

2023

Performance-aware site-wide data center power management

<https://hdl.handle.net/2144/46654>

"Downloaded from OpenBU. Boston University's institutional repository."

BOSTON UNIVERSITY
COLLEGE OF ENGINEERING

Dissertation

**PERFORMANCE-AWARE SITE-WIDE
DATA CENTER POWER MANAGEMENT**

by

DANIEL C. WILSON

B.S., North Carolina State University, 2013

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2023

© 2023 by
DANIEL C. WILSON
All rights reserved

Approved by

First Reader

Ayse K. Coskun, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Second Reader

Martin C. Herbordt, PhD
Professor of Electrical and Computer Engineering

Third Reader

Orran Krieger, PhD
Professor of Electrical and Computer Engineering

Fourth Reader

Siddhartha Jana, PhD
HPC Research Scientist
Intel Corporation
Energy Efficient HPC Working Group

Not all those who wander are lost

—J.R.R. Tolkien

Acknowledgments

This thesis is possible because of support and guidance from many individuals.

I want to thank my advisor, Prof. Ayse K. Coskun, for her guidance through the overall journey as well as the individual challenges that came up along the way. I appreciate her focus on enabling me to keep an eye on how I can find meaningful takeaways from our work. With her help, I've learned the joy of developing a conceptual question into a published paper that I can present to a group of other people who love working on the same things.

I also want to thank my committee members, Prof. Martin Herbordt, Prof. Orran Krieger, and Dr. Siddhartha Jana. Each member has helped me develop personally through my university program or Intel internship, and has provided valuable feedback while working on my thesis.

I would like to thank Dr. Jonathan Eastep and Federico Ardanaz for the opportunities and guidance they provided in my time working as an intern at Intel. I learned a lot about software power management under their leadership, and received a lot of unique opportunities while working as part of their teams.

I also want to thank my other collaborators, Prof. Ioannis Ch. Paschalidis, Fatih Acun, Dr. Yijia Zhang, Christopher M. Cantalupo, Diana R. Guttman, Brad Geltz, Lowren H. Lawson, Asma H. Al-rawi, Dr. Ali Mohammad, Dr. Fuat Keceli, Brandon Baker, Dr. Aniruddha Marathe, and Dr. Stephanie Brink. I am also thankful for the advice and encouragement I've received from all of my lab mates in the PeacLab and my co-workers at Intel.

The support from my family has helped me throughout this work. I am thankful for my parents, my sisters, and their families for sharing their words of encouragement while working through their own challenges. My wife Elena has given support and encouragement that have been instrumental to all of my successes, beginning well before I applied to a PhD program and continuing through today.

Chapter 3 includes reprinted content from the following papers:

- Daniel C. Wilson, Siddhartha Jana, Aniruddha Marathe, Stephanie Brink, Christopher M. Cantalupo, Diana R. Guttman, Brad Geltz, Lowren H. Lawson, Asma H. Al-rawi, Ali Mohammad, Fuat Keceli, Federico Ardanaz, Jonathan M. Eastep, and Ayse K. Coskun. Introducing Application Awareness Into a Unified Power Management Stack. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021, pp. 320–329. DOI: [10.1109/IPDPS49936.2021.00040](https://doi.org/10.1109/IPDPS49936.2021.00040)
- Daniel C. Wilson, Ioannis Ch. Paschalidis, and Ayse K. Coskun. Site-Wide HPC Data Center Demand Response. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 2022, pp. 1–7. DOI: [10.1109/HPEC55821.2022.9926322](https://doi.org/10.1109/HPEC55821.2022.9926322)

Chapter 4 includes reprinted content from the following paper:

- Daniel C. Wilson, Asma H. Al-rawi, Lowren H. Lawson, Siddhartha Jana, Federico Ardanaz, Jonathan M. Eastep, and Ayse K. Coskun. Guiding Hardware-Driven Turbo with Application Performance Awareness. In *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*. 2022, pp. 1–8. DOI: [10.1109/IGSC55832.2022.9969356](https://doi.org/10.1109/IGSC55832.2022.9969356)

Some of the work in this thesis was partially funded by the Hariri Institute and the Institute for Global Sustainability at Boston University. Development of the GEOPM software package has been partially funded through contract B609815 with Argonne National Laboratory. Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-815595).

