies can be used within these fields, in current and future research, with the goal to reduce energy consumption in software.

In the following sections, we provide insights on what the results mean in terms of feasibility, future research and overall impact.

7.1.1 PDU

One of the main take-aways from this research is that PDUs are indeed a viable entry point to obtain energy consumption data of an individual server's software processes. This finding constitutes and important requirement for future the design of future

data centers and is a true contribution of this research.

A pivotal moment in this research that allowed us to proceed, was brought about by the collaboration with the university's data center operations staff. The staff granted us access to their internal metrics and monitoring software. With this access, we were able to see (nearly) live changes in energy consumption as our experiments ran. After we established that PDUs in data centers were indeed accessible to us, our case studies were designed to answer progressively complex research questions. The PDU as a viable source for software energy consumption data is significant because it is the one place that allows us, under current data center setups, to obtain energy consumption data for entire servers. This setup is superior to using built-in sensors as it is guaranteed to include all energy consumption. Conversely, built-in sensors do not cover the entire server as not every component can be fitted with a sensor. Moreover, the more sensors are used, the greater is the compounding effect of errors and measurement tolerances. Thus, using built-in server instrumentation will not provide a true overview of the energy consumption of the hardware.
Thus, not every component in a server is instrumented. The benefit of our approach and arrangement is that every data center has PDUs connected to their server racks. Consequently, we view the PDU as a readily available data source that provides a holistic view of the energy consumption of a server. As an extension, we can scale this to multiple PDUs not just in one data center, but in a distributed data center.

7.1.2 Initial Steps

One of the first questions that had to be answered was whether running software applications would produce changes in energy consumption that was measurable at the PDU level.

One of the first case studies we designed focussed on the identification and analysis of the main causes of energy consumption in servers, laptops, desktops and mobile devices. Prior research and work conducted in the Rigi lab as part indicated that this might be a promising avenue and lead to our further experiments in the UVic EDC2 data center. More specifically, we were interested in demonstrating if energy consumption profiles could be tied to resource usage of software applications when

using PDU power measures as a data source.

Once we had established that the PDU would provide usable data for us, we refined our experiments. Using the PDU to demonstrate that different resource usage of software applications produces measurable differences in energy consumption is another contribution of this research. Illustrating this, among others, are Figure 5.10 and Figure 5.12. These two figures demonstrate clearly that the effects of hardware usage on energy consumption are measurable and visible. This is significant, because it allowed us to progress in our research further and not only measure the energy consumption of different software applications, but also attempt underlying system modifications aimed at producing observable effects in performance and energy consumption. More specifically, once we knew that we could measure how much energy a software application requires to run, we could attempt to alter the software to achieve better performance and/or better energy consumption patterns. In some cases, this can be achieved successfully with minor modifications; in other cases, it requires careful analysis of the underlying software to produce effective changes that do not alter the overall side effects of the software application. As such, this avenue is promising as a direction of future research. Our approach of relatively high level system call redirection and modification is also in line with related works that discourages lower level modifications at the assembly level or binary instruction level. At the same time, our approach, however, is at a lower level than the often attempted modifications at the software development level.

7.1.3 Focussing on Finer Level Control

Once we established that we were able to not only produce and measure different energy profiles, we needed to automate the process—automation is key for scalability. We developed scripts and software that would collect and analyse the energy consumption data. This analysis stems from our own machine learning approach using *Neural Networks*. We settled on Neural Networks after also experimenting with Support Vector Machines. However, Neural Networks were easier to implement, produced better results and were more suitable to the problem of classifying multiple categories.

With this approach, we are able to classify energy usage patterns into categories of

resource usage. In other words, we developed a system that is capable of correlating energy consumption with resource usage. Given a pattern of energy consumption, we can now attribute this energy consumption to a specific resource (e.g., memory, disk, CPU or network). Automating this, now allows us to make deployment decisions. From energy consumption we identify resource usage. From resource usage, we can then identify other machines that have "better" energy profiles which represent the server where a software application "should" run for optimal energy and performance results.

We also explored individual resource usages further at a more fine grained level. We investigated changes in energy consumption down to specific CPUs and processor cores. We also designed a case study to determine whether modifications at the system call level as well as intelligent restructuring of resource requests at run time can impact the overall energy consumption of digital ecosystems. To achieve this, we modified applications and system calls to affect energy consumption and determine which changes yield the greatest improvements. One such case study, where we alter software in order to observe and measure changes in energy consumption, is showcased in Figure 5.8. The experiment depicted in this figure demonstrates that there are differences not only in the type of hardware resource a software application is using, but also in the way that a particular resource is accessed. This figure shows that system call redirection to more energy efficient system calls can yield performance improvements.

We can take this line of research one step further and consider the conclusions we can draw from Figure 6.8. What stands out in this figure is that there is a clear difference in the amount of energy consumed for different hardware resources. Not only that, but the differences between machines at different resource utilization levels stand out. Consequently, it becomes evident that this information can be used to automate scheduling and deployment decisions based on energy and performance requirements. Moreover, the figure also clearly highlights that the largest contributor to software energy consumption is CPU. This allows us to direct optimization efforts towards the area in software which yields the largest energy consumption gains. Contrary to our approach, another important consideration is that usually in modelling data centers identical machines are simplified to use identical amounts of energy. We have shown that this is perhaps an over-simplification, though, since our work is new,

further research in this domain is needed.

Ultimately, it is important to note that none of these findings would have been possible without the ability to use the PDU as a source of energy consumption data for the entire server.

7.2 Avenues for Future Research

This section provides guidelines and recommendations to future researchers interested in the intersection of *green computing and software engineering.* It is intended to be a reflection of the lessons learned during the process of going through this research program. Both the high-level and low-level insights obtained pursuing this research topic lend themselves to recommendations for future generations, who may undertake similar research projects.

7.2.1 Risks and Progress in PhD Studies

When we began this project, we knew very little about the feasibility of this research. The primary concern was whether we would be able to obtain measurements to support the development of our experiments. While this in and of itself is a somewhat frightening prospect, this is an intrinsic part of research and is associated with risks involved concluding the research project. This factor is to the fact that research is supposed to be novel and, thus, involves significant unknowns.

One of these unknowns was the availability of data points. At the beginning, it became clear that it is incredibly important to ensure that both the sampling rate and precision of available measurements in data centers matches the requirements we had to make meaningful statements about green computing and software applications. For us, we were limited to one data point every minute by the design of the data center monitoring setup. This, in turn, informed our experimental design and resulted in experimental setups that generated enough data points. More precisely, it also meant that our experiments could not run for less than one minute. In our setup, we generally had experiments that ran for at least half an hour to generate at least 30 data point. One potential problem with this arrangement is that we are

dependant on external groups for our success. Continued close cooperation with the University's data center operations team was required to obtain measurements on an ongoing basis. This presented a risk to this research as their support could have been withdrawn due to factors outside our control at any moment. It is important to evaluate carefully whether such risks are worth it prior to starting the experiments and have contingency plans ready.

