Fall 2015

Distributed control system for demand response by servers

Joseph Edward Hall
University of Iowa

DISTRIBUTED CONTROL SYSTEM FOR DEMAND RESPONSE BY SERVERS

by

Joseph Edward Hall

A thesis submitted in partial fulfillment of the
requirements for the Master of Science
degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

December 2015

Thesis Supervisor: Associate Professor Raghuraman Mudumbai

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Joseph Edward Hall

has been approved by the Examining Committee for the thesis requirement for the Master of Science degree in Electrical and Computer Engineering at the December 2015 graduation.

Thesis Committee: _____
Raghu Mudumbai, Thesis Supervisor

_____
Soura Dasgupta

_____
John Kuhl

ACKNOWLEDGEMENTS

Thanks especially to: Dr Raghu Mudumbai and Dr Soura Dasgupta, for seeing promise in my abilities and for supporting my research and education professionally and financially; Josiah McClurg to whom is due most of the credit for the research presented; Joan Hall, Ann Tudor, and Mama Achieng for their prayers which have helped me to leap through what has been for me a most terrifying and arduous hoop; and Jessica Hall for her friendship, criticism, encouragement and love.

ABSTRACT

Within the broad topical designation of smart grid, research in demand response, or demand-side management, focuses on investigating possibilities for electrically powered devices to adapt their power consumption patterns to better match generation and more efficiently integrate intermittent renewable energy sources, especially wind. Devices such as battery chargers, heating and cooling systems, and computers can be controlled to change the time, duration, and magnitude of their power consumption while still meeting workload constraints such as deadlines and rate of throughput. This thesis presents a system by which a computer server, or multiple servers in a data center, can estimate the power imbalance on the electrical grid and use that information to dynamically change the power consumption as a service to the grid. Implementation on a testbed demonstrates the system with a hypothetical but realistic usage case scenario of an online video streaming service in which there are workloads with deadlines (high-priority) and workloads without deadlines (low-priority). The testbed is implemented with real servers, estimates the power imbalance from the grid frequency with real-time measurements of the live outlet, and uses a distributed, real-time algorithm to dynamically adjust the power consumption of the servers based on the frequency estimate and the throughput of video transcoder workloads. Analysis of the system explains and justifies multiple design choices, compares the significance of the system in relation to similar publications in the literature, and explores the potential impact of the system.

PUBLIC ABSTRACT

Traditionally, in the electrical power grid, generators must change the amount of power they generate in order to match the amount of power used by consumers (loads). In such a system, some generators are continually ramping up and down their output in order to match demand which is not fuel efficient. Moreover, wind power generators compound the variability of the generation-load imbalance due to the intermittent nature of wind power. Demand response proposes that electrical loads can participate in generation-load balancing by changing how much power they consume in order to allow fuel powered generators to run closer to a more efficient constant rate. Computer servers in data centers are good candidates as demand responsive loads due to their ability to quickly change their power consumption and the fact that many of their computational jobs are deferrable, meaning they can be processed later. Distributed demand response is an approach that allows each device in a network to make its own control decisions while at the same time acting in collaboration with other devices to produce a desired result on the aggregate. This thesis presents an experimental demonstration of a distributed control algorithm for servers to participate in demand response.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Demand Response

The electrical power grid consists of interconnected paths of transmission wires which connect electrical generators (power plants, solar panels, wind farms, etc.) and electrical loads (lights, computers, heaters, conveyor belts, etc.). At the most basic level, the power flow of the transmission wires must be balanced such that the power generated equals the power consumed by the loads. Traditionally, power in the grid is supplied by base load generators (typically coal-fired or nuclear) which are highly economically efficient, but cannot adjust their generations dynamically, and are therefore supplemented by spinning reserves of peaking generators (typically based on natural gas or diesel). The latter are much more expensive, but are needed for the load-following function. In recent years, the increased penetration of wind and other renewable energy sources has led to increased peaking generation requirement to keep up with variability in generation as well as in load. This has posed a huge challenge to grid operators and may limit the amount of renewable generation that can be economically integrated into the grid. Costs of not meeting this challenge include: "approximately 25 TWh of wind energy [...] curtailed (idled) in the U.S. [in 2010] to keep the off-peak grid energy price from frequently going negative. That is about equal to the energy in 700 million gallons of gasoline just being thrown away" [1]; and in the UK, the National Grid paid wind farms £32 million in 2013

to remain idle and £3 million in a single day on October 26, 2014 [2]. Additionally, the ramping up and down of the peaking generators is itself less efficient use of fuel than if those generators would operate at steady-state or with slower ramp rates, although the costs associated with integration of renewables are extremely difficult to quantify [3].

Integration costs of utilizing intermittent sources of renewable energy generation, such as wind and solar, include reduction in thermal generation efficiency due the need for generators to balance power generation and load. Intermittent sources both generation and load lead to inefficiencies in the overall grid due to increased generator cycling and less optimal unit commitment due to prediction errors [3]. Demand response (DR) or load-side management (LSM) encompasses diverse proposals and implementations of how electrical workloads can adjust the magnitude and time of their power consumption to match set points designed to increase electrical grid stability and efficiency. The simple proposal is to delay or store-up deferrable demand in order to utilize intermittent sources of power generation most efficiently.

The promise of demand response is that by adaptively controlling loads in such a way that it offsets the fluctuations in the uncontrollable load and generation units, the variability of the aggregate effective load that need to be serviced by conventional generators can be minimized and thereby also the need for increased peaking reserves. For example, when the lights turn on at a football stadium, the generators have to work a little harder to supply the extra power consumed by the lights. When the lights turn off, the generators have to decrease their output to maintain the

balance. Demand response is a concept which invites loads to participate in this power balancing. In demand response, a group of loads might be able to turn-on/off and speed-up/slow-down while still performing their jobs adequately on average. These types of loads can *defer* some of the work they need to do to a later time, or they can speed up and do the work early. Such deferrable loads can then consume more or less power to help balance the total power flow on the grid. While some authors might choose to differentiate between demand response (DR), load-shedding, demand-side management, and possibly other terms, the remainder of this document will define and conflate all with the term, *demand response*:

> the intentional or designed increase or decrease in power consumption of an electric load in response to some predefined input or control signal for the purpose of affecting aggregate load-balance on an electrical power grid.

1.2 Methods of Demand Response

Demand response has been an extremely active area of research in recent years and many methods for its implementation have been proposed and studied in the literature. One popular category of previous work in this area is focused on discovering optimal adaptive algorithms [4], i.e. find a procedure to adjust the power consumption of flexible loads over time to maximize a suitable performance metric while satisfying device constraints (typically energy and deadline requirements).

The big problem with this approach is that basically, *it is not at all obvious*

just what to optimize i.e. there is no widely accepted performance metric that is both physically meaningful for a sufficiently large class of flexible loads, and is amenable to analytical optimization techniques.

There is another popular class of related work that uses dynamic pricing to provide incentives for loads to adjust their consumption [5, 6]. This approach is attractive for its generality, decentralized functioning and the minimal communication (pricing signals) requirement. However, these methods rely on the assumption that markets are able to achieve optimal outcomes for practically useful device models and utility functions, which in turn relies on the ability of profit-maximizing consumers to figure out how to adapt their power consumption to achieve their goal. In other words, pricing-based methods may indeed offer an elegant solution to the demand-response problem from the perspective of the grid operator, but they do so by delegating the most difficult part of it to end-users.

1.3 Data Center Demand Response

Demand response resources can participate as ancillary services in power systems depending upon their response time, duration, and capacity. For example, in the US system, frequency regulation services must be dispatchable in less than 30 seconds and last for seconds to minutes and regulating reserves must respond within 4 seconds to 5 minutes and last for minutes [7]. At the other extremes, The Electric Reliability Council of Texas (ERCOT)'s emergency demand response program in 2012 required resources to respond within 30 minutes [7]. According to the Midcontinent

Independent System Operator (MISO), only a relatively small number of current demand response loads can be used for regulation reserve, due to ramp-rate restrictions of large loads and the ineffectiveness of small loads; for example demand response resources with controllable loads must have at least 1 MW capacity to be included in network model planning [8]. For example, slowing down production at a factory may have a significant impact on the grid, but it can't be done in just a few seconds. By a similar token, it may be possible to rapidly control the power consumption of a single water heater – but doing so will not have a significant impact on the grid. Data centers are uniquely positioned as particularly versatile controllable load resources in that they use a lot of power [9], and that power is controllable at fast time scales – on the order of seconds. The motivation for demand response at a fast timescale is shown in Figure 1.1, which is data from the MISO website [10], showing the difference between scheduled generation and actual generation. Neglecting the effects of prediction error which would make the effect more pronounced, it is clear that large power mismatches are common, and vary rapidly in time. If demand response were implemented on a fast timescale, the average of 250 MW of deficit power could easily have been avoided by rescheduling loads to draw from the average 500 MW of excess power. Moreover, there would still be 250 MW of excess power that could be put to useful work, were there loads which would benefit from it.

Several studies have proposed ways to manage computers and data centers utilize intermittent renewable energy generation such as wind and solar in data center infrastructure. Most of these studies model two types of server workloads: batch

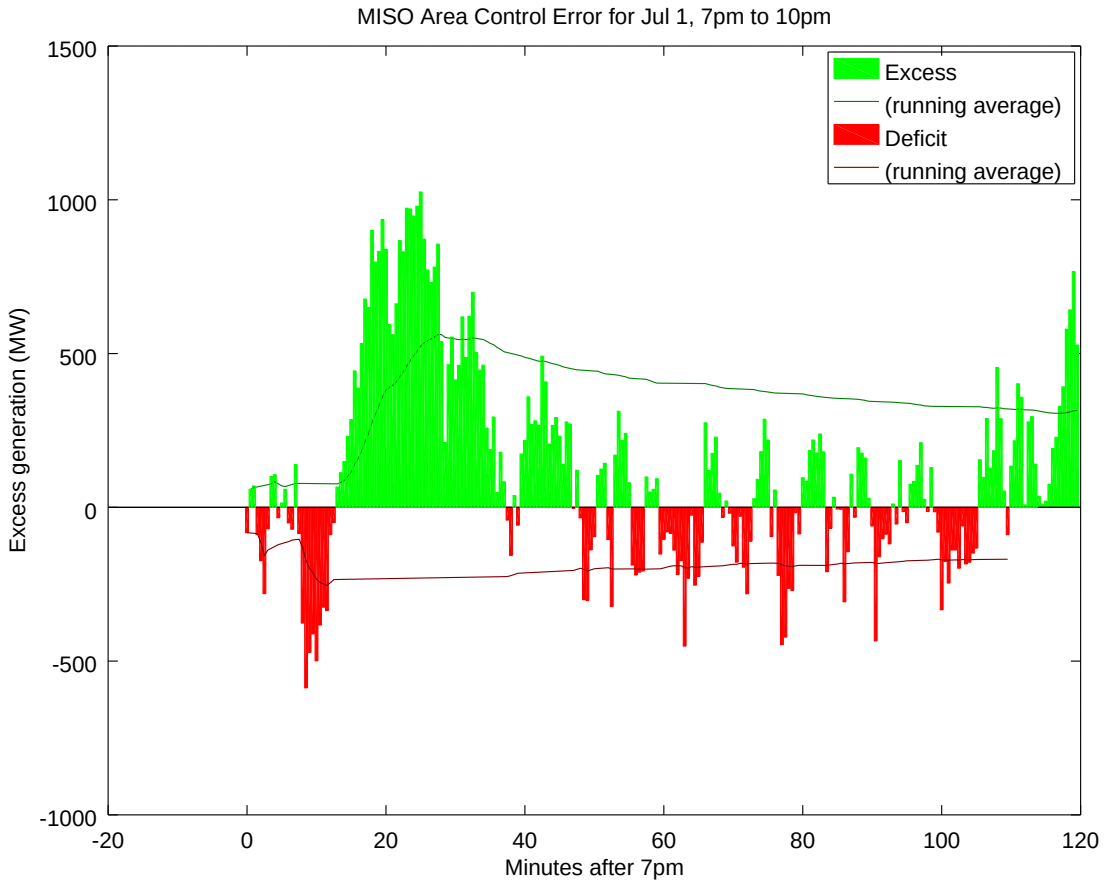


Figure 1.1: Opportunity for Saving Hundreds of Megawatts

processes which can be deferred on the order of minutes and hours and web-server requests which can only be deferred by milliseconds. Most of these studies utilize highly particular job-schedulers and optimization techniques to match server or data center power consumption with renewable energy generation. Some use cost minimization algorithm with pricing [11] or in a self-sustaining data center model collocated with a finite supply of renewable generation [12]. Others use collocated batteries as additional storage devices in a data center [13, 14]. Still others use day-ahead pricing signals or other stochastic devices for predicting server workload or load imbalance

on the grid [15, 16].