PERFORMANCE-AWARE SITE-WIDE DATA CENTER POWER MANAGEMENT

DANIEL C. WILSON

Boston University, College of Engineering, 2023

Major Professor: Ayse K. Coskun, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

ABSTRACT

Top high performance computing (HPC) data centers recently entered the era of exascale computing, requiring up to tens of megawatts for a computing facility to meet its users' computing needs. The massive capacity for power at a single site comes with challenges in power management. Poorly managed power may result in unnecessarily high demand on costly energy or may cause the system to under-perform. An HPC data center may have many types of users and workloads with non-trivial power requirements, making it difficult to select a one-size-fits-all policy. But the high power capacity also offers opportunities for data centers to be key players enabling greater adoption of renewable energy across a power grid. Data centers can adjust their demand through software power management policies to help smart grids balance against nature's time-varying green energy supply.

This thesis claims that multi-tiered power management methods are essential for data centers to implement site-wide power management policies that accurately respond to changing power constraints at a higher cluster-level tier while reacting to application-specific performance impacts at a lower job-level tier. Through investigations over site, cluster, job, and server characteristics, we demonstrate that a feedback-driven multi-tiered

power management approach meets power management objectives more effectively than siloed solutions. We design a cluster power management policy that distributes power across jobs using knowledge about job power-performance properties, demonstrating up to 7% reduction in system time dedicated to jobs and up to 11% savings in energy, compared to a policy without job awareness. We provide a power management framework that enables accurate, dynamic cluster power control while reacting to incomplete or inaccurate prior knowledge about job power and performance properties. We add a site-wide power model to a cluster power management policy that offers regulation services in a smart grid, showing 1.3x cost savings compared to a policy that is unaware of site-wide power consumption. We introduce a job power management policy that integrates job performance awareness with knowledge of hardware power-performance trade-offs, demonstrating up to 40% energy reduction and 17% execution time reduction in an imbalanced, compute-bound benchmark compared to a policy without frequency throttling.

Contents

1	Introduction	1
1.1	Thesis Statement	4
1.2	Thesis Contributions	5
1.2.1	Multi-Level Power Management	5
1.2.2	Performance-Aware Job-Level Power Management	7
2	Power Management in Data Centers	9
2.1	Power Capping	10
2.1.1	Motivations for Power Capping	10
2.1.2	Power Capping Controls in Data Centers	11
2.2	Power Prediction	12
2.3	Energy Cost Reduction	13
2.3.1	Energy-Efficient Power Management	13
2.3.2	Grid-Aware Power Management	15
3	Multi-Level Power Management	18
3.1	Related Work	19
3.1.1	Application-Level Power Control	19
3.1.2	System-Level Power Control	20
3.1.3	Multi-Level Power Control	20
3.1.4	Demand-Response-Driven Power Control	21
3.2	Job Performance Awareness in Cluster Power Management	22
3.2.1	Motivation for a Unified Solution	24

3.2.2	Adaptive Power Capping Policies	25
3.2.3	A Micro-Benchmark For Varied Imbalance and Power Properties	29
3.2.4	Experimental Environment	32
3.2.5	Results & Takeaways in Unified Cluster Power Management	38
3.3	Integrating a Performance-Aware Job Runtime in Cluster Power Management	44
3.3.1	A Framework for Multi-Tiered Power Management: ANOR	46
3.3.2	Implementing ANOR for Demand Response	48
3.3.3	Experimental Methodology	53
3.3.4	Results	58
3.3.5	Discussion	67
3.4	Cluster-Level Policies For Site-Level Power Objectives	68
3.4.1	Site-Wide Demand Response	70
3.4.2	Methodology	73
3.4.3	Evaluation	76
3.5	Concluding Remarks in Multi-Level Power Management	80
4	Performance-Aware Job-Level Power Management	82
4.1	Imbalance in Bulk-Synchronous Parallel Processing	82
4.2	Related Work in Performance-Aware Power Management	83
4.2.1	Hardware-Level Efficiency Tuning	83
4.2.2	Application-Level Efficiency Tuning	85
4.3	Guiding a CPU Power Manager to Rebalance Parallel Applications	86
4.3.1	Configuring CPU Core Priority in Response to Imbalance	88
4.3.2	Environment and Tools For Imbalance Experiments	95
4.3.3	Measured Impacts of Performance-Guided Frequency Boosting	101
4.4	Concluding Remarks in Performance-Aware Power Management	107

5	Conclusions and Future Work	108
5.1	Multi-Level Power Management	109
5.2	Adapting Job Schedules for Changing Performance Properties	113
5.3	Performance-Aware Job-Level Power Management	114
A	Open-Source Software Contributions	117
	Bibliography	119
	Curriculum Vitae	130