Equally important is a clear understanding of what defines the successful completion of the data collection and experimental phases. To achieve this, it is important to define a measure of success prior to beginning the experiments. This measure of success must factor into the associated risks (e.g., loss of data sources). Collecting as much data as possible and as quickly as possible seemed to be the best approach considering the external risk of losing access to the data reporting tools.

Another critical recommendation is to branch out to other fields early in the research project. It is important to not work in isolation, both in terms of collaborators and in terms of exposure to ideas. Existing ideas may help advance the project to otherwise unattainable heights by applying lessons and advances to your own research. For example, this research drew heavily from the *software engineering for self-adaptive systems (SEAMS)* community. It is advantageous to adapt and acknowledge the lessons and contributions from other groups, with modifications, to drive progress your own research program—no need to reinvent the wheel.

Finally, a more general recommendation is to be aware of the effects of your experiments. One interesting questions throughout this research endeavour was to consider how much energy each experiment requires and how much energy we would be able to save eventually by our findings in the future. It is fun to think about these almost paradoxical stipulations.

7.3 Summary

In this section, we revisited some of the findings from earlier chapters and put them into perspective in terms of the overall goals of this dissertation—*reduce energy consumption while increasing or maintaining performance.* Step by step, our research

progressed into more intricate details and fine grained observations and system modifications. Along with these detailed investigations we developed analysis tools that helped us classify the changes we were observing.

At the PDU level, we are able to observe these fine grained changes to software and their effects on energy consumption. This path of using measurements at the PDU level along with the support software we developed is an interesting and worthwhile avenue of green computing research.

Chapter 8

Conclusions and Future Research

This chapter summarizes our case study findings and outlines avenues for future research.

8.1 Summary

In the course of this research, we conducted several experiments and developed numerous software applications in support of this research and other research projects that coincided with the main thrust of this line of work [18, 64, 87].

Over time, the dissertation's topic evolved significantly. Starting with studying related works and conducting a literature survey (cf. Chapter 3), we were able to situate our research approach among existing efforts.

The two fundamental fingds are that (1) energy consumption patterns can be used to identify software applications running on a server in a data center, and (2) data from PDUs can be used to do this type of analysis and pattern recognition [17]. Using these two aspects of our work as the building blocks for the remainder of our research, we were able to investigate the energy consumption of software and their impact on hardware resource usage. To facilitate this line of work, we needed to develop analytics tools that collect, sanitize, process and analyse data. In turn, we developed a machine learning approach that works in conjunction with our profiling tools [19].

Our work was possible because we had access to energy consumption metrics of

servers in a real data centre. In the future, we would like to have the ability to measure energy consumption in desktops and smartphones more accurately, albeit with more effort and resources—both financially and in terms of hardware. Within the data center, we had control over the exact machine instruction and system call that is generated from any given piece of software (i.e., through the compiler or redirection and modification of system calls). This allowed us to create baselines against which we were able to measure the impact of software changes as well as hardware changes (cf. Chapter 5).

Another contribution is to provide the ability to actually profile and benchmark the energy efficiency of software through the use of statistical models, frameworks and actual measurements that are unique to each target environment, yet automated for any number of devices. We showcase the process of utilizing case studies and data analytics for use in our models in Chapters 4 and 5. Chapter 6 highlights, using our data, the type of scheduling decisions one can make within our system by using performance and energy efficiency criteria.

The software framework we used to profile and analyse servers in our data center is another contribution which integrates existing principles with new ones into a comprehensive unit to allow accurate information on the energy efficiency of software on a given hardware.

We illustrate the applicability of our research findings with two scenarios. Assume that the operator of a large data centre wishes to install new servers with several viable models on the market. The main considerations are performance and energy consumption (both direct and indirect). With the help of our profiling framework, one can answer the question of direct energy consumption relative to performance. Our software framework can be installed on the new hardware and it begins to perform software benchmarks designed to establish the energy efficiency and performance metrics of the new hardware. Energy consumption data will be fed into models correlating function calls, compile time optimizations and runtime performance to provide a comprehensive view on the energy consumption properties of the given hardware dependant on the type of software that is run. Using this framework, the operator will be able to decide what hardware is effective in terms of energy consumption and performance for a given workload. These insights can guide purchasing and scheduling

decisions. The software framework can also continually monitor the energy consumption and efficiency of the software to detect potential imminent failures, malware infections or other anomalies through the use of the energy consumption models. While the exact categorization of energy consumption anomalies was not part of this research, we did detect anomalies. GENI rack server 04 was approximately 20% more energy efficient than the other two GENI rack servers despite being nominally identical.

Performing this type of analysis with our benchmarking tool is possible by using the software excerpts provided in the appendix of this dissertation as well as through customizations of the architecture of the scripts showcased through this work in tables and figures. The resulting framework and research findings will be of interest to anyone who can be incentivised to reduce energy consumption of software.

Who will potentially benefit from the results of our work? First of all data centres need to consider the implications brought forward here further when considering the type of hardware they are to buy (i.e., energy efficiency must not mean lower performance in the traditional sense [17]). Moreover, information of the proposed framework can be made available to clients of the data centre, who in turn can be incentivised to create and run energy efficient software on energy efficient hardware. Third, software developers for mobile devices care already a great deal about energy efficiency, but little actual hard data are available and used to make decisions when developing software. With our framework, the developer is relieved from a large portion of the tasks needed for energy optimization by supporting the redirection of system calls to more energy efficient servers. Lastly, through this framework and the knowledge base contained within it, the applications themselves can apply dynamic decision making models for self-optimization in regard to deployment decisions.

8.2 Contributions

We are pleased to report that the research questions posed in Chapter 1 have been answered.

R1:

What are the key resource usage factors of software applications that contribute to the energy consumption profile of interconnected IoT applications?

Contributions:

- Automated method for identifying a server's energy consumption per resource type and load.
- Energy consumption profiling tool for a data centre.

This was achieved and demonstrated in our publications [17, 19] and described in Chapters 5 and 6. We have used machine learning to identify a server's energy consumption per resource type and load automatically as part of our energy profiling tool. With this tool, we are also able to identify the key resource usage factors of software applications—namely the CPU.

R2:

What is the degree to which high-level modifications at the application level, in terms of resource utilization and service deployment, can be utilized to optimize the overall energy consumption of the application's digital ecosystem?