1.4 Our Contribution

We propose a radically simpler method of effecting demand-response which prioritizes matching server power consumption to intermittent load imbalance as an ancillary service to the grid. We do not include service-level agreements, predictive analysis, exogenous storage devices, or centralized optimization schedulers in our algorithm. We propose a radically different approach to demand-response. Instead of looking for a solution that is guaranteed to be optimal for some specific performance metric and device model, we take a minimalistic approach. We describe a simple adaptation scheme that can be intuitively derived from the definition of the demand response problem. We do not require precise estimates of future loads, or knowledge of the statistics of load variations. The adaptation scheme is designed for distributed implementation at the loads without any global coordination, and the only communication required is knowledge of the aggregate load in the grid in the previous time period (or more precisely the deviation of the aggregate load from the desired operating point). We presented this idea in [17] using simulated case study of electric vehicle battery chargers as deferrable loads for distributed demand response. This present work adapts the idea to deferrable computational loads in servers with a hardware based testbed using real servers.

The central proposition of the algorithm is that multiple distributed loads can contribute to a control loop to track a setpoint as an aggregate. Further, loads are

able to *jostle* their share of the power in small perturbations related to their deadlines or other constraints in order to readjust their portion of the available resource. The present work offers no guarantees of optimality or convergence. However, an elegant proof of the proposition in the case of economic dispatch in a network of power generators is given in [18]. Development of the demand response algorithm to accommodate a similar proof is an interesting avenue for future research, however the proposition is the same in each case. In the generator dispatch case, a generator jostles to generate its relative share of the load according to its marginal cost of generation. In [17], the battery chargers jostle to consume their share of the load according to their charge state and the time remaining until their deadline. In the present work, the computational jobs jostle to consume their share of the load according to the state of and constraints on their workload throughput.

The term *jostle*, used above, conveys a very simple and intuitive process in which multiple actors, working towards similar goals, push each other around in order to find a place acceptable to some, most, or all depending upon the relative strength of each. The algorithm for *jostling* of jobs in a server or server cluster proposed in the present work is to add a small bias term to the error as shown in Equation 1.1, where $P_i[k]$ is the power consumed by the server or cluster on which job i is running, $e[k]$ is the global error, A is the gain of the plant, and $\beta_i(x_i[k])$ is the bias added by job i and is a function of variable, $x_i[k]$ which describes the state of the job's throughput in relation to its constraints.

$$\text{DR Server Algorithm:} \quad P_i[k+1] = P_i[k] + AK_i(e[k] + \beta_i(x_i[k])) \tag{1.1}$$

Therefore jobs approaching their deadline or other constraint which requires them to obtain more share of CPU utilization need only to increase their bias term. Similarly jobs which are running too fast (e.g. are in danger of overwriting a buffer) need only to decrease their bias term. Jobs which are comfortably far from any constraints will leave their bias term equal to zero. This preliminary discussion is intended only as a simple foretaste of algorithm proposed. Further discussion of this dynamic and how it is implemented occurs particularly in Sections 3.3 and 3.4.

1.5 Outline for Rest of the Thesis

This thesis presents a description of testing and analysis of the algorithm proposed above in Equation 1.1. In Chapter 2, Sections 2.1 and 2.2 describe the physical mechanisms explaining how it is possible to modulate the power consumption of a server, Section 2.3 compares multiple software interfaces, or 'control knobs' from which to control the power modulation, Section 2.4 describes an integral controller design for a particular 'control knob', and Section 2.5 adapts the integral controller to implement distributed control. Chapter 3 provides a demonstration of the distributed control of servers running multiple constrained jobs to track a global setpoint to minimize the generation-load imbalance. Sections 3.1 and 3.2 describe the demo problem statement and the methods used for data collection. Section 3.3 describes the specific implementation of the proposed algorithm in Equation 1.1 in the context of the demo and Section 3.4 discusses the results. Chapter 4 concludes the thesis by summarizing results and offering suggestions for future work.

CHAPTER 2
CONTROL OF POWER CONSUMPTION IN COMPUTERS FOR DEMAND RESPONSE

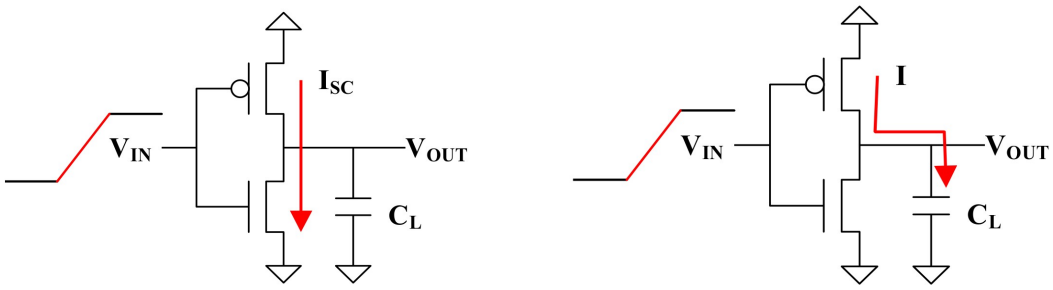
This chapter discusses and compares the methods by which it is possible to control the power consumption of servers and discusses in detail a simple controller which is implemented in the demo.

2.1 Mechanisms of Power Dissipation in Computer Circuits

The ability of computers to ramp power consumption depends most heavily on the proportional relationship between power and the effective frequency of the system. Equation 2.1 models the overall power consumption of a microprocessor as the sum of dynamic and static power dissipation [19].

$$P = ACV^2 f + V I_{leak} \tag{2.1}$$

Here, A is the fraction of gates actively switching, C is the capacitive load of the gates, f is the operating frequency, V is the supply voltage, and I_{leak} is the CMOS leakage current. Equation (2.1) shows that power consumption can be dynamically affected by scaling the supply voltage, the frequency, and the fraction of actively switching gates. The circuits in Figure 2.1 show two mechanisms of dynamic power dissipation during each occurrence of a logical switch in the input signal. Short-circuit current is dissipated by transistors due to finite-slope input signals (when transitioning between High/Low states, transistors are both partially ON and thus allow current to flow from the supply-rail to ground as shown in Figure 2.1a. Further, as shown in Figure 2.1b,



(a) Short-Circuit Power Dissipation (b) Capacitive Power Dissipation

Figure 2.1: Two Types of Logic Switching Power Dissipation [20]

current flows to charge the capacitive load when the logic switches from Low to High. Thus, each change in the input signal to a logic gate dissipates both capacitive and short-circuit current. The rate of this dissipation therefore directly depends upon the operating voltage, V, and frequency, f, and the fraction of actively switching gates, A.

2.2 Methods of Power Control in Computers

Therefore there are two basic mechanisms by which to dynamically adjust the power consumption of a computer: dynamic voltage frequency scaling (DVFS) and idle cycle injection (ICI). The former adjusts V and f in Equation 2.1 and the latter adjusts A.

2.2.1 Dynamic Voltage Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) adjusts the voltage and frequency together in discrete preset level pairs. Voltage needs to be adjusted with

frequency because in order to maintain an adequate rise-time for different switching frequencies, the supply voltage must also scale in proportion with the frequency. Methods for DVFS are generally implemented automatically by modern processors which known as P-states in Intel processor architectures [21]. Recently, Advanced Micro Devices (AMD) has developed Adaptive Voltage Frequency Scaling (AVFS) which uses a closed-loop system to adjust the voltage to a level appropriate for the frequency and temperature states of the processor rather than using fixed "worst-case-scenario" voltage-frequency pairs as in DVFS [22].

Modern operating-system-level requests to change the operating voltage-frequency state of modern Intel processors is merely a hint to the on-chip power management infrastructure which can be overridden by thermal management constraints [23, 24]. P-states are chosen automatically by most power management software according to CPU utilization [21]. Some papers like [25,26] have successfully applied the userspace governor of the Linux cpufreq kernel module to implement energy-aware DVFS on older Intel processors. We reproduce this method in Section 2.3.1.

2.2.2 Idle Cycle Injection

Idle cycle injection (ICI) is fundamentally forcing processor cores and/or packages into idle states for a variable number of clock cycles [27], also described as hardware duty cycling (HDC) [23, see 14.5 Vol. 3B]. This involves shutting down the system clock to different components and/or cores and reducing and/or cutting power to components and/or cores of the processor for a variable number of clock

cycles, different combinations of these states are called C-states in Intel processor architecture [21, 23]. Availability of different C-states and access to controlling them without a custom OS is platform specific. We implement one such API in Section 2.3.4.

In addition to directly forcing hardware level idle states, ICI can be implemented at the process or OS levels by setting processes to sleep, thus inviting whatever lower-level power management (PM) features designed to engage based upon CPU utilization to do so. This scheme is rather agnostic regarding what power management features exist, and trusts them to bring the machine into as efficient an operating state as possible for the given amount of work given to the processors by injecting variable length sleep states into processes or into the OS itself. Such schemes are presented in Sections 2.3.3, 2.3.5, and 2.3.6.

2.3 Empirical Comparison of Power Control Architectures

Previous research has used a variety of different software techniques to modify the power profile (or some function thereof, such as temperature) of computers. However, there are no previous papers which specifically evaluate the suitability of different fundamental strategies for modifying the power profile of servers and groups of servers. It should be noted that while much of the discussion below applies specifically to Intel processors, other manufacturers often include similar technologies on server-oriented processors. Additionally, Intel has long held the clear majority market share in server processors (98% in 2014) [28], so the discussion is immediately appli-

cable in the majority of existing data centers. In hopes of fostering future research into this area, this paper presents an empirical evaluation of several software techniques ("control knobs") that can be utilized to quickly scale the power consumption of modern servers for data center demand response.

The experimental setup is a Dell PowerEdge R320 server with an Intel Xeon E5-2400 series processor, running Ubuntu 12.04 with the 3.16.0-41-generic Linux kernel. We've instrumented the server with a current sensor for validation of the on-chip power usage estimate obtained through the software interface. During each test, the Linux stress program [29] (running repeated square root operations) was used as the workload, with the exception of Section 2.3.5, which used a simplified version of the stress program, modified to include custom delays. Each test ran for 15 seconds of data at each control setpoint, and ran each control setpoint twice for comparison. The package power was estimated using the RAPL interface, and was sampled once ever 150 milliseconds. The possible power range is 7W (completely idle) - 35W (maximum CPU utilization).

The plots below are a standard box plot, with the box representing the interquartile range (IQR) for the power at each control input value and the whiskers representing the minimum and maximum values. The outliers (greater than 10 times IQR) are shown in red.

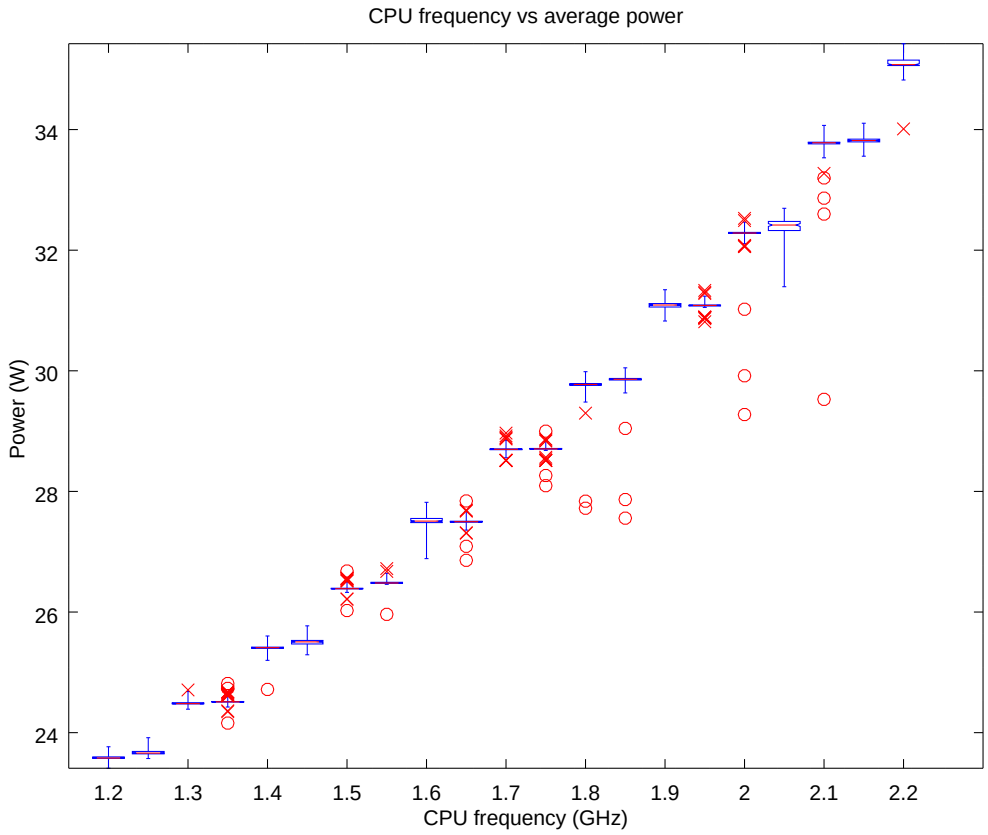


Figure 2.2: DVFS Using ACPI-CPUFREQ Driver with Userspace Governor

2.3.1 Direct Dynamic Voltage-Frequency Scaling

The acpi-cpufreq driver is the predecessor to the more recent intel_pstate driver used in modern servers. Unlike the more recent driver, which in the default configuration offloads the majority of power management to the processor itself, the acpi-cpufreq driver uses software to suggest different voltage-frequency pairs called power states (P-states) through the ACPI interface. As mentioned earlier, Intel documentation states that these requests may be overruled by the internal thermal management logic of the processor. However, it has been the experience of the authors that the

requested P-state is usually entered unless it is thermally unsafe to do so.