List of Tables

3.1	Knowledge available to each power management policy	28
3.2	Quartz System Properties	34
3.3	Workloads in Each Workload Mix	36
3.4	Power Budgets for Each Workload Mix	38
3.5	Membership of applications within workload mixes.	74
4.1	System Properties	96
4.2	System Package SST-TF Frequency Limits	96

List of Figures

2.1	Overview of regulation service components.	16
3.1	Power usage of Quartz system at LLNL over a period of one year.	23
3.2	Design characteristics of the synthetic micro-benchmark used for emulating typical HPC application properties.	30
3.3	The roofline plot of the synthetic kernel when executed on the the Quartz platform.	31
3.4	Total CPU power per node for different workload configurations, when running the benchmark variant that uses 256-bit (ymm) vector registers with no power limit under the GEOPM monitor agent.	32
3.5	Total CPU power per node for different workload configurations, when running the benchmark variant that uses 256-bit (ymm) vector registers under the GEOPM power_balancer agent.	33
3.6	Achieved frequencies of 2000 nodes in the Quartz cluster, under 70 watt CPU power limits.	34
3.7	Mean power used by each policy, across workload mixes.	39
3.8	The impact of three system-wide power management policies on different workload mixes and power budgets.	41
3.9	Tiers of power management entities in the ANOR framework.	47
3.10	Our implementation of ANOR for demand response.	48
3.11	Comparison of balancing mechanisms used by two power-capping methods.	50

3.12	Execution time of each job type under varied power caps, relative to the execution time at a 280 W CPU power cap per node. Error bars show standard deviation over 10 runs.	54
3.13	High-level organization of our data center demand response simulator. . . .	57
3.14	Estimated job slowdown when 8 job types are each executing one instance concurrently under a range of shared power budgets.	59
3.15	Performance impact when a medium-sensitivity job is misclassified as one with higher or lower sensitivity than its true behavior, while co-scheduled with both high-sensitivity and low-sensitivity jobs.	61
3.16	Job slowdown when BT and SP are scheduled under a shared power budget with 75% of TDP.	62
3.17	Job slowdown when two instances of BT scheduled under a shared power budget with 75% of TDP, with one instance misclassified as IS.	63
3.18	Job slowdown when two instances of SP are scheduled under a shared power budget with 75% of TDP, with one instance misclassified as EP.	63
3.19	Time-varying cluster power targets and measurements using ANOR over an hour of job arrivals from 6 job types.	64
3.20	Mean execution-time slowdown of job types under a 1-hour schedule with time-varying cluster power caps.	65
3.21	QoS degradation under different levels of performance variation.	66
3.22	The simulator places control logic in an orchestrator that schedules work and applies power limits to servers to satisfy user and independent service operator objectives.	70
3.23	PUE of the MGHPCC data center as a function of outdoor wet bulb temperature.	72

3.24	Cost of demand response (DR) policies with and without site-wide awareness, compared to the cost of electricity purchase policies that do not participate in demand response programs, averaged across 3 simulations of the workload mixes described in Section 3.4.2. Shaded regions indicate the 95% confidence interval.	77
3.25	Sensitivity of worst-case QoS, regulation signal tracking error, and cost savings as a response to corner cases of PUE modeling error.	78
3.26	Cumulative distribution function of target-power-tracking error as a percentage of the <i>R</i> bid, for the W13 workload mix at 50°F wet-bulb temperature.	79
3.27	Cumulative distribution functions of QoS degradation as a percentage of each application’s QoS threshold, for the W13 workload mix at 50°F wet-bulb temperature.	80
3.28	Job weights selected by training the AQA demand response policy against each workload mix.	80
4.1	SST-TF CPU core frequency trade-offs.	87
4.2	Leading and lagging processes in an iteration of a bulk-synchronous loop.	89
4.3	Data flow for performance-aware SST-TF configuration.	90
4.4	Target time selection searches for the SST-TF configuration that is estimated to minimize the time spent in an application’s critical path.	91
4.5	Performance of selected benchmarks under various frequency limits.	99
4.6	Imbalance of the selected benchmarks.	100
4.7	Distributions of average frequency across CPU cores and across time in monitor-only runs and under frequency-control variants of the performance rebalancer.	101
4.8	Time savings for imbalanced applications under different frequency-control variants of the performance rebalancer.	102

4.9	Distributions of average achieved CPU core frequency for cores that spend the most non-networking time in each iteration of the application.	104
4.10	Energy savings for imbalanced applications under different frequency-control variants of the performance rebalancer.	106