Contributions:

- Application specific, automated and dynamic software modification to reduce energy consumption.
- Dynamic scheduling solution for software within data centres.
- Identification of potential energy savings specific to individual applications.

We were able to identify several energy saving approaches for specific applications. For CPU-intensive applications, we were able to show that smart allocation of resources must match the software's capabilities (e.g., allocate CPUs according to multi-threading requirements). Specifically, we make the case that increasing the number of threads or CPUs for an application must be coordinated to prevent a deterioration in performance.

R3:

Does dynamic redirection of specific system calls and resource provisioning requests lead to energy optimization while not negatively impacting performance (i.e., either maintaining the same performance level or improving the performance level)?

Contributions:

- Automatic and dynamic changes to application level interactions with the underlying system to optimize energy consumption of software applications.

We were able to answer this research question through system call modifications for certain applications. Our system call modifications for disk I/O-heavy operations resulted in performance increases as well as energy usage decreases.

R4:

How can we obtain, store and manage contextual information of software applications and their digital ecosystem?

Contributions:

- A contextual model to aggregate, store and analyze energy data within a new framework for data centres with the goal of improving energy consumption of software.

We have introduced our model to collect, store, analyse and effect system change with data. Our proposed model is based on dynamic reference models which cover similar problem domains. The distinction of our model is that it is applied to energy consumption in data centers for software energy consumption.

R5:

To what extent is dynamic runtime modification of resource allocation manageable in a self-adaptive framework using self-adaptive models and control theory approaches?

Contributions:

- Identification of efficacy of current approaches taken by this dissertation.

- Identification of promising future avenues of research.

Dynamic runtime modification of resources is automatable, measureable and analysable. We have shown this in our research. However, the extent to which this can be applied to resource allocation and software deployment decisions at a large scale remains an open question. We were only able to apply our model at the small scale setup in the UVic's EDC2 data center where we controlled two server racks.

8.3 Future Research

Great research produces more questions than it answers. While answering and addressing several research questions, our research lead to the identification of further challenges and open problems.

One open problem is to deploy our profiling software at a much larger scale. For example, at multiple racks in the data center. Another open problem related to this one is the improvement of our scheduling component, the component which identifies the hardware, where a job should run after its initial profiling has been completed. This would require more investigation into improving our models and the interaction with existing scheduling research.

Machine learning approaches can be improved by tuning of our existing approach. This type of future work can be geared toward improving the classification rate of software jobs based purely on their energy consumption.

Another interesting future research avenue is to automate, at scale, the deployment of software to the correct server based on policies which interface with the scheduling models. Ultimately, the more distinct components are brought together the more usable and measurable the effects become. Our existing framework is designed in such a way as to easily allow its integration or consumption with other frameworks.

Finally, it is important to mention that there is more work to be done in the realm of distributed applications. Our work, as it is, applies to the lower levels of the distributed software scheduling and deployment hierarchy. However, it might be

worthwhile to investigate how data obtained from the level at which we operate, can influence and benefit smart deployment engines of distributed digital ecosystems.

Appendix A

Benchmark Sample Code

A.1 Sample Code to Copy File with *fputs/fgets*

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char ch;
   FILE *source, *target;
   char source_file[] = "in.txt";
    char target_file[] = "out.txt";
    source = fopen(source_file, "r");

   if( source == NULL )
   {
      printf("File not found %s...\n", source_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }

    target = fopen(target_file, "w");

   if( target == NULL )
   {
      fclose(source);
      printf("Cannot open file for writing %s...\n", target_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }

   while( ( ch = fgetc(source) ) != EOF )
      fputc(ch, target);

   printf("File copied successfully.\n");
```

```
    fclose(source);
    fclose(target);

    return 0;
}
```

A.2 Sample Code to Copy File with *fputs/fgets*

```
\lstset{language=C,
           basicstyle=\ttfamily\scriptsize,
           keywordstyle=\color{blue}\ttfamily,
           stringstyle=\color{red}\ttfamily,
           commentstyle=\color{green}\ttfamily,
          breaklines=true
          }
\begin{lstlisting}
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char ch;
   FILE *source, *target;
   char source_file[] = "in.txt";
    char target_file[] = "out.txt";
    source = fopen(source_file, "r");

   if( source == NULL )
   {
      printf("File␣not␣found␣%s...\n", source_file);
      printf("Press␣any␣key␣to␣exit...\n");
      exit(EXIT_FAILURE);
   }

    target = fopen(target_file, "w");

   if( target == NULL )
   {
      fclose(source);
      printf("Cannot␣open␣file␣for␣writing␣%s...\n", target_file);
      printf("Press␣any␣key␣to␣exit...\n");
      exit(EXIT_FAILURE);
   }

    char line [80];
    int SIZE = 80;
   while( ( ch = fgets(line, SIZE-1,  source) ) != NULL ) {
      line[SIZE-1]='\0';
      fputs(line, target);
   }

   printf("File␣copied␣successfully.\n");

   fclose(source);
   fclose(target);

   return 0;
}
```

A.3 Sample Code Used to Copy File with *read-/write* Large Buffer

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define READ_NUM 20000
int main()
{
   char ch;
   FILE *source, *target;
   char source_file[] = "in.txt";
    char target_file[] = "out.txt";
    source = fopen(source_file, "rb");

   if( source == NULL )
   {
      printf("File not found %s...\n", source_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }
    int f_in = fileno(source);

    target = fopen(target_file, "wb");

   if( target == NULL )
   {
      fclose(source);
      printf("Cannot open file for writing %s...\n", target_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }
   int f_out = fileno(target);

    char line [READ_NUM+1];
    int SIZE = READ_NUM;
    int n;
   while((n = read(f_in, line, SIZE))  != 0 ) {
      write(f_out, line, n);
   }


   printf("File copied successfully.\n");

   fclose(source);
   fclose(target);

   return 0;
```

```
}
```

A.4 Sample Code Used to Copy File with *read-/write* Small Buffer

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define READ_NUM 8000
int main()
{
   char ch;
   FILE *source, *target;
   char source_file[] = "in.txt";
    char target_file[] = "out.txt";
    source = fopen(source_file, "rb");

   if( source == NULL )
   {
      printf("File not found %s...\n", source_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }
    int f_in = fileno(source);

    target = fopen(target_file, "wb");

   if( target == NULL )
   {
      fclose(source);
      printf("Cannot open file for writing %s...\n", target_file);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }
   int f_out = fileno(target);

    char line [READ_NUM+1];
    int SIZE = READ_NUM;
    int n;
   while((n = read(f_in, line, SIZE))  != 0 ) {
      write(f_out, line, n);
   }


   printf("File copied successfully.\n");

   fclose(source);
   fclose(target);

   return 0;
}
```

A.5 Script to Control CPU and Memory Benchmarks

```
#needed for logging
import inspect
import logging

#needed for starting subprocesses
import subprocess

#needed to start and stop process at correct times
import time

#needed for making a call graph
from pycallgraph import PyCallGraph
from pycallgraph.output import GraphvizOutput