The acpi-cpufreq driver provides of power management policies that the operating system uses to determine which P-state to request and when to do so. The userspace governor allows the user to specify a target frequency, and the driver will determine the corresponding P-state to request. The benefits and drawbacks of this method are shown in Figure 2.2. The main advantage is that the variance of the power is very low (notice how tight the box plot is). The downsides of this technique are two. First, the range of power levels is relatively small – starting at 24 watts, over twice the idle power consumption. This is due to the operating system never placing the processor in a sleep state. Second, the number of unique power levels available is quite small (only twelve on the processor used in this study). This is due to the processor having a limited number of P-states available for DVFS.

2.3.2 Proprietary On-chip Power-Limiting

Intel published results from an early version of RAPL in 2010 which was capable of finely shaping the power consumption of memory by rapidly switching between adjacent memory bandwidth states [30].

This technique has the advantage of providing very tight control over a wide range of power values between around 24 watts and the maximum power. The primary disadvantage, as 2.3 shows, is that similar to the cpufreq method, RAPL does not allow power to be capped across the full range of server power consumptions. Moreover, because not all processors (even within the Intel Xeon family) support

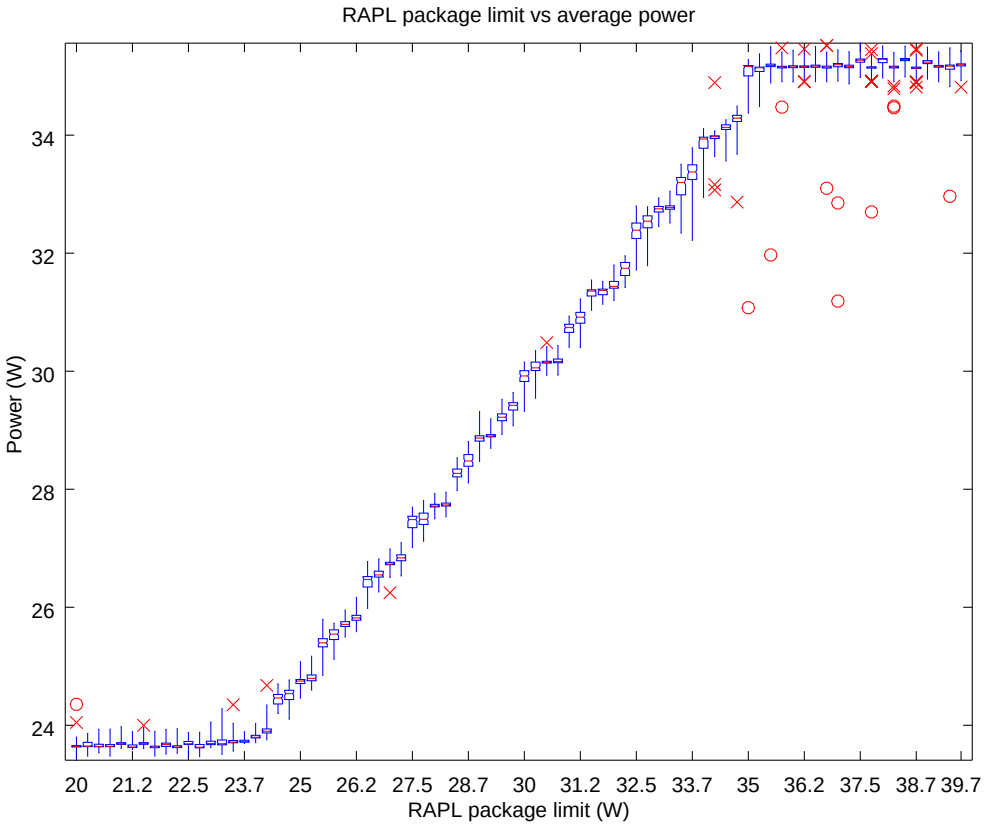


Figure 2.3: Intel RAPL Interface

RAPL power limiting features, the technique may not be an option for many server configurations.

2.3.3 Hypervisor Capping

The hypervisor method of controlling server power consumption was inspired by [31] who used VMware vSphere 5.1 ESXi hypervisor. The Xen Project hypervisor [32] is a type-1 (also called "baremetal") hypervisor, which means that it runs architecturally "beneath" any guest operating systems that it is coordinating. While VMWare remains the industry de facto standard, Xen has a wide deployment (in

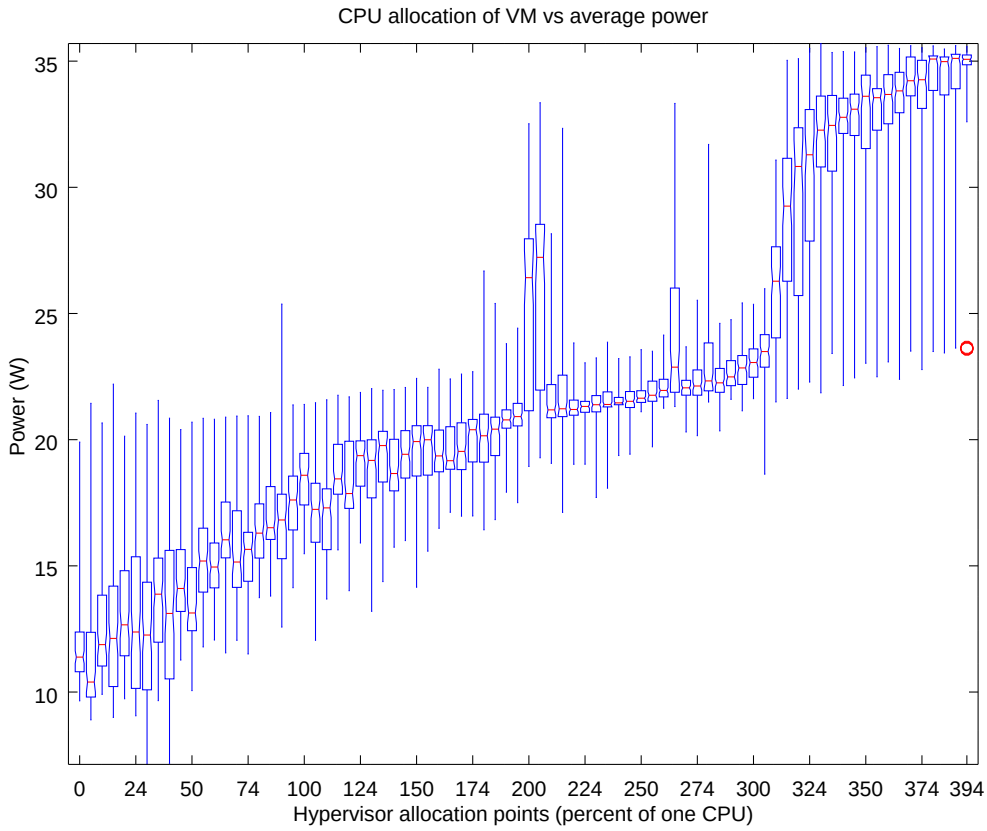


Figure 2.4: Hypervisor (Xen) CPU Capping

2012, it held the second-largest market share of any virtualization technology [33]) and has the advantage of being free and open source software. Xen contains a variety of resource contention management features, including a processor utilization capping mechanism called sched-credit. The current implementation uses a priority queue and an accounting thread to ensure that the virtual CPUs assigned to the virtual machines do not exceed their CPU utilization cap.

Figure 2.4 shows results gathered by cpu capping a paravirtualized Linux guest VM allocated four virtual CPUs and running four threads of the stress program. One advantage of this technique is that it offers a full range of power values between idle

maximum power. The reason it is able to achieve low power values is that hypervisor is allowed to transition the processor to a sleep state for a portion of each 30 ms accounting period (when the guest VM has exceeded its allotted allocation credits). The primary downside of this technique is that it is difficult to tightly control the power consumption at power levels corresponding to hypervisor allocation settings which cause the number of physical cores allocated to the guest operating system to oscillate. Note, for example, the increased variance at 200 allocation points – where the guest operating system is alternately being allocated between two and three physical cores. Similarly, there is another jump in power variance at 300 allocation points in a highly non-linear transition.

2.3.4 Intel PowerClamp

The Intel PowerClamp driver was developed to force the processor to enter certain sleep states through synchronized idle injection across all threads [34]. The driver implements a closed loop control with limits on the amount of idle time that can be injected.

While the system is indeed entering sleep states during the lower-power portions of Figure 2.5, the fact that idle injection is limited to 50% idle time means that the power of the fully-loaded system still cannot be reduced below about 17 W. Moreover, compared to the hypervisor solution, there is a relatively large spread in power at every control setpoint.

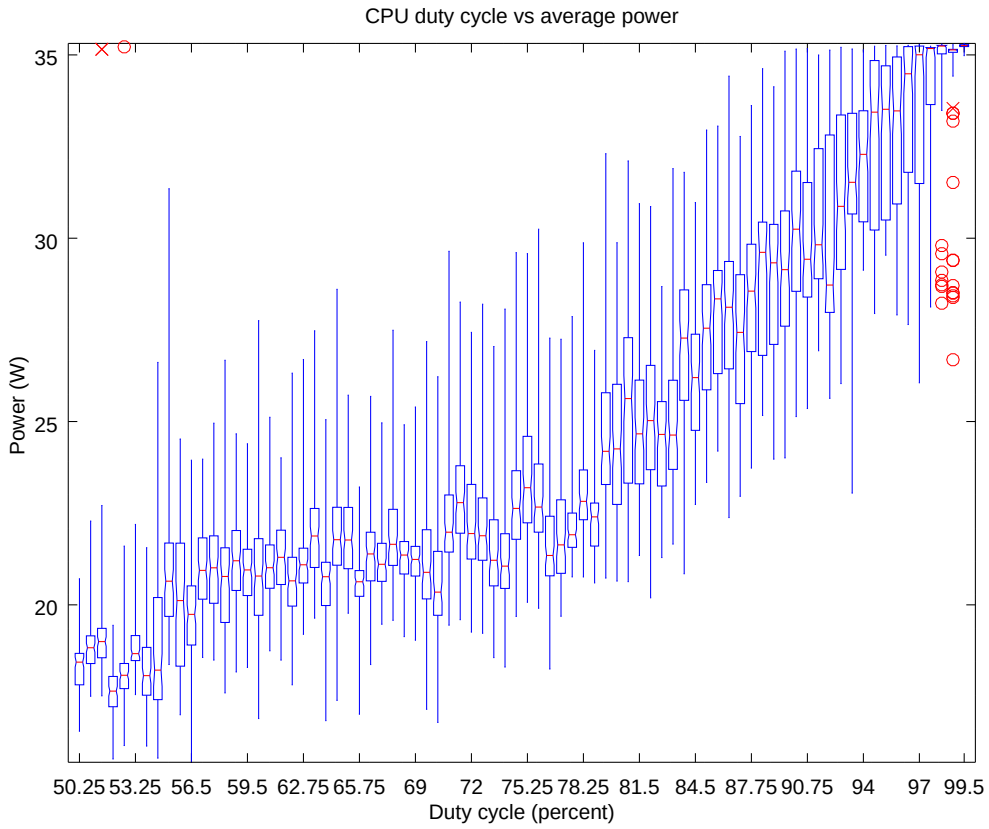


Figure 2.5: Inserting Delays at Kernel Level with Intel Powerclamp Driver

2.3.5 Power-aware Workload

The authors have provided two custom software solutions for comparison with the aforementioned power-shaping technologies. The first of these is a baseline "best-case scenario" workload which has been specifically tailored to use a certain percentage of each processor. Each of the four threads in the workload uses POSIX timers and the usleep() command to insert delays between blocks of 50000 square root operations. The amount of delay was calculated to correspond to a specified CPU utilization. An ad hoc procedure was used to tune the number of square root operations and the number of threads to achieve the maximum dynamic range of