List of Abbreviations

ACPI	Advanced Configuration and Power Interface
ANOR	Attach Nested-Objective Runtimes
CPU	Central Processing Unit
DR	Demand Response
DVFS	Dynamic Voltage and Frequency Scaling
GEOPM	Global Extensible Open Power Manager
GPU	Graphics Processing Unit
HPC	High-Performance Computing
IPMI	Intelligent Platform Management Interface
ISO	Independent Service Operator
JSRM	Job Scheduler and Resource Manager
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LLNL	Lawrence Livermore National Laboratory
LRZ	Leibniz Supercomputing Center
MPI	Message Passing Interface
MSR	Model-Specific Register
PUE	Power Usage Effectiveness
QoS	Quality of Service
RAPL	Running Average Power Limit
RIKEN	Institute of Physical and Chemical Research
RSR	Regulation Service Reserves
SST-CP	Speed Select Technology Core Priority
SST-TF	Speed Select Technology Turbo Frequency
TDP	Thermal Design Power

Chapter 1

Introduction

Data centers currently consume about 3% of the global energy supply each year, about double their consumption from 10 years earlier [46]. We have entered the exascale era of computing, where supercomputing systems can demand tens of megawatts at their peak capacity [85], with planned facilities being provisioned for up to 85 MW of capacity [6]. As demand for faster, bigger, and more fine-grained computations increases, so will the demand for high-power data centers.

The high demand for computing power is expensive in terms of the direct cost of energy and in system infrastructure costs. Electricity service contracts with supercomputing centers increasingly include variable cost mechanisms that motivate a need for grid-aware data centers [20]. Computing facilities are exposed to unpredictable energy pricing, which can ultimately limit their ability to run at full capacity. In 2022, supercomputer Fugaku was taken 30% offline for 3 months to limit energy expenses during a surge in electricity prices [70].

High power demand also requires additional spending on infrastructure to support the facility's peak power consumption. Supercomputing centers may choose to over-provision computing equipment, meaning that they provision computing equipment with peak power demand that exceeds the site's power capacity. Over-provisioning in this fashion optimizes equipment spending for cases where typical power demand is much lower than peak power demand. High-performance computing centers may use 80% or less of their power capacity during normal operation even when compute nodes are near fully utilized [10, 66, 81]. A facility can choose to limit total power consumption to a lower level in order to

support over-provisioning. Computing performance is impeded if power is not effectively distributed within the data center, so over-provisioned data centers may focus on using power management policies that optimize system performance under a power cap [68, 80].

Data centers are capable of managing power consumption in multiple levels of scope. They can manage power across the data center's network through power capping mechanisms exposed through out-of-band management interfaces such as Redfish and Intelligent Platform Management Interface (IPMI). Batch job schedulers and resource managers (JSRMs), common in high-performance computing (HPC) systems, additionally simplify matters by applying power management policies across jobs in a cluster [82, 2]. JSRMs typically perform their responsibilities agnostic to the power and performance properties of the jobs they manage, relying only on resource constraints (e.g., count of processing units needed, expected execution time, hand-selected power states, etc.) provided by users. Many power-aware job runtimes provide power management solutions that exploit power and performance properties of specific job types [76, 13, 29, 22], providing the means to automatically improve power, performance, and energy trade-offs *within* a compatible job.

The degree of power control in data centers offers opportunities for data centers to contribute toward greater electricity grid flexibility, which is a key component of achieving net-zero emissions by 2050. The International Energy Agency reports that demand response is expected to meet about half of global demand flexibility needs by 2050 [39], and that many nations are introducing new grid flexibility initiatives. Data centers can reduce their net energy costs by offering demand response programs [14], and can do so while offering performance guarantees [97]. In 2022, Lawrence Livermore National Laboratory (LLNL) was asked by their electricity provider to prepare for requests of up to 8 MW in power cuts to help balance against spiking demand from a heat wave [49]. As different markets continue to develop new grid flexibility initiatives, data centers will be increasingly exposed to opportunities in demand response participation.

While there are many methods and tools to manage power and performance in data centers, there is a need for end-to-end management techniques that allow for objectives and system behavior that can change over time. Several works investigate methods to respond to variable power objectives [73, 18, 64, 51, 97, 45] or to meet static power objectives under unknown or changing job properties [13, 22, 25, 29, 34, 48, 95]. However, these works do not consider cases where constraints and objectives at the data center level and the job level may change over time.