SECOND = 1
MINUTE = SECOND * 0
TEST_TIME = 1 # Minute

def function_logger(file_level, console_level=None):
    function_name = inspect.stack()[1][3]
    logger = logging.getLogger(function_name)
    logger.setLevel(logging.DEBUG) #By default, logs all messages

    if console_level != None:
        ch = logging.StreamHandler() #StreamHandler logs to console
        ch.setLevel(console_level)
        ch_format = logging.Formatter('%(asctime)s - %(message)s')
        ch.setFormatter(ch_format)
        logger.addHandler(ch)

    fh = logging.FileHandler("{0}.log".format(function_name))
    fh.setLevel(file_level)
    fh_format = logging.Formatter('%(asctime)s - %(lineno)d - %(levelname)-8s - %(
        message)s')
    fh.setFormatter(fh_format)
    logger.addHandler(fh)

    return logger

def f1():
    f1_logger = function_logger(logging.DEBUG, logging.ERROR)
    f1_logger.debug('debug message')
    f1_logger.info('info message andi')
    f1_logger.warn('warn message')
    f1_logger.error('error message')
    f1_logger.critical('critical message')
```

```
def f2():
    f2_logger = function_logger(logging.WARNING)
    f2_logger.debug('debug message')
    f2_logger.info('info message some info')
    f2_logger.warn('warn message')
    f2_logger.error('error message')
    f2_logger.critical('critical message')



def f_idle(duration):
    """duration in minutes"""
    f_idle_logger = function_logger(logging.DEBUG, logging.ERROR)

    f_idle_logger.info('STARTING idle process')
    # process = subprocess.Popen(["./a.out"])
    time.sleep(MINUTE * duration)
    # process.kill()
    f_idle_logger.info('FINISHED idle process')

def f_cpu(duration, cpus):
    f_cpu_logger = function_logger(logging.DEBUG, logging.ERROR)
    f_cpu_logger.info('STARTING '+str(cpus)+'CPUs process')
    shell_script = "./test_"+str(cpus)+"_cpus.sh"
    process = subprocess.Popen([shell_script])
    time.sleep(MINUTE * duration)
    process.kill()
    subprocess.call(['killall', 'sysbench'])
    f_cpu_logger.info('FINISHED '+str(cpus)+'CPUs process')


def f_cp_copy(duration):
    """copies a file with fgets"""
    f_cp_logger = function_logger(logging.DEBUG, logging.ERROR)

    f_cp_logger.info('STARTING copy process ' + str(duration) + 'minutes')
    process = subprocess.Popen(["./copy.sh"])
    time.sleep(MINUTE * duration)
    f_cp_logger.info('ATTEMPTING TO FINISH copy process')
    process.kill()
    f_cp_logger.info('FINISHED copy process')
    f_cp_logger.info('CLEANING UP copy process')
    process = subprocess.call(["./clean.sh"])

def f_cp_small_copy(duration):
    """copies a file with fgets"""
    f_cps_logger = function_logger(logging.DEBUG, logging.ERROR)

    f_cps_logger.info('STARTING copy process ' + str(duration) + 'minutes')
    process = subprocess.Popen(["./copy_small.sh"])
    time.sleep(MINUTE * duration)
    f_cps_logger.info('ATTEMPTING TO FINISH copy process')
    process.kill()
```

```
    f_cps_logger.info('FINISHED copy process')
    f_cps_logger.info('CLEANING UP copy process')
    process = subprocess.call(["./clean_small.sh"])

def f_mem_volatile(duration):
    """copies a file with fgets"""
    f_memv_logger = function_logger(logging.DEBUG, logging.ERROR)

    f_memv_logger.info('STARTING memv process ' + str(duration) + 'minutes')
    process = subprocess.Popen(["./a.out"])
    time.sleep(MINUTE * duration)
    f_memv_logger.info('ATTEMPTING TO FINISH memv process')
    process.kill()
    f_memv_logger.info('FINISHED memv process')


def main():
    f_main_logger = function_logger(logging.DEBUG, logging.ERROR)
    f_main_logger.info('STARTING A NEW TEST SUITE')
    f_main_logger.info('STARTING f1')
    f1()
    f_main_logger.info('Finished f1')
    f_main_logger.info('STARTING f2')
    f2()
    #IDLE
    f_main_logger.info('STARTING idle process')
    f_idle(10)
    f_main_logger.info('FINSIHED idle process')
    #CPU4
    duration = 60
    max_cpu = 20 + 1 # cpus + 1 for range()
    for i in range(2, max_cpu, 2):
        f_main_logger.info('STARTING '+str(i)+'CPU')
        f_cpu(duration, i )
        f_main_logger.info('FINISHED '+str(i)+'CPU')
        #IDLE for an hour
        f_main_logger.info('STARTING idle process')
        f_idle(20)
        f_main_logger.info('FINSIHED idle process')
    #f_idle_logger.info('STARTING idle process')
    #COPY
    # cp to copy
    f_main_logger.info('STARTING copy process')
    f_cp_copy(20)
    f_main_logger.info('FINISHED copy process')
    # cp to copy
    f_main_logger.info('STARTING copy process')
    f_cp_copy(20)
    f_main_logger.info('FINISHED copy process')
    # cp to copy
    f_main_logger.info('STARTING copy process')
    f_cp_copy(20)
    f_main_logger.info('FINISHED copy process')
```

```
    #IDLE
    #IDLE
    f_main_logger.info('STARTING idle process')
    f_idle(20)
    f_main_logger.info('FINSIHED idle process')
    #COPY many small files
    # cp -r to copy
    f_main_logger.info('STARTING copy_small process')
    f_cp_copy(60)
    f_main_logger.info('FINISHED copy_small process')
    #IDLE

    f_main_logger.info('STARTING mem_volatile process')
    f_mem_volatile(60)
    f_main_logger.info('FINISHED mem_volatile process')

    logging.shutdown()

if __name__ == "__main__" :
    MAKE_GRAPH = True
    graphviz = GraphvizOutput()
    graphviz.output_file = "callgraph.png"
    if MAKE_GRAPH :
        with PyCallGraph(output=graphviz):
            main()
    else:
        main()

}
```

A.6 Script to Control CPU

```
#!/bin/bash
while :
do
    echo "starting␣4␣CPU␣test"
    taskset -c 0,1,2,3 sysbench --test=cpu --num-threads=4 --cpu-max-prime=9000000
        run; #should be 1 zero less to complete before time is up
    sleep 1
done
}
```

A.7 Primitive Code for Memory Testing

```
int main() {
    long size = 2294967296;
    int *a = malloc((sizeof(int))*size);
    long i;
    int x = 2;
    while (1) {
        x += 1;
        for (i = 0; i < 1000; i++) {
            a[i] = x+(int)i;
        }

    }
}
```

A.8 Primitive Code for Memory Testing

```
#include <unistd.h>
int main() {
    long size = 2294967296;
    int **a = malloc(sizeof(int)*size);
    long i;
    int x = 2;
    while (1) {
        sleep(2);
    }
}
```

A.9 Second Script for Memory Testing

All other CPU benchmarks differ in the number of allowed cores and threads only.