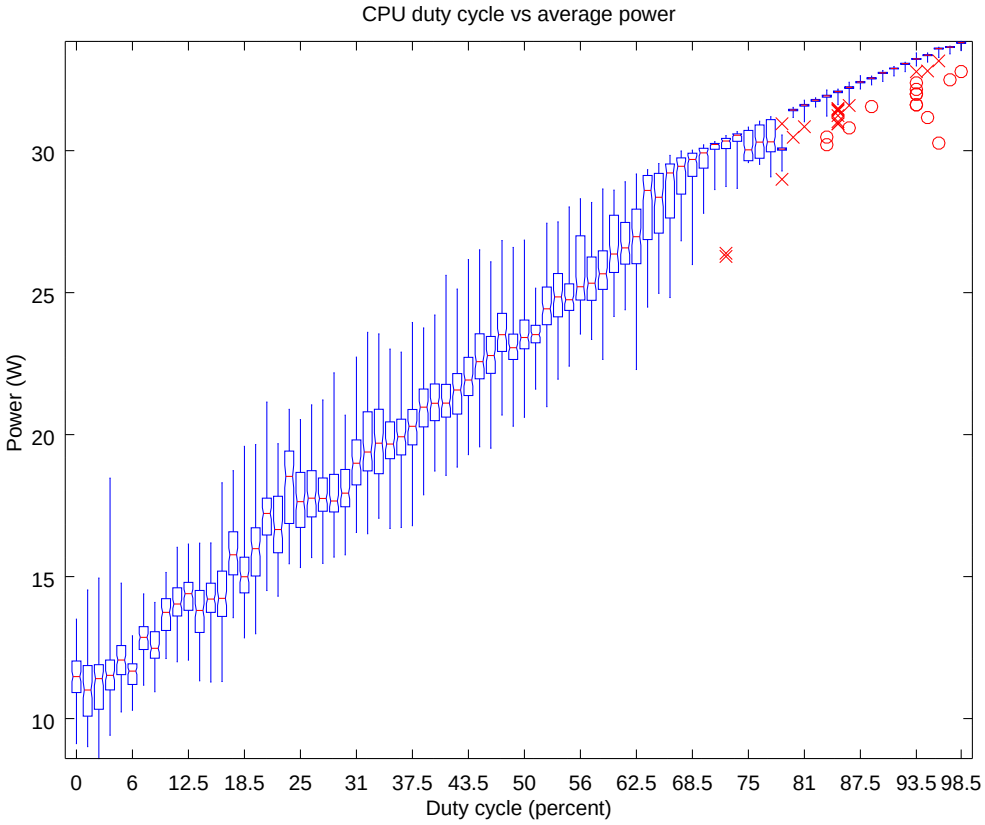


Figure 2.6: Inserting Delays at Application Level with usleep()

power.

The benefit of such a workload is clear. Not only is there a roughly linear correspondence between CPU utilization and power consumption, but the variance at each setpoint is relatively small, and the full dynamic range of power consumption is achievable. The downside of this technique is that it is extremely invasive – requiring a rewrite of all CPU-intensive software on the system.

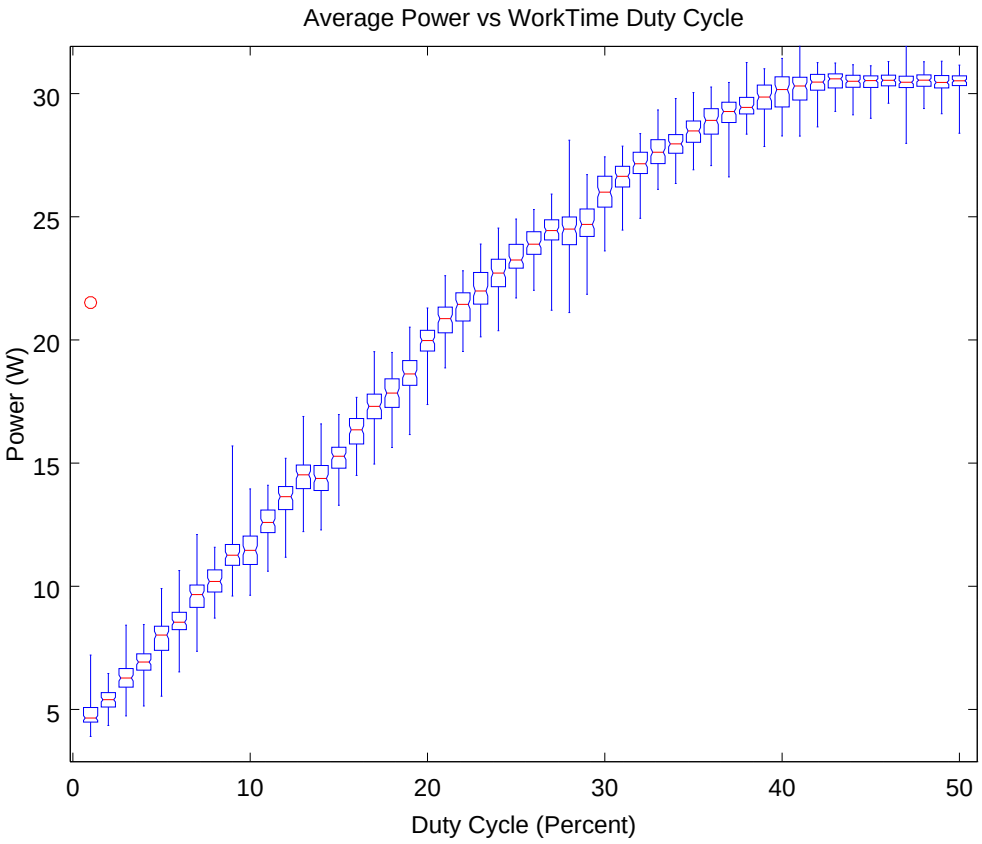


Figure 2.7: Inserting delays with SIGSTOP and SIGCONT

2.3.6 Custom Userspace Software

To provide a comparison and a more general method for inserting delays in the user space, the authors designed a tool which modulates the CPU utilization of processes using the Linux SIGSTOP and SIGCONT signals in a manner inspired by the cpulimit [35] program. Results are shown in Figure 2.7. Because this program is running in userspace, it was expected to have significantly increased power variance compared to the powerclamp driver. However, the power variance of the method is less than all others excepting RAPL and DVFS. A large range of available power settings

is available with fine granularity, by changing the ratio (duty cycle) of sleep times between each SIGSTOP and SIGCONT signal. A further advantage of such a tool is that it can be run in the background of any POSIX operating system, without the need for accessing custom MSRs, modifying hypervisor power management policies, or kernel drivers.

2.4 Power Consumption Controller Design

2.4.1 Integral Controller Design

PI Parameters		K	T_i
Equation*		$0.9/RL$	$3L$
Exp. Values†	DAQ	0.03	0.51
	RAPL	0.05	0.42
Max. Values‡	DAQ	0.013	0.6
	RAPL	0.019	0.6

* Equations are from [36].
† R and L as shown in Figure 2.10.
‡ R and L given as maximum possible.

Table 2.1: Controller Parameters from the Ziegler-Nichols Step-Response Method

Figure 2.8 demonstrates that the relationship between the server power consumption less 36 watts and the duty-cycle of the open-loop signal-sleep controller is approximately linear in the range of 36 - 67 watts and 0.001 - 0.4 duty, s.t. $y - 36W = 80W * Duty$, where 36W is the minimum power at idle and y is the

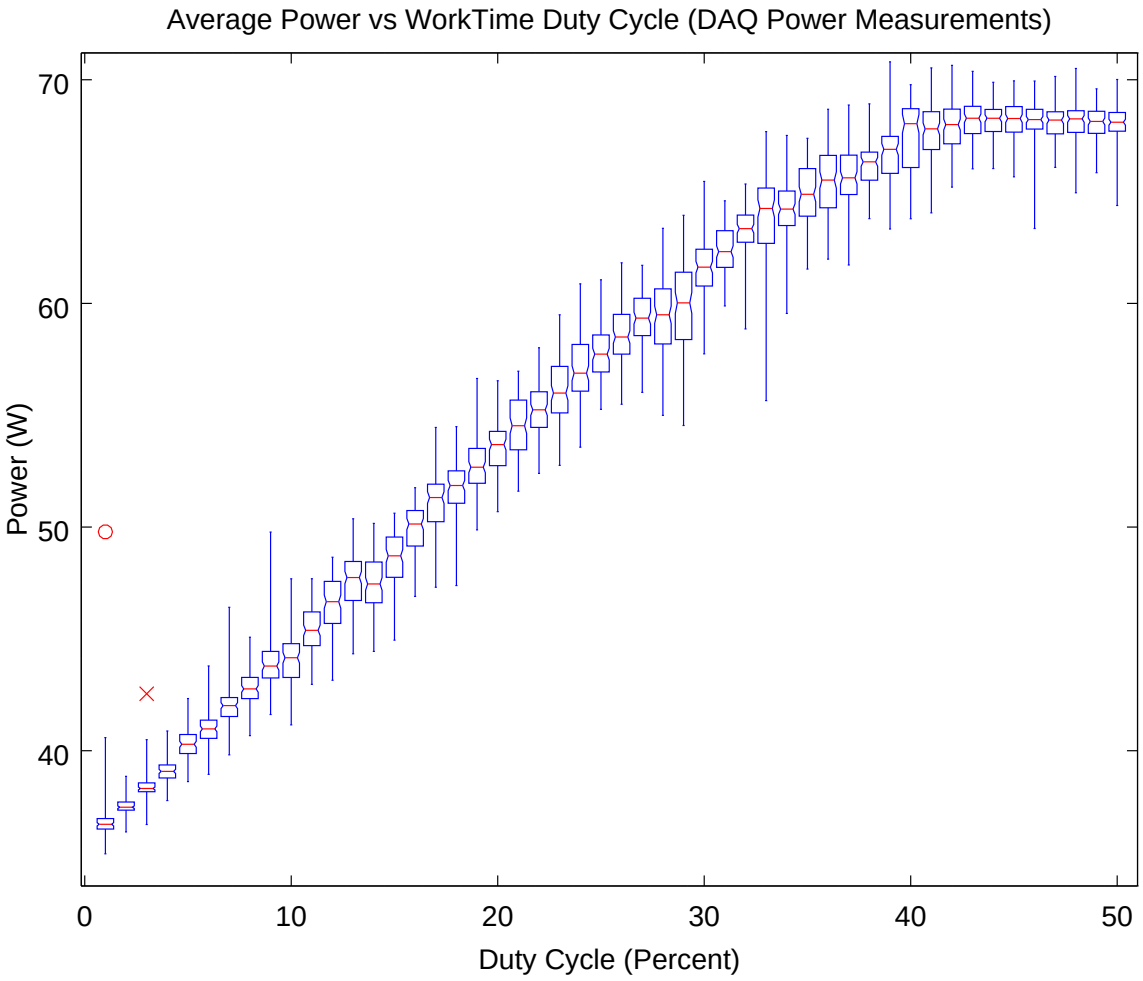


Figure 2.8: Inserting Delays with SIGSTOP and SIGCONT

power consumed by the server. Therefore, the minimum idle power can be considered as the zero-input response, y_0, of the incrementally linear system block shown in Figure 2.9, where the output of the incrementally linear system, y, is the superposition of the linear plant response, y_p with the zero-input response, y_0, such that $y = y_p + y_0$.