This thesis investigates scenarios where HPC data centers dynamically change their power consumption in order to offer demand response services in a smart grid while offering Quality of Service (QoS)¹ guarantees for user jobs running in the data center. We achieve improved energy efficiency and performance under static cluster-wide power caps by adding a job performance awareness to a cluster-level power manager. We present a multi-tier power management framework that accurately tracks time-varying cluster-wide power caps while adapting to unexpected job power-performance properties. We demonstrate that a cluster-level power manager can respond to site-wide time-varying power objectives through the use of a simple site power model. Lastly, we introduce a method to combine awareness of a compute node’s power-performance tradeoffs with awareness of an application’s performance properties in order to improve performance of imbalanced parallel applications, which do not require uniform access to CPU resources for efficient operation.

Our investigations are scoped to HPC data center power management and performance challenges on systems where users get exclusive access to hardware while their jobs execute. Some concepts are broadly applicable to data centers that can share power across servers and measure performance in terms of the time it takes for a user’s computational request to be fully serviced. For example, other types of large data centers often have many levels

¹In this work, we refer to QoS in terms of probabilistic deadlines for batch-computing jobs. That is, a job should finish executing by a given deadline for some percent of instances of that job type.

of power management concerns ranging from constraints at the whole facility down to individual computing components [7]. Although work may be scheduled at different units (e.g., requests, jobs, tasks, etc.), scheduling still exists as a key control knob. Power capping or throttling are generally also available, but their usage may be more complex in multi-tenant environments (e.g., cloud computing) since multiple users may be impacted by a single power capping decision. The level of prior knowledge about workloads varies widely in different computing facilities, where application-specific computing platforms (e.g., server farms for build automation or computer-aided design simulations) may have relatively high confidence about upcoming workload properties, but other systems (e.g., platform-as-a-service) may not have much prior knowledge of user workload properties. Across types of data centers, effective power management finds a balance between power, energy, and performance by allocating power where it is needed most.

1.1 Thesis Statement

Modern HPC data centers need to manage megawatts of computing power, potentially with time-varying power and performance objectives. This thesis claims that multi-tiered power management methods are essential for data centers to implement site-wide power management policies that accurately respond to changing power constraints at a higher cluster-level tier while reacting to application-specific performance impacts at a lower job-level tier. That is, power and performance optimizations are limited without visibility and control that is both wide at the cluster scope and deep at job scope. We evaluate opportunities and challenges in multi-tiered power management policy design, and investigate ways to use power management techniques in response to HPC performance challenges. Much of our investigation is framed in demand response scenarios as a way to motivate opportunities to manage time-varying power objectives. However, time-varying power objectives are broadly relevant across data center management scenarios. For example, time-varying site

power constraints may be introduced by cooling capacity and cost as a function of weather conditions and the thermal load of computing equipment, or may be driven by a facility's budgeting objectives under an electricity contract that has time-of-day tariffs, or could be motivated by temporarily degraded operation during periods of system maintenance.

1.2 Thesis Contributions

We evaluate the thesis through two fronts. First we investigate opportunities and software policies for multi-level power management, where we demonstrate methods to combine power management knowledge of power and performance across site, cluster, and job levels. Then, we investigate methods that react to specific performance challenges in high-performance computing, where we design a method to speed up jobs that place a non-uniform load on CPUs by combining knowledge of a job's work imbalance with awareness of hardware-level performance trade-offs. The high-level theme of our contributions is that we more strongly meet power management objectives (less energy, more accurate power capping, or faster job performance) by introducing visibility into a neighboring power-management layer. In particular, our work highlights the need for data center power management solutions that react to both time-varying system-wide power constraints and time-varying power-performance responses within the data center.

1.2.1 Multi-Level Power Management

The first part of our work (Chapter 3) progressively works toward higher levels of scope in data center power management, starting with pre-characterized cluster-level concerns, moving toward online objectives, and ending with site-level management. The end goal is to gain the performance-awareness benefits of job-level software runtimes while also inheriting the system efficiency benefits of more widely scoped system-wide power management mechanisms. Our aim is to achieve that end goal by loosely coupling the different power

management components so that each layer can exchange information uniquely visible to that layer without requiring a single policy that maintains full awareness across all management layers.

We begin with an opportunity analysis that demonstrates the need for such a multi-level power management framework (Section 3.2). We propose a workload-aware and system-power-aware policy that reduces time to solution in a power-limited cluster by distributing power both across and within workloads in a performance-aware manner. We identify a set of application design characteristics that emulate different power and performance trade-offs seen in HPC applications, and we design a micro-benchmark to trigger these different conditions. We evaluate the proposed policy against two other dynamic policies with either system power awareness or