```
#!/bin/bash
echo date
echo "starting␣memory␣test"
for i in {1..5}
do
        stress --vm 1 --vm-bytes 1G --vm-keep --timeout 290s
        sleep 10
        echo "done␣memory␣test"
done

sleep 300

for i in {1..5}
do
        stress --vm 1 --vm-bytes 5G --vm-keep --timeout 290s
        sleep 10
        echo "done␣memory␣test"
done

sleep 300
}
```

Bibliography

[1] Abts, D., Marty, M. R., Wells, P. M., Klausler, P., and Liu, H. Energy proportional datacenter networks. *ACM SIGARCH Computer Architecture News 38*, 3 (2010), 338–347.

[2] Aggarwal, K., Zhang, C., Campbell, J. C., Hindle, A., and Stroulia, E. The power of system call traces: Predicting the software energy consumption impact of changes. In *Proceedings of the 24th Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2014)* (2014), ACM, pp. 219–233.

[3] Aksanli, B. *Energy and Cost Efficient Data Centers*. PhD thesis, University of California, San Diego (UCSD), California, 2015.

[4] Aksanli, B., Venkatesh, J., Zhang, L., and Rosing, T. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. *ACM SIGOPS Operating Systems Review 45*, 3 (2012), 53–57.

[5] Amin-Naseri, M. R., and Soroush, A. R. Combined use of unsupervised and supervised learning for daily peak load forecasting. *Energy Conversion and Management 49*, 6 (2008), 1302–1308.

[6] Ardagna, D., Ghezzi, C., and Mirandola, R. Rethinking the use of models in software architecture. In *Quality of Software Architectures. Models and Architectures*. Springer, 2008, pp. 1–27.

[7] Assmann, U., Götz, S., Jézéquel, J.-M., Morin, B., and Trapp, M. A reference architecture and roadmap for Models@run.time systems. In *Models@run.time*, LNCS 8378. Springer, 2014, pp. 1–18.

[8] Balakrishnan, G., Reps, T., Melski, D., and Teitelbaum, T. WYSINWYX: What you see is not what you execute. In *Verified Software: Theories, Tools, Experiments*. Springer, 2008, pp. 202–213.

[9] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC)* (2009), ACM, pp. 280–293.

[10] Baresi, L., and Pasquale, L. Adaptive goals for self-adaptive service compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS 2010)* (2010), IEEE, pp. 353–360.

[11] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review 37*, 5 (2003), 164–177.

[12] Barroso, L. A., and Hölzle, U. The case for energy-proportional computing. *IEEE Computer 40*, 12 (2007), 33–37.

[13] Beloglazov, A., and Buyya, R. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)* (2010), IEEE, pp. 826–831.

[14] Beloglazov, A., and Buyya, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience 24*, 13 (2012), 1397–1420.

[15] Bencomo, N., France, R., Cheng, B. H. C., and Assmann, U. *Models@run.time—Foundations, Applications and Roadmaps*. LNCS 8378. Springer, 2014.

[16] Bergen, A., Coady, Y., and McGeer, R. Client bandwidth: The forgotten metric of online storage providers. In *Proceedings of the IEEE Pacific Rim Conference onCommunications, Computers and Signal Processing (PacRim 2011)* (2011), IEEE, pp. 543–548.

[17] Bergen, A., Desmarais, R., Ganti, S., and Stege, U. Towards software-adaptive green computing based on server power consumption. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014)* (2014), ACM, pp. 9–16.

[18] Bergen, A., Taherimakhsousi, N., Jain, P., Castañeda, L., and Müller, H. A. Dynamic context extraction in personal communication applications. In *Proceedings of the 23rd Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2013)* (2013), ACM, pp. 261–273.

[19] Bergen, A., Taherimakhsousi, N., and Müller, H. A. Adaptive management of energy consumption using adaptive runtime models. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)* (2015), IEEE, pp. 120–126.

[20] Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M. Q., and Pentikousis, K. Energy-efficient cloud computing. *The Computer Journal 53*, 7 (2010), 1045–1051.

[21] Berral, J. L., Goiri, Í., Nou, R., Julià, F., Guitart, J., Gavaldà, R., and Torres, J. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking* (2010), ACM, pp. 215–224.

[22] Bobroff, N., Kochut, A., and Beaty, K. Dynamic placement of virtual machines for managing sla violations. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)* (2007), IEEE, pp. 119–128.

[23] Bohra, A. E., and Chaudhary, V. Vmeter: Power modelling for virtualized clouds. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW 2010)* (2010), IEEE, pp. 1–8.

[24] Brun, Y., Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., and Shaw, M. Engineering Self-Adaptive Systems Through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*. Springer, 2009, pp. 48–70.

[25] Bruneo, D., Longo, F., Ghosh, R., Scarpa, M., Puliafito, A., and Trivedi, K. S. Analytical modeling of reactive autonomic management techniques in iaas clouds. In *Proceedings of the IEEE 8th International Conference on Cloud Computing (CLOUD 2015)* (2015), IEEE, pp. 797–804.

[26] Bunse, C., Höpfner, H., Roychoudhury, S., and Mansour, E. Choosing the "best" sorting algorithm for optimal energy consumption. In *Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT 2009)* (2009), pp. 199–206.

[27] Buyya, R., Beloglazov, A., and Abawajy, J. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308* (2010).

[28] Buyya, R., Ranjan, R., and Calheiros, R. N. Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *Proceedings of the ACM/IEEE International Conference on High Performance Computing & Simulation (HPCS 2009)* (2009), IEEE, pp. 1–11.

[29] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience 41*, 1 (2011), 23–50.

[30] Calheiros, R. N., Ranjan, R., De Rose, C. A., and Buyya, R. CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525* (2009).

[31] Caporuscio, M., Di Marco, A., and Inverardi, P. Model-based system reconfiguration for dynamic performance management. *Journal of Systems and Software (JSS) 80*, 4 (2007), 455–473.

[32] Chakrabarti, C., and Gaitonde, D. Instruction level power model of microcontrollers. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 1999)* (1999), IEEE, pp. 76–79.

[33] Changarti, P. *Xen Virtualization: A Practical Handbook*. Packt Publishing Ltd., 2007.

[34] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review 35*, 5 (2001), 103–116.

[35] Cheng, B. H., De Lemos, R., Giese, H., Inverardi, P., and Magee, J. *Software Engineering for Self-Adaptive Systems*. LNCS 5525. Springer, 2009.