For each job, the actuator which receives the controller output signal, $u^j(t)$, is a program designed to inject idle cycles into the job process, its threads, and its child-processes by means of Linux SIGSTOP and SIGCONT signals discussed in Section

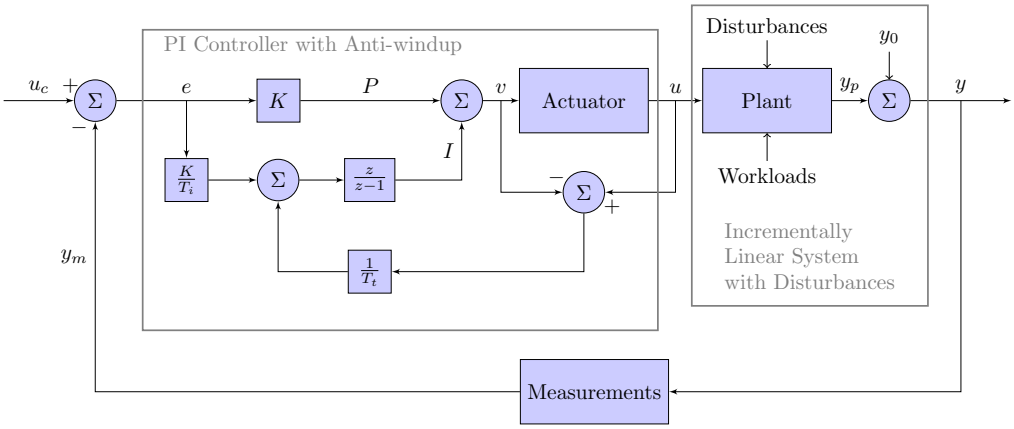


Figure 2.9: Block Diagram of Control System Model

2.3.6. The control signal itself, $u[k]$ is a value between (0.001, 0.999) which represents the *duty cycle* of the process, defined as $\frac{t_{work}[k]}{t_{work}[k] + t_{idle}[k]}$, where $t_{idle}[k]$ is the time between SIGSTOP and SIGCONT signals and $t_{work}[k]$ is the time between SIGCONT and SIGSTOP signals. In the specific implementation, $t_{work}[k] = t_w$ is a constant and $t_{idle}[k] = (\frac{1}{u[k]} - 1)t_w$. Note that a duty cycle which requires very long idle times would then render the job unresponsive on short time scales. Therefore the idle and work times should be constrained by:

$$t_w < \frac{t_{idle}^{max}}{\frac{1}{u_{min}} - 1} \tag{2.2}$$

where t_{idle}^{max} is the maximum desired idle time and $u_{min} = D_{min} > 0$ is the minimum allowed duty cycle.

The Ziegler-Nichols step-response method produces the PI controller parameters: the feedback gain K, the integration time T_i and the tracking-time T_t; as shown in Table 2.1. This method is appropriate when the sampling period, h, is within

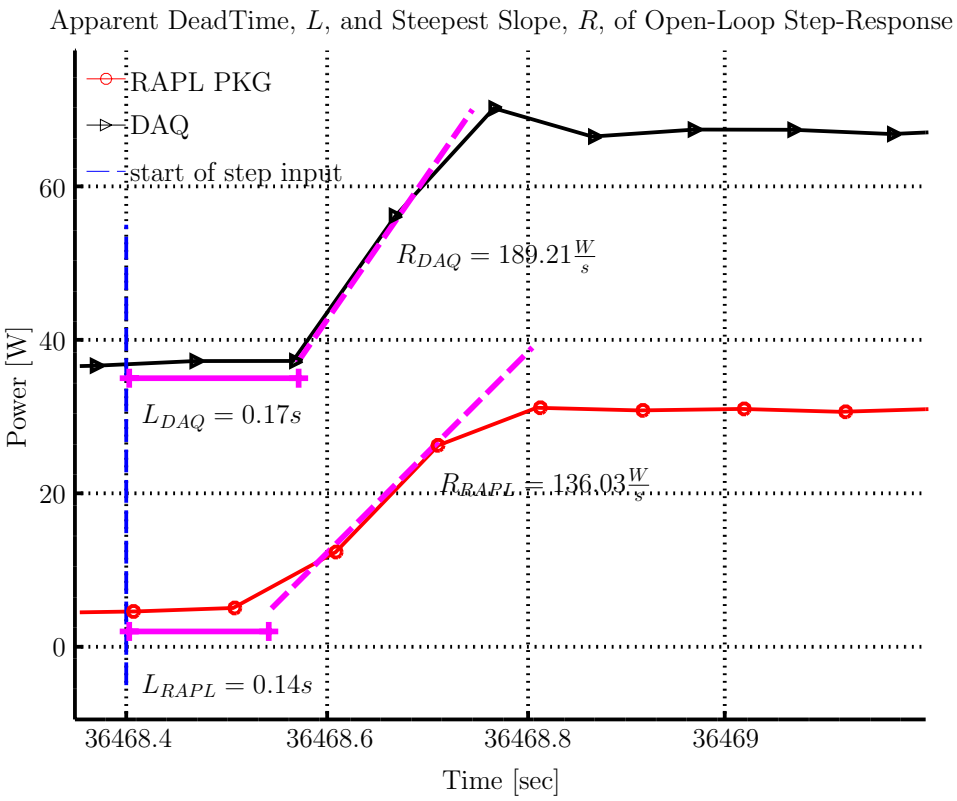


Figure 2.10: Calculation of R and L Using Ziegler-Nichols Step Response Method

valid range given by $\frac{h}{T_i} \approx 0.1$ to 0.3, following [36, §8.5]. Using a sampling period of $h = 0.1sec$ and measuring the open-loop step-response through the incrementally linear system in Figure 2.9 provides the apparent deadtime, L, and the step-response slope, R, as shown in Figure 2.10. The calculated values, R and L can then be plugged into the equations in Table 2.1 to obtain the experimental values for the gain, K, and the integration time, T_i. However, it is more useful to use the maximum possible values of $R_{max} = \frac{P_{max} - P_{min}}{h}$ and $L_{max} = 2h$. These values are given in the last rows of Table 2.1. Using these values to check that the sample period crite-

rion, $\frac{h}{T_i} \approx 0.1$ to 0.3 is met by $\frac{h}{T_i} = \frac{0.1}{0.6} = 0.167$ for the both the DAQ and RAPL power estimates, verifies that the Ziegler-Nichols step-response method has been used appropriately.

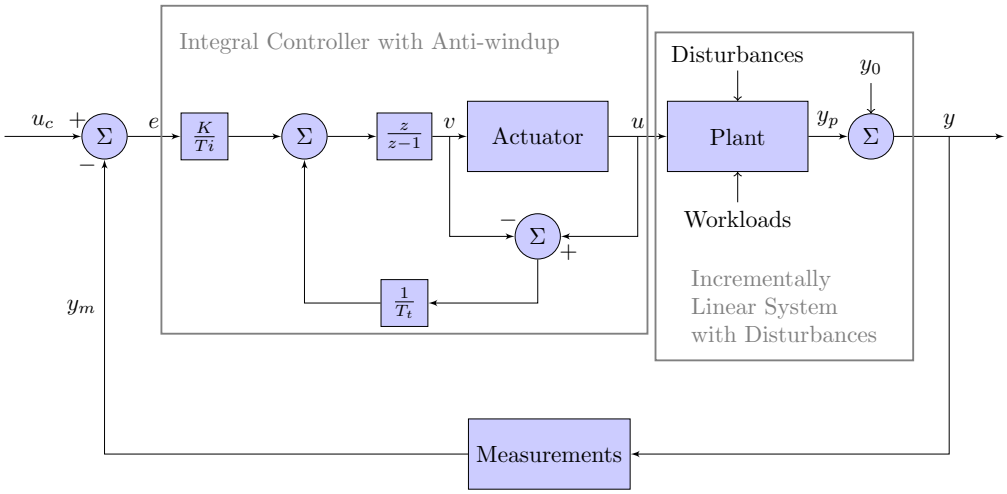


Figure 2.11: Block Diagram of Integral Controller

In practice, the error signal can be quite large from one time instance to the next due to the fact that disturbances in the plant occur at large magnitudes on short time scales. Here disturbances are due to independent processes running on the server over which our algorithm has no control. Therefore the proportional block, K, in the control loop in Figure 2.9 adds instabilities by responding to these disturbances. The integral term, on the other hand, responds to disturbances much more slowly and serves to drive the average error to zero, which is sufficient for our purposes. Therefore in practice, the proposed design only includes the integrator and the anti-

windup blocks in the controller, while the proportional block is omitted, as shown in Figure 2.11.

A second method for obtaining parameters for a PI controller, known as the first order plus dead time (FOPDT) model gives good results even without omitting the proportional term. The FOPDT model uses experimentally obtained parameters: process gain K_p (linear gain of the plant), process time constant T_p (time between initial response and 63% of peak response), and process dead time Θ_p (time between step input and initial response) [37]. Estimating the gain of the plant in the linear region shown in Figure 2.8 gives $K_p = \frac{\triangle y_m[k]}{\triangle u[k]} = 80$. The maximum dead time, $\Theta_p = 0.2$s. The time constant $T_p = 0.1$. Using these parameters to obtain the proportional gain, $K = \frac{1}{K_p}\frac{T_p}{\Theta_p + T_p}$, and integral time, $T_i = T_p$ gives a PI controller which performs just as well as the integral controller described above.

Implementing the integral controller in Figure 2.12 in a difference equation while ignoring the anti-windup and linear offset terms for simplicity and using the plant gain, A, and integral gain $\frac{K}{T_i}$, produces Equation 2.3:

$$\begin{aligned} y_m[k+1] &= \frac{AK}{T_i}\sum_{\tau=0}^{k-1} u_c[\tau] - y_m[\tau] + \frac{AK}{T_i}(u_c[k] - y_m[k]) \\ &= AK_i(e[k] + \sum_{\tau=0}^{k-1} e[\tau]) \\ &= y_m[k] + AK_i e[k] \end{aligned} \tag{2.3}$$

Figure 2.12 shows the integral controller track a setpoint over nearly the full power range of the server. As the setpoint exceeds the capacity of the controlled job to increase the power consumption, the power consumption saturates at about 67 watts

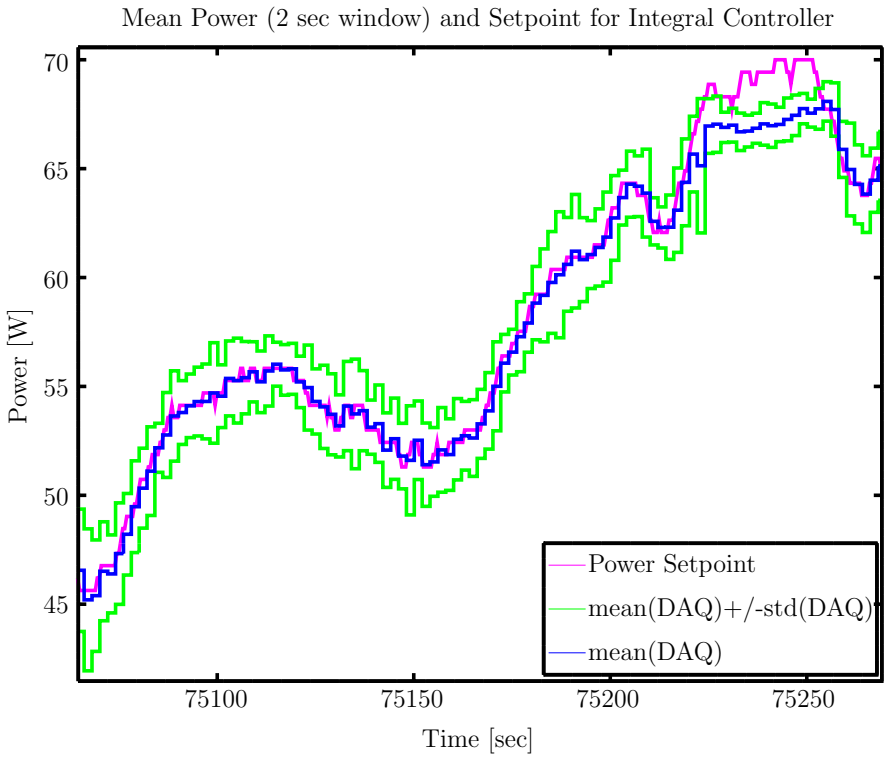
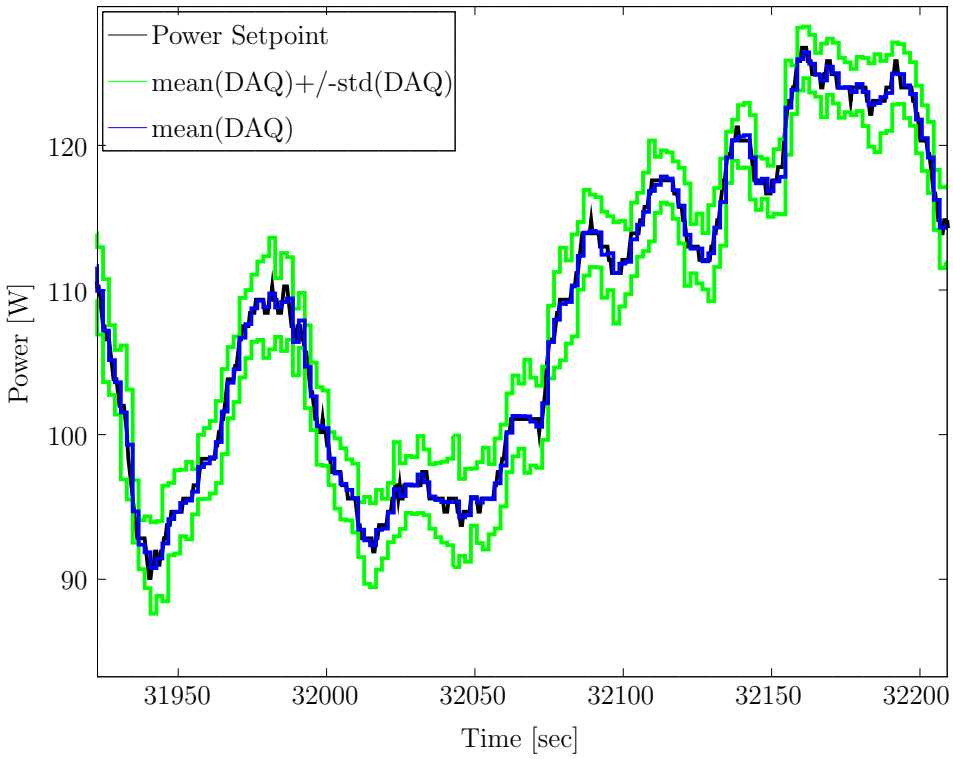