[36] Cheng, B. H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, LNCS 5525. Springer, 2009, pp. 1–26.

[37] Chitsaz, H., Shaker, H., Zareipour, H., Wood, D., and Amjady, N. Short-term electricity load forecasting of buildings in microgrids. *Energy and Buildings 99* (2015), 50–60.

[38] Chiu, D., Stewart, C., and McManus, B. Electric grid balancing through lowcost workload migration. *ACM SIGMETRICS Performance Evaluation Review 40*, 3 (2012), 48–52.

[39] Chowdhury, S. A., Kumar, L. N., Imam, M. T., Jabbar, M. S. M., Sapra, V., Aggarwal, K., Hindle, A., and Greiner, R. A system-call based model of software energy consumption without hardware instrumentation. In *Proceedings of the Sixth International Green Computing Conference and Sustainable Computing Conference (IGSC 2015)* (2015), IEEE, pp. 1–6.

[40] Corral, L., Georgiev, A. B., Sillitti, A., and Succi, G. Can execution time describe accurately the energy consumption of mobile apps? An experiment in Android. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014)* (2014), ACM, pp. 31–37.

[41] Desmarais, R. J. *Adaptive Solutions to Resource Provisioning and Task Allocation Problems for Cloud Computing*. PhD Thesis, Department of Computer Science, University of Victoria, 2013.

[42] Dinita, R.-I., Wilson, G., Winckles, A., Cirstea, M., and Rowsell, T. A novel autonomous management distributed system for cloud computing environments. In *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society (IECON 2013)* (2013), IEEE, pp. 5620–5625.

[43] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification*. John Wiley & Sons, 2012.

[44] Duy, T. V. T., Sato, Y., and Inoguchi, Y. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing—Workshops and PhD Forum (IPDPSW)* (2010), IEEE, pp. 1–8.

[45] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 285–311.

[46] Fan, X., Weber, W.-D., and Barroso, L. A. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News* (2007), vol. 35, ACM, pp. 13–23.

[47] Foo, Y. W., Goh, C., Lim, H. C., Zhan, Z.-H., and Li, Y. Evolutionary neural network based energy consumption forecast for cloud computing. In *Proceedings of the International Conference on Cloud Computing Research and Innovation (ICCCRI 2015)* (2015), pp. 53–64.

[48] Friess, P. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, 2013.

[49] Ghezzi, C., and Tamburrelli, G. Predicting performance properties for open systems with KAMI. In *Architectures for Adaptive Software Systems*. Springer, 2009, pp. 70–85.

[50] GNU. Optimize options for GCC. `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`. [Online, last accessed May 2017].

[51] Goiri, I., Julia, F., Nou, R., Berral, J. L., Guitart, J., and Torres, J. Energy-aware scheduling in virtualized datacenters. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 2010)* (2010), IEEE, pp. 58–67.

[52] Hähnel, M., Döbel, B., Völp, M., and Härtig, H. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review 40*, 3 (2012), 13–17.

[53] Hamilton, J. Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR 2009)* (2009), pp. 1–8.

[54] Harmon, R. R., and Auseklis, N. Sustainable it services: Assessing the impact of green computing practices. In *Proceedings of the Portland International Conference on Management of Engineering & Technology (PICMET 2009)* (2009), IEEE, pp. 1707–1717.

[55] Heinrich, R. Architectural run-time models for performance and privacy analysis in dynamic cloud applications. *ACM SIGMETRICS Performance Evaluation Review 43*, 4 (2016), 13–22.

[56] Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N. ElasticTree: Saving energy in data center networks. In *Networked Systems Design and Implementation* (2010), vol. 10, pp. 249–264.

[57] Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[58] Hindle, A. Green mining: A methodology of relating software change to power consumption. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR 2012)* (2012), IEEE, pp. 78–87.

[59] Hindle, A., Wilson, A., Rasmussen, K., Barlow, E. J., Campbell, J. C., and Romansky, S. GreenMiner: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)* (2014), ACM, pp. 12–21.

[60] Hong, W.-C. Electric load forecasting by support vector model. *Applied Mathematical Modelling 33*, 5 (2009), 2444–2454.

[61] Horn, P. Autonomic Computing: IBM's perspective on the state of information technology. http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf, Oct. 2001. [Online, last accessed May 2017].

[62] Huebscher, M. C., and Mccann, J. A. A Survey of Autonomic Computing: Degrees, Models, and Applications. *ACM Compututing Surveys 40*, 3 (2008), 1–28.

[63] IBM. An architectural blueprint for autonomic computing. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.183.5437&rep=rep1&type=pdf, June 2006. [Online, last accessed May 2017].

[64] Jain, P., Bergen, A., Castaneda, L., and Müller, H. A. PALTask Chat: A personalized automated context aware web resources listing tool. In *Proceedings of the IEEE Ninth World Congress on Services (SERVICES 2013)* (2013), IEEE, pp. 154–157.

[65] Jiang, H.-P., Chuck, D., and Chen, W.-M. Energy-aware data center networks. *Journal of Network and Computer Applications* (2016), 80–89.

[66] Kansal, A., and Zhao, F. Fine-grained energy profiling for power-aware application design. *Newsletter ACM SIGMETRICS Performance Evaluation Review 36*, 2 (2008), 26–31.

[67] Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC 2010)* (2010), ACM, pp. 39–50.

[68] Kazman, R., Bass, L., Klein, M., Lattanze, T., and Northrop, L. A basis for analyzing software architecture analysis methods. *Software Quality Journal 13*, 4 (2005), 329–355.

[69] Keele, S. Guidelines for performing systematic literature reviews in software engineering. Tech. rep., EBSE-2007-01, 2007.

[70] Kephart, J. O., Chan, H., Das, R., Levine, D. W., Tesauro, G., Rawson III, F. L., and Lefurgy, C. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proceedings of the*

IEEE International Conference on Autonomic Computing (ICAC 2007) (2007), pp. 145–154.

[71] Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *IEEE Computer 36*, 1 (2003), 41–50.

[72] Kim, J. H., and Lee, M. J. *Green IT: Technologies and Applications*. Springer, 2011.

[73] Kim, Y.-J., Seok, J.-S., Lee, M. S., Kim, J.-S., and Jung, Y. Design of self-adaptive system observation overInternet of Things. *Advanced Science and Technology Letters (ASTL) 117* (2015), 165–171.

[74] Kliazovich, D., Bouvry, P., and Khan, S. U. GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing 62*, 3 (2012), 1263–1283.

[75] Koomey, J. G. Growth in data center electricity use 2006 to 2010: Analytics press report at the request of The New York Times. In *The New York Times*. 2011.

[76] Krintz, C., Wen, Y., and Wolski, R. Predicting program power consumption. Tech. rep., University of Santa Barbara (UCSB), California, 2002.

[77] Lago, P., Gu, Q., Bozzelli, P., et al. A systematic literature review of green software metrics. Tech. rep., Vreije Universiteit, The Netherlands, 2014.

[78] Laverick, M. Virtual machine (VM) consolidation ratios not the key to the virtual data centre. `http://www.computerweekly.com/news/1380058/VM-consolidation-ratios-not-key-to-virtual-data-centre`, 2010. [Online, last accessed May 2017].

[79] Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., and Nguyen, T. D. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)* (2011), ACM, pp. 22:1–22:12.

[80] Lee, E. A. The past, present and future of cyber-physical systems: A focus on models. *Sensors 15*, 3 (2015), 4837–4869.

[81] Li, D., and Halfond, W. G. J. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014)* (2014), IEEE, pp. 46–53.

[82] Li, D., Hao, S., Gui, J., and Halfond, W. G. J. An empirical study of the energy consumption of android applications. In *Proceedings of the IEEE International Conference onSoftware Maintenance and Evolution (ICSME 2014)* (2014), IEEE, pp. 121–130.

[83] Li, X., Bowers, C. P., and Schnier, T. Classification of energy consumption in buildings with outlier detection. *IEEE Transactions on Industrial Electronics 57*, 11 (2010), 3639–3644.

[84] Liu, H., Li, J., and Wong, L. A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. *Genome Informatics Series* (2002), 51–60.

[85] Liu, J., Zhao, F., Liu, X., and He, W. Challenges towards elastic power management in internet data centers. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2009)* (2009), IEEE, pp. 65–72.

[86] Liu, L., Wang, H., Liu, X., Jin, X., He, W. B., Wang, Q. B., and Chen, Y. Greencloud: a new architecture for green data center. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session* (2009), ACM, pp. 29–38.

[87] MacLeod, L., Storey, M.-A., and Bergen, A. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Proceedings of the IEEE 23rd International Conference on Program Comprehension (ICPC 2015)* (2015), IEEE, pp. 104–114.

[88] MacVittie, L. The dynamic data center: Cloud's overlooked little brother. `https://devcentral.f5.com/articles/the-dynamic-data-center-clouds-overlooked-little-brother`, 2012. [Online, last accessed May 2017].

[89] Mao, M., and Humphrey, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011* (2011), IEEE, pp. 1–12.

[90] Maren, A. J., Harston, C. T., and Pap, R. M. *Handbook of Neural Computing Applications*. Academic Press, 2014.

[91] Mazzucco, M., and Mitrani, I. Empirical evaluation of power saving policies for data centers. *ACM SIGMETRICS Performance Evaluation Review 40*, 3 (2012), 18–22.

[92] Medina, V., and García, J. M. A survey of migration mechanisms of virtual machines. *ACM Computing Surveys (CSUR) 46*, 3 (2014), 30.

[93] Mehta, H., Owens, R. M., Irwin, M. J., Chen, R., and Ghosh, D. Techniques for low energy software. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED 1997)* (1997), ACM, pp. 72–75.

[94] Meisner, D., Gold, B. T., and Wenisch, T. F. Powernap: Eliminating server idle power. In *Proceedings 14th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2009)* (2009), pp. 205–216.

[95] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. MIT press, 2012.

[96] Müller, H. A., Kienle, H. M., and Stege, U. Autonomic Computing: Now You See It, Now You Don't. design and evolution of autonomic software systems. In *International Summer School on Software Engineering (ISSE) 2006–2008*, LNCS 5413. Springer, 2009, pp. 32–54.

[97] Müller, H. A., Pezzè, M., and Shaw, M. Visibility of Control in Adaptive Systems. *Proceedings 2nd International Workshop on Ultra-Large-Scalesoftware-Intensive Systems ULSSIS 2008* (2008), 23–26.

[98] Müller, H. A., and Villegas, N. Runtime Evolution of Highly Dynamic Software. In *Evolving Software Systems*. Springer, 2013, pp. 229–264.

[99] Nathuji, R., and Schwan, K. Virtualpower: Coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 265–278.

[100] Neto, A. H., and Fiorelli, F. A. S. Comparison between detailed model simulation and artificial neural network for forecasting building energy consumption. *Energy and Buildings 40*, 12 (2008), 2169–2176.

[101] Niles, S., and Donovan, P. Virtualization and cloud computing: Optimized power, cooling, and management maximizes benefits. Tech. Rep. White Paper 118 Revision 4, 2008.

[102] Oreizy, P., Medvidovic, N., and Taylor, R. N. Runtime software adaptation: Framework, approaches, and styles. In *Companion of the 30th ACM/IEEE International Conference on Software Engineering (ICSE 2008)* (2008), ACM, pp. 899–910.

[103] Parashar, M., and Hariri, S. Autonomic computing: An overview. In *Unconventional Programming Paradigms*. Springer, 2005, pp. 257–269.

[104] Park, S., and Song, J. Self-adaptive middleware framework for Internet of Things. In *Proceedings of the 4th IEEE Global Conference on Consumer Electronics (GCCE 2015)* (2015), IEEE, pp. 81–82.

[105] Parunak, H. V. D., and Brueckner, S. A. Software engineering for self-organizing systems. *The Knowledge Engineering Review: Challenges in Agent-Oriented Software Engineering 30*, 4 (2011), 419–434.

[106] Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y.-M. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys 2011* (2011), ACM, pp. 153–168.

[107] Platon, R., Dehkordi, V. R., and Martel, J. Hourly prediction of a building's electricity consumption using case-based reasoning, artificial neural networks and principal component analysis. *Energy and Buildings 92* (2015), 10–18.

[108] Quang-Hung, N., Nien, P. D., Nam, N. H., Tuong, N. H., and Thoai, N. A genetic algorithm for power-aware virtual machine allocation in private cloud. In *Information and Communication Technology*. Springer, 2013, pp. 183–191.

[109] Rao, L., Liu, X., Xie, L., and Liu, W. Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2010)* (2010), IEEE, pp. 1–9.

[110] Rasmussen, K., Wilson, A., and Hindle, A. Green mining: energy consumption of advertisement blocking methods. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014)* (2014), ACM, pp. 38–45.