Figure 2.12: Closed-loop Performance of Integral Controller

which is simply the result of the controller reducing the length of the inserted sleep time to a point at which the job is using up its maximum possible CPU utilization. This does not cause an instability in the system and the system remains responsive immediately after the saturation event due to the anti-windup action of the controller.

2.4.2 Multiple Server Extension

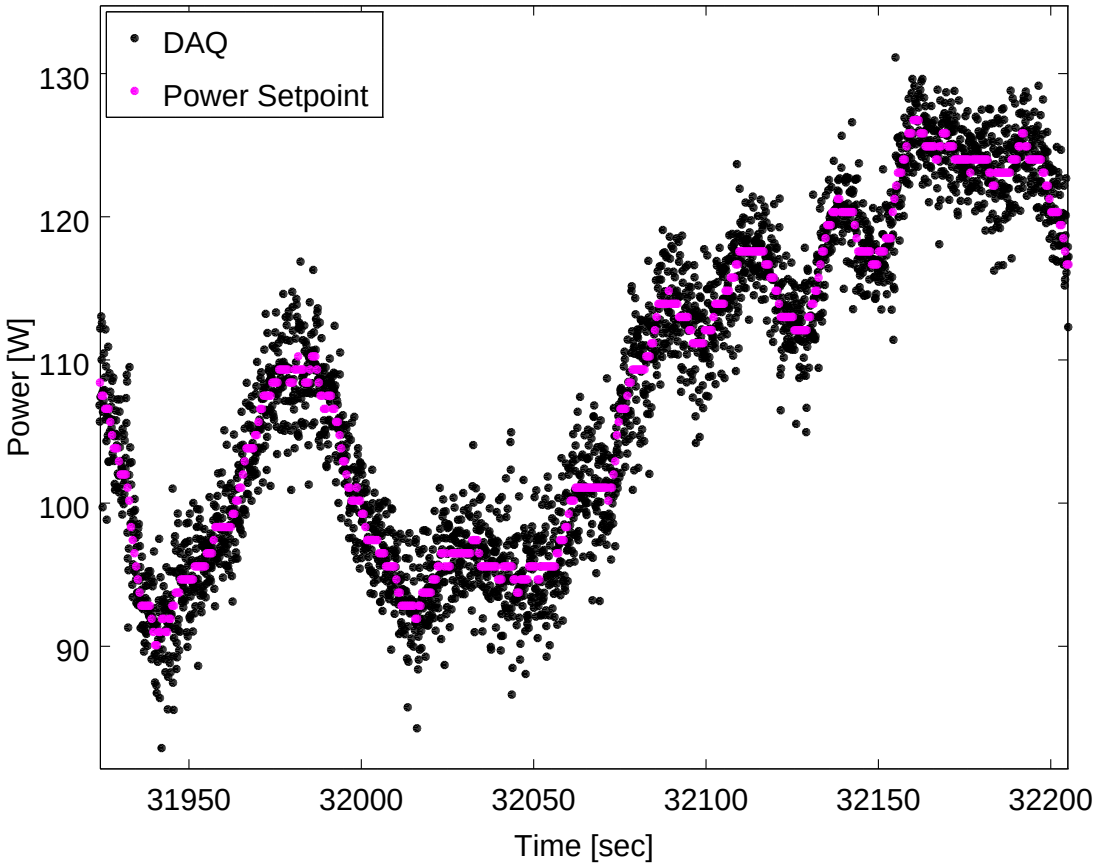
The control of multiple servers running multiple jobs is implemented exactly in the same way as the single server case except that the setpoint, u_c, is now the desired aggregate power consumption of the servers and the measured output, y_m, fed back to create the error signal is the measured aggregate power. Each job on each server can



(Mean and standard deviation of power is taken over 2 second windows)

Figure 2.13: Two Servers Tracking Aggregate Setpoint with Individual Controllers

use its own controller with these global input signals. Figure 2.13 shows the ability of two servers to cooperate to track the aggregate setpoint, each using its own integral controller, with global setpoint and feedback as just decribed. Figure 2.14 shows the the power measurements used in the feedback loop of the controller to demonstrate the raw feedback data stream from which the mean and standard deviation displayed in 2.13 is derived. The block diagram in Figure 2.15 shows the general architecture of the multiple server, multiple job controller. However, in the control system described



(Measurements are given as average power every 100ms)

Figure 2.14: Two Servers Tracking Global Setpoint with Individual Controllers

thus far and shown in Figures 2.13 and 2.14, the bias term β_i for each controller has been neglected, such that $\beta_i = 0$ for all jobs. The significance of the bias term is described in the next section.

2.5 Sharing Resources in Multi-Job/Multi-Server Distributed Control

Note that the integral controller used in Figure 2.13 and depicted in the block diagram in Figure 2.11 does not take into account workload throughput. Therefore

the control loop presented in Equation 2.3 and 2.11 equates to Equation 1.1 in the special case where $\beta_i = 0$. Therefore, in the case of two servers controlling separate jobs to track an aggregate setpoint, neither server gives preference to the throughput of its jobs and therefore it is likely that the two servers will not be consuming equal amounts of power and that the jobs of one server will be processed faster than those of the other. In fact, the same is true in the single server case in which different jobs are controlled by different controllers. Each controller will drive the injected sleep time of its jobs to minimize the error. How multiple controllers minimize the error has an infinite number of solutions and the question becomes: How can multiple controllers work together to share resources? E.g. if the setpoint is 120W, how can multiple controllers divide those 120W among themselves such that they each work fast enough to meet their required deadlines?

Requirements in answering this question are (1) that the solution must be decentralized and distributed (i.e. the servers must be able to make their own decisions rather than receiving control signals from a central authority), and (2) that the solution must be amenable and easily adaptable to multiple different kinds of jobs. Laxities in the possible solutions include: (a) deadlines should be met most of the time, but are not guaranteed, and (b) the goal is a workable, rather than optimal, solution.

The control algorithm was already given in the Introduction in Equation 1.1. Thus far, in the discussion of the integral controller in this chapter, we have treated only a special case of the algorithm of Equation 1.1 where $\beta_i = 0$. By allowing non-

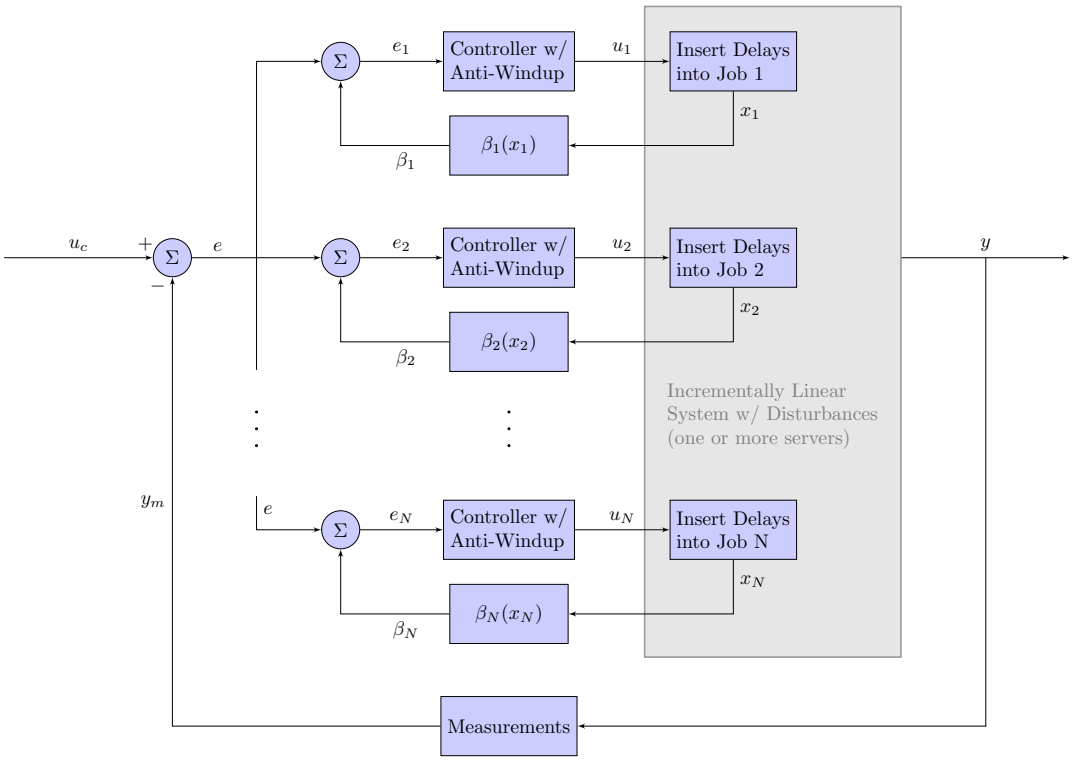


Figure 2.15: Distributed Controller Block Diagram

zero values of β_i, the proposed solution is to allow each job i to redefine its value of the aggregate setpoint in proportion to its own proximity to a deadline violation by adding a bias term, β_i, to the global setpoint u_c. This allows each job to change the error signal, $e_i[k] = \beta_i(x_i[k]) + u_c[k] - y_m[k]$ in its own controller by a small amount thus increasing its effective computational rate and its share of the power consumption. In turn, other controllers will perceive changes in their error signals, $e_j[k] = \beta_j(x_j[k]) + u_c[k] - y_m[k], j \neq i$ of the opposite magnitude and thus decrease their effective computational rate in order to minimize the error. The block diagram depicting this architecture is depicted in Figure 2.15. Assuming redefinitions of the

aggregate setpoint are very small, this scheme has little to no perceptible effect on the true global error, $e[k] = u_c[k] - y_m[k]$. Chapter 3 describes a demonstration of this effect using realistic workloads and a simple implementation of the bias term.

CHAPTER 3
DEMAND RESPONSE BY SERVERS - DEMO

3.1 Problem Formulation

This chapter presents a demonstration of a practical application of demand response by servers and clusters processing deferrable workloads. The demonstration is implemented on Dell PowerEdge R320 servers running video transcoding jobs in a contrived example of an online video streaming service. In the simplest description, the server(s) attempt to track a power setpoint while streaming data fast enough to avoid glitches and delays in viewer experience. In the single server case, one server juggles multiple jobs while tracking the setpoint. In the multiple server case, servers each juggle multiple jobs while tracking an aggregate setpoint. In the latter case, the servers know their aggregate error signal (actual power - setpoint).

Specifically, we assume that: N users have submitted video streaming requests; that these users have frame buffers of limited size, B; and that users are watching the video at a constant frame rate, R. Thus, the jobs servicing each request must transcode and transmit the video frames such that for each user, N, the number of frames sent, F, minus the number of frames watched, $t_{elapsed} * R$, does not exceed the buffer limit, B, or fall below 0, where $t_{elapsed}$ is the time-elapsed since the request. Therefore the job must transcode and transmit video frames to the user under the following constraint in Figure 3.1 to avoid delays in viewer experience.

$$0 < F - t_{elapsed} * R < B \tag{3.1}$$

We further assume that there is a queue of transcoding jobs without frame buffer or frame rate constraints. Such a queue of jobs could be videos to be downloaded to client's harddrives for watching later. A service could keep track of which episodes of TV series viewers watch and anticipate which episodes viewers are likely to view next. As part of the service agreement, the online service can send videos to users at any time, to be written on the users' hard-drives in anticipation that the user will wish to watch it. Therefore there is no buffer limit constraint in these anticipatory transcoding jobs and the server is free to transcode and transmit the video as quickly as it is able. These jobs therefore will simply use a zero bias term in their controller as discussed in Equation 1.1.

3.2 Hardware, Power Monitoring, & Workload Monitoring

The demo is implemented on Dell PowerEdge R320 server with an Intel Xeon E5-2400 series processor, running Ubuntu 12.04 with the 3.16.0-41-generic Linux kernel. Power consumption is monitored using a National Instruments 6323 X Series DAQ sampling AC voltage and current supplied to the server at 10kSa/s and 16-bit resolution. Voltage is sampled directly from a voltage divider circuit to differential inputs in the DAQ. Current is sampled from the output of bi-directional current sensor circuit using the Linear Technology LT1999 High Voltage, Bidirectional Current Sense Amplifier [38]. Grid frequency used as input for the algorithm is estimated using a custom device verified to estimate the grid frequency with ±10mHz accuracy [39]. Trancoder jobs were implemented using an adaptation of `avconv`, a video

transcoder program from *Libav* which is an open source set of libraries originating from the *FFmpeg* codebase [40]. The `avconv.c` file is changed to log the number of frames transcoded. The log file is read by controller in real-time.

3.3 Demand Response Algorithm

Continuing with the notation used in Section 2.5, each transcoding job on a server has its own control loop which takes as input the frequency of the grid, $f[k]$, the frames in the client's buffer $x_i[k]$, and the aggregate power of the servers, $y_m[k]$. Here $x_i[k] = F(kh) - kh * R$ with constraints in Equation 3.1, assuming that $kh = 0$ when the job begins. The job then uses this information to calculate a setpoint for the aggregate power, $u_c[k]$, as shown in Equation 3.2 and uses an integral controller to attempt the track the setpoint.

$$u_c[k] = [f[k] - 60Hz][\frac{P_{max} - P_{min}}{\widehat{f_{max}} - \widehat{f_{min}}}] + \frac{P_{max} + P_{min}}{2} \tag{3.2}$$

$$\beta_i(x_i[k]) = \begin{cases} P^+_{bias}(1 - \frac{x_i[k]}{0.3B}), & \text{if } x_i[k] < 0.3B \\ P^-_{bias}(1 - \frac{B - x_i[k]}{0.3B}), & \text{if } x_i[k] > 0.7B \\ 0, & \text{else} \end{cases} \tag{3.3}$$

Additional parameters in Equation 3.2 include the expected maximum and minimum frequency $\widehat{f_{min}}$ & $\widehat{f_{max}}$ estimated from historical data. For example, North American Electric Reliability Corporation (NERC) reports that frequency deviations greater than ±0.060Hz occur only 1-day in 10-years, while the frequency deviation is within ±0.04 for all but 0.5% of observations in the Eastern and Western interconnections [41]. Therefore reasonable values might be $\widehat{f_{max}} = \widehat{f_{max}} = 0.04$Hz. The maximum

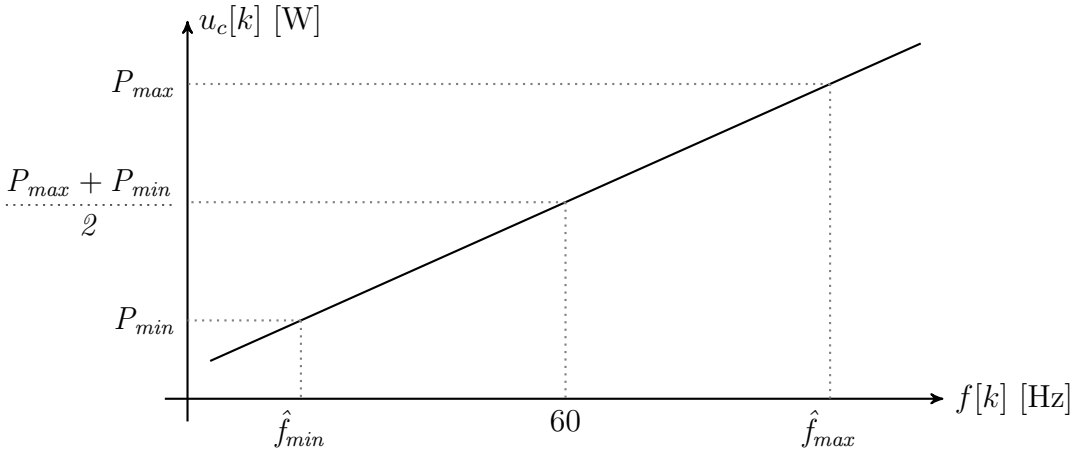


Figure 3.1: Power Target Scaled from Frequency

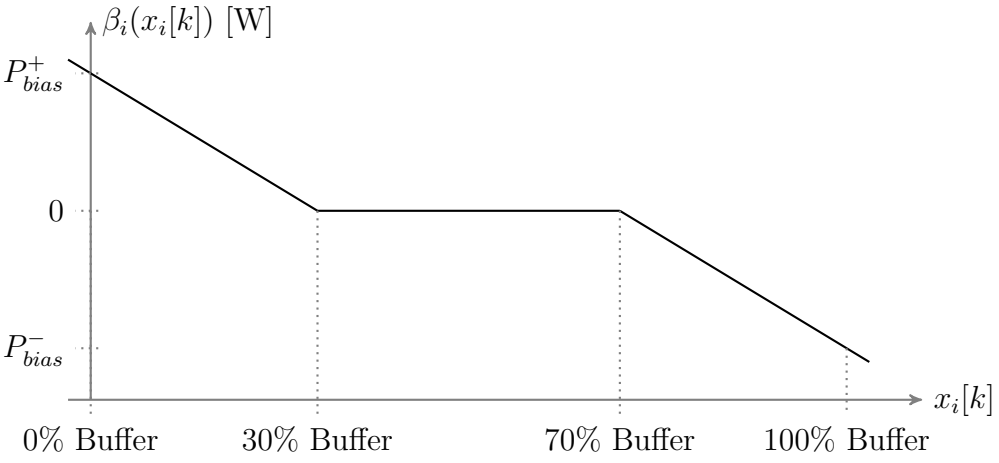


Figure 3.2: Controller Bias Function as in Eq. 3.3

and minimum operating power of the server(s), P_{max} & P_{min} is measured directly with the server(s) running at 100% and 0% CPU utilization, respectively. The meaning of these parameters is shown graphically in Figures 3.1 and 3.2. Simply put, the power target is a proportional scaling of the expected maximum and minimum frequency deviations to the maximum and minimum power consumption limits of the server, respectively. As our tiny data center cannot hope to balance the frequency deviation, we therefore simply approximate a scaling of the load imbalance as proportional to the frequency deviation. We then use the scaled load imbalance off-set by the median server power as the setpoint, $u_c[k]$. The bias term $\beta_i(x_i[k])$ is then added to the target power in order to maintain the buffer constraints in Equation 3.1. The exact function of $\beta_i(x_i[k])$ is described in more detail below.

Because all jobs receive the same global frequency input, $f[k]$, and calculate the same target power, $u_c[k]$, without some knowledge of the progress of the jobs, some jobs can become stuck in highly idle states. For example, if several jobs are running and some of them are running fast enough for the server power consumption to track the setpoint, the remaining jobs could be idling for too long and violating their frame buffer constraints. Adding the bias term, $\beta_i(x_i[k])$ allows each job to push its setpoint slightly above or below the target, thus increasing or decreasing its share of the CPU. If $\beta_i(x_i[k])$ is small (e.g. $P^+_{bias} \approx \left|P^-_{bias}\right| < 5\%\ P_{max}$ in Equation 3.3) for each job $i \in \{1, 2, ..., N\}$, with total number of jobs, N, then each job can vie for more or less CPU without substantial perturbations from $u_c[k]$. For each job, this means that if other jobs push the power consumption of the server above its own setpoint,

i.e. if $u_c[k] + \beta_i(x_i[k]) < y_m[k]$, the job's controller will increase the idle cycles of the job, thus decreasing its CPU utilization. In turn, this will slightly lower $y_m[k+1]$ and the controllers of any jobs whose setpoints were equal to $y_m[k]$ in the last instant, will now decrease the idle cycles of those jobs, thus increasing their CPU utilization while driving y_m back to u_c.

3.4 Results & Observations

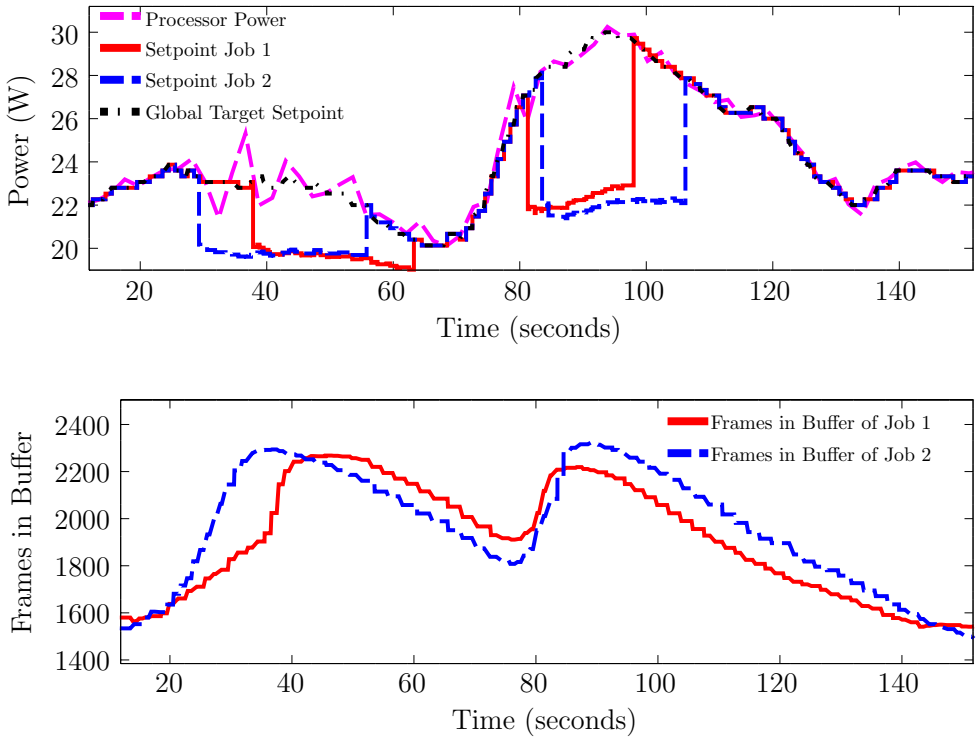


Figure 3.3: Exaggerated Bias Term in Controller for Two Transcoder Jobs

Figure 3.3 shows a test case in which multiple transcoder jobs are runnning on a single server, each with its own controller. The effective setpoints of two jobs, $u_c[k]+\beta(x_1[k])$ and $u_c[k]+\beta(x_2[k])$ and frame buffer states, $x_1[k]$ and $x_2[k]$, are shown for two of the jobs, along with the global target setpoint $u_c[k]$ and the aggregate power, $y_m[k]$. The bias terms of the two jobs are set with very high gain in Figure 3.3, to make the effect of the bias terms on the jobs' setpoints easily visible. The setpoints of the two jobs with deadlines clearly rise to and fall from from the aggregate setpoint whenever the jobs approach or leave their buffer limits (set at 3000 frames), respectively. The consequence of a job's setpoint falling below the global target is clearly seen in the decrease of frames in that job's buffer. Due to the overly large bias gain used for demonstrative effect, the aggregate processor power does oscillate between times (30, 50) seconds. However, this also shows the stability of the system even given grossly large perturbations from the bias term. Also note the very steep increase of frames in Buffer 1 around time 39 seconds, and in Buffer 2 around time 85 seconds. These steep increases are each preceeded by a drop in the setpoint of the other job. This demonstrates the event in which one job increases its injected idle times and the other job, if able, makes up for it by decreasing its idle times in order to keep the global error minimized.