[111] Rosenblum, M. The reincarnation of virtual machines. *ACM Queue 2*, 5 (2004), 34:1–34:7.

[112] Rouse, M. Definition server consolidation. `http://searchdatacenter.techtarget.com/definition/server-consolidation`, 2007. [Online, last accessed May 2017].

[113] Runeson, P., and Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering 14*, 2 (2009), 131–164.

[114] Russell, J. T., and Jacome, M. F. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors (ICCD 1998)* (1998), IEEE, pp. 328–333.

[115] Sharma, N. K., and Reddy, G. Novel energy efficient virtual machine allocation at data center using genetic algorithm. In *Proceedings of the 3rd International Conference onSignal Processing, Communication and Networking (ICSCN 2015)* (2015), IEEE, pp. 1–6.

[116] Singh, A., Korupolu, M., and Mohapatra, D. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC 2008)* (2008), IEEE, p. 53.

[117] Sinha, A., and Chandrakasan, A. P. Jouletrack: a web based tool for software energy profiling. In *Proceedings of the 38th Annual Design Automation Conference (DAC 2001)* (2001), ACM, pp. 220–225.

[118] Söderström, T., and Stoica, P. *System Identification*. Prentice-Hall, 1988.

[119] Srikantaiah, S., Kansal, A., and Zhao, F. Energy aware consolidation for cloud computing. In *Proceedings of the Conference on Power Aware Computing and Systems (HotPower 2008)* (2008), USENIX, pp. 1–5.

[120] Sterritt, R., and Bustard, D. Towards an autonomic computing environment. In *2012 23rd International Workshop on Database and Expert Systems Applications* (2003), IEEE, pp. 699–699.

[121] Suganthi, L., and Samuel, A. A. Energy models for demand forecastinga review. *Renewable and Sustainable Energy Reviews 16*, 2 (2012), 1223–1240.

[122] Szvetits, M., and Zdun, U. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling 15*, 1 (2013), 1–39.

[123] Teodorescu, R., and Torrellas, J. Variation-aware application scheduling and power management for chip multiprocessors. In *ACM SIGARCH Computer Architecture News* (2008), vol. 36, IEEE, pp. 363–374.

[124] Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., and Lefurgy, C. Managing power consumption and performance of computing systems using reinforcement learning. In *Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS 2007)* (2007), ACM, pp. 1497–1504.

[125] Tiwari, V., Malik, S., and Wolfe, A. Compilation techniques for low energy: An overview. In *Proceedings of the IEEE Symposium on Low Power Electronics (ISLPED 1994)* (1994), IEEE, pp. 38–39.

[126] Tiwari, V., Malik, S., and Wolfe, A. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2*, 4 (1994), 437–445.

[127] Tiwari, V., Malik, S., Wolfe, A., and Lee, M. T.-C. Instruction level power analysis and optimization of software. In *Technologies for Wireless Computing*. Springer, 1996, pp. 139–154.

[128] Villegas, N. M. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD Thesis, Department of Computer Science, University of Victoria, 2013.

[129] Villegas, N. M., Tamura, G., Müller, H. A., Duchien, L., and Casallas, R. DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 265–293.

[130] Waltsgott, J., Götz, S., Fritzsche, R., Cech, S., and Wilke, C. State of the art: Hardware energy management. Tech. Rep. TUD-FI-11-06, Technische Universität Dresden, 2011.

[131] Wang, H., Huang, J., Lin, X., and Mohsenian-Rad, H. Exploring smart grid and data center interactions for electric power load balancing. *ACM SIGMETRICS Performance Evaluation Review 41*, 3 (2013), 89–94.

[132] Wang, X., Vasilakos, A. V., Chen, M., Liu, Y., and Kwon, T. T. A survey of green mobile networks: Opportunities and challenges. *Mobile Networks and Applications 17*, 1 (2012), 4–20.

[133] Wang, X., Yao, Y., Wang, X., Lu, K., and Cao, Q. Carpo: Correlation-aware power optimization in data center networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2012)* (2012), IEEE, pp. 1125–1133.

[134] Wickremasinghe, B., Calheiros, R. N., and Buyya, R. Cloudanalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications. In *Proceedings of the 24th IEEE International Conference onAdvanced Information Networking and Applications (AINA 2010)* (2010), IEEE, pp. 446–452.

[135] Widjaja, I., Walid, A., Luo, Y., Xu, Y., and Chao, H. J. Switch sizing for energy-efficient datacenter networks. *ACM SIGMETRICS Performance Evaluation Review 41*, 3 (2013), 98–100.

[136] Wikipedia. Control Theory; last accessed April 2016. `https://en.wikipedia.org/wiki/Control_theory`. [Online, last accessed May 2017].

[137] Wikipedia. Signal Processing; last accessed April 2016. `https://en.wikipedia.org/wiki/Signal_processing`. [Online, last accessed May 2017].

[138] Wilke, C., Götz, S., Cech, S., Waltsgott, J., and Fritzsche., R. Aspects of softwares energy consumption. Tech. Rep. TUD-FI-11-04, Technische Universität Dresden, 2011.

[139] Wilke, C., Götz, S., Reimann, J., and Assmann, U. Towards model-based energy testing. In *Model Driven Engineering Languages and Systems (MODELS 2011)* (2011), J. Whittle, T. Clark, and T. Kühne, Eds., LNCS 6981, Springer, pp. 480–489.

[140] Wu, W., Du, W., Zhou, H., Zhong, J., and Guo, Z. An optimization model on virtual machines allocation based on radial basis function neural networks. *International Journal of Hybrid Information Technology 8*, 6 (2015), 299–308.

[141] Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *Cloud Computing (CLOUD), 2010* (2010), IEEE, pp. 91–98.

[142] Yin, R. K. *Case Study Research: Design and Methods*. Sage Publications, 2013.

[143] Zhang, C., Hindle, A., and German, D. M. The impact of user choice on energy consumption. *IEEE Software 31*, 3 (2014), 69–75.

[144] Zhang, Q., Zhu, Q., and Boutaba, R. Dynamic resource allocation for spot markets in cloud computing environments. In *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC 2011)* (2011), IEEE, pp. 178–185.

[145] Zhang, Z., Guan, Q., and Fu, S. An adaptive power management framework for autonomic resource configuration in cloud computing infrastructures. In *Proceedings of the 31st IEEE International Performance Computing and Communications Conference (IPCCC 2012)* (2012), IEEE, pp. 51–60.

[146] Zhuravlev, S., Saez, J. C., Blagodurov, S., Fedorova, A., and Prieto, M. Survey of energy-cognizant scheduling techniques. *IEEE Transactions on Parallel and Distributed Systems 24*, 7 (2013), 1447–1464.