A major advantage to the presented demand response system architecture is that it can be implemented and deployed easily on current servers and clusters and that it can be specifically deployed on particular jobs running on the same machine as other jobs which are not participating in demand response. In the data center man-

agement, reliability is paramount. There are many jobs and services which are too important to be possibly compromised by novel power management schemes which affect the performance of an entire server. Further, the job scheduling and prioritization required to balance the workload throughput of mission critical jobs along with the power management requirements of demand response become very complex using a centralized scheduler. The system presented here provides a distributed system in which each job tracks its own throughput constraints and indirectly, but effectively, adjusts its share of the CPU utilization in relation to other jobs; *indirectly*, because a job does not change its allocated resources, but only the duration of its idle time; *effectively*, because by changing its idle time, a job directly changes the power consumption of the server thus causing the other jobs to readjust their idle times in order to compensate for the change and track the setpoint.

CHAPTER 4
CONCLUSIONS

This research is intended as a proof of concept and starting point for research into distributed demand response by servers. While previous work has been done in the area of computers as demand response devices, as discussed in Section 1.3, they had either required specialized on-chip power regulation technology such as Intel's RAPL, used centralized job scheduling, or advanced prediction of workload and energy resources. The main contribution of this thesis has been to demonstrate a distributed algorithm for dynamically tracking a global power setpoint using multiple servers and deferrable jobs without stochastic data and using simple standard command signals available in every Linux distribution. While are numerous opportunities for future work in this area, three possibilities for refining the system presented here are generalizing the bias function for heterogeneous types of workloads, analysis of scalability, and adapting the system to provide performance guarantees. This paper concludes with a brief discussion of these three items.

- **Generalized Bias Function for Diverse Job Types**

  The proposed algorithm allows for multiple job types to participate in demand response, each with different definitions of its deadline constraints so long as all jobs use the same maximum and minimum bias terms in their algorithm. For example, a batch job of financial data computations might need to finish

N executions each hour. In such a case, a simple adaptation of Equation 3.3 could be given by Equation 4.1, where $N(t)$ is the number of executions in the last hour, and the criterion is $x_i[k] = N(kh) - kh * \frac{N}{3600} > 0$.

$$\beta(x_i[k]) = \begin{cases} [P^+_{bias}][1 - \frac{x_i[k]}{0.3N}], & \text{if } x_i[k] < 0.3N \\ 0, & \text{else} \end{cases} \tag{4.1}$$

Different jobs, one using Bias Equation 3.3 and another using Bias Equation 4.1, would be able to fairly jostle for their required share of CPU utilization to meet their throughput constraints given equivalent values of P^+_{bias} in each case. This and similar reformulations of the algorithm adapted to diverse deferrable jobs is a natural extension to this study.

- **Scalability**

  Prior to an efficiency analysis, the code implementing the demand response architecture could be streamlined for better efficiency and scalability. Currently, every block element of the control loop for each job is handled by a separate program and these programs are connected where appropriate via Linux pipes. A more scalable architecture with less overhead might be a client-server topology in which there is only one integral controller process which handles the state-feedback from each job and updates the work-time duty-cycle of each job. Scalability and real-world applicability could also be improved by creating a wrapper which integrates the distributed demand response algorithm with distributed computing utilities such as Hadoop. The present study did not examine the overall computation/power efficiency of the system. While the system con-

straints do provide the effect of a demand response system, it is possible that the current proposed architecture could result in overall loss in power efficiency. While controlling for power consumption levels, a comparison of the workload throughput under standard power management schemes versus throughput under the proposed control architecture would demonstrate the degree to which, the proposed architecture decreases a server's throughput capacity.

- **Performance Guarantees**

  While the presented system does perform well in tracking the global setpoint and operating within job constraints, it does not offer guarantees for either. Further analysis of the proposed algorithm and research into different possible bias functions may allow some performance guarantees which would be a substantial advance on the current proposal. For example an algorithm such as in [18] might provide a similar scheme for normalization between different job types, as discussed above, yet could also provide guarantees for convergence to the global setpoint and minimum throughput for the jobs.

REFERENCES

[1] F. D. Doty, "*Kicking Oil Addiction With Windfuels*," [Online] http://www.greentechmedia.com/articles/read/guest-post-kicking-oil-addiction-permanently-with-windfuels, Feb 2011, accessed 2015-10-16.

[2] B. Spencer, "*Wind farms paid £3million to stand idle so far this year because they were producing more power than the National Grid could handle*," [Online] http://www.dailymail.co.uk/news/article-2827555/Wind-farms-paid-43million-stand-idle-far-year-producing-power-National-Grid-handle.html, 2015-06-16, accessed 2015-10-10.

[3] M. Milligan, E. Ela, B.-M. Hodge, B. Kirby, D. Lew, C. Clark, J. DeCesaro, and K. Lynn, "Cost-Causation and Integration Cost Analysis for Variable Generation," pp. 1–37, June 2011.

[4] S.-J. Kim and G. B. Giannakis, "Scalable and robust demand response with mixed-integer constraints," *Smart Grid, IEEE Transactions on*, vol. 4, no. 4, pp. 2089–2099, Dec 2013.

[5] N. Li, L. Chen, and S. Low, "Optimal demand response based on utility maximization in power networks," in *IEEE Power and Energy Society General Meeting*, July 2011, pp. 1–8.

[6] P. Samadi, H. Mohsenian-Rad, V. Wong, and R. Schober, "Tackling the load uncertainty challenges for energy consumption scheduling in smart grid," *Smart Grid, IEEE Transactions on*, vol. 4, no. 2, pp. 1007–1016, June 2013.

[7] D. Hurley, P. Peterson, and M. Whited, "*Demand Response as a Power System Resource: Program Designs, Performance, and Lessons Learned in the United States*," [Online] http://www.synapse-energy.com/sites/default/files/SynapseReport.2013-03.RAP_.US-Demand-Response.12-080.pdf, p. 76, 2013.

[8] Midcontinent Independent System Operator. (2014) *Level 100 - Demand Response as a Resource*. [Online] https://www.misoenergy.org/Library/Repository/Meeting%20Material/Stakeholder/Training%20Materials/100%20Level%20Training/Level%20100%20-%20Demand%20Response%20as%20a%20Resource.pdf. Midcontinent Independent System Operator. Accessed 2015-10-29.

[9] J. Koomey, *Growth in data center electricity use 2005 to 2010*. Oakland, CA: Analytics Press, August 1, 2011, [Online]. Available: http://www.analyticspress.com/datacenters.html.

[10] Midcontinent Independent System Operator, "ACE chart," [Online] https://www.misoenergy.org/MARKETSOPERATIONS/REALTIMEMARKETDATA/Pages/ACEChart.aspx, June 2015, accessed 2015-10-29.

[11] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, p. 175, 2012.

[12] I. n. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 51–64, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2490301.2451123

[13] B. Aksanli and T. Rosing, "Providing regulation services and managing data center peak power budgets," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–4.

[14] S. Li, M. Brocanelli, W. Zhang, and X. Wang, "Integrated power management of data centers and electric vehicles for energy and regulation market participation," *Smart Grid, IEEE Transactions on*, vol. 5, no. 5, pp. 2283–2294, Sept 2014.

[15] M. Ghasemi-Gol, Y. Wang, and M. Pedram, "An optimization framework for data centers to minimize electric bill under day-ahead dynamic energy prices while providing regulation services," in *Green Computing Conference (IGCC), 2014 International*, Nov 2014, pp. 1–9.

[16] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, NY, USA: ACM, 2011, pp. 5:1–5:5. [Online]. Available: http://doi.acm.org/10.1145/2039252.2039257

[17] S. Goguri, J. Hall, R. Mudumbai, and S. Dasgupta, "A distributed, real-time and non-parametric approach to demand response in the smart grid," in *Information Sciences and Systems (CISS), 2015 49th Annual Conference on*, March 2015, pp. 1–5.

[18] R. Mudumbai, S. Dasgupta, and B. Cho, "Distributed control for optimal economic dispatch of a network of heterogeneous power generators," *Power Systems, IEEE Transactions on*, vol. 27, no. 4, pp. 1750–1760, Nov 2012.

[19] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. Jane Irwin, M. Kandemir, and V. Narayanan, "Leakage Current: Moore's Law Meets Static Power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[20] D. Brooks, "*Computer Science 246: Advanced Computer Architecture - Lecture 2*," Course Website [Online] http://www.eecs.harvard.edu/~dbrooks/cs246/cs246-lecture2.pdf, 2008, accessed 2015-09-23. [Online]. Available: http://www.eecs.harvard.edu/~dbrooks/cs246/cs246-lecture2.pdf

[21] T. K. (Intel), "*Power Management States: P-States, C-States, and Package C-States*," [Online] https://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states#_ednref1, April 2014, accessed 2015-10-29.

[22] S. Naffziger, "*AMDs Commitment to Accelerating Energy Efficiency*," [Online] http://www.amd.com/Documents/energy-efficiency-whitepaper.pdf, 2014, accessed 2015-10-29.

[23] "*Intel 64 and IA-32 Architectures Software Developers Manual, Volume 3B: System Programming Guide, Part 2*," [Online] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf, 2015, accessed 2015-06-28.

[24] "*Linux Kernel Documentation - Intel P-state driver*," [Online] https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt, accessed 2015-06-27.

[25] S. Li, T. Abdelzaher, and M. Yuan, "Tapa: Temperature aware power allocation in data center with map-reduce," in *Green Computing Conference and Workshops (IGCC), 2011 International*, July 2011, pp. 1–8.

[26] M. Karpowicz, "On the design of energy-efficient service rate control mechanisms: Cpu frequency control for linux," in *Digital Communications - Green ICT (TIWDC), 2013 24th Tyrrhenian International Workshop on*, Sept 2013, pp. 1–6.

[27] J. Corbet, "*Idle Cycle Injection*," [Online] https://lwn.net/Articles/383368/, April 2010, accessed 2015-10-30.

[28] Trefis Analysts, "*Intel (INTC) Detailed Analysis*," [Online] http://www.trefis.com/stock/intc/model/trefis?easyAccessToken=PROVIDER_87633ea72bd9dca1d79b8bc41462481e651ee6b5, accessed 2015-07-01.

[29] Amos Waterland, "*Stress POSIX Workload Generator*," [Online] http://people.seas.harvard.edu/~apw/stress/, 2013-10-24.

[30] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, Aug 2010, pp. 189–194.

[31] H. Chen, A. Coskun, and M. Caramanis, "Real-time power control of data centers for providing regulation service," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, Dec 2013, pp. 4314–4321.

[32] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177. [Online]. Available: http://doi.acm.org/10.1145/945445.945462

[33] Aberdeen Group and D. Csaplar, "*Is the Hypervisor Market Expanding or Contracting*," [Online] http://blogs.aberdeen.com/it-infrastructure/is-the-hypervisor-market-expanding-or-contracting/, 2012-09-25, accessed 2015-07-03.

[34] Arjan van de Ven and Jacob Pan, "*Intel PowerClamp Driver*," [Online] https://www.kernel.org/doc/Documentation/thermal/intel_powerclamp.txt, accessed 2015-07-03.

[35] Angelo Marletta, "*Cpulimit*," [Online] https://github.com/opsengine/cpulimit, 2015-06-16, accessed 2015-07-03.

[36] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design, 3rd Ed.* Mineola, New York: Dover Publications, 2011.

[37] controlguru. (2015, March) *Process Data, Dynamic Modeling and a Recipe for Profitable Control*. [Online] http://controlguru.com/process-data-dynamic-modeling-and-a-recipe-for-profitable-control/. Control Station Inc. Accessed 2015-11-04.

[38] "*LT1999-10/20/50 High Voltage, Bidirectional Current Sense Amplifier*," [Online] http://cds.linear.com/docs/en/datasheet/1999fd.pdf, 2015, accessed 2015-10-30.

[39] C. A. Bryceson, C. and J. Hall, "*Frequency Measurement Device*," Unpublished. Shared online https://www.dropbox.com/sh/0gk9ujkkcqzmo0z/AAC_2OonBGIMoUTL34rM_yvOa?dl=0, December 2014, submitted by authors in partial completion of senior design course at the University of Iowa.

[40] "*avconv Documentation*," [Online] https://libav.org/avconv.html, 2015.

[41] T. I. S. NERC, "*Interconnection Criteria for Frequency Response Requirements*," [Online] http://www.nerc.com/docs/pc/tis/Agenda_Item_5.d_Draft_TIS_IFRO_Criteria%20Rev_Final.pdf, August 2011, accessed 2015-10-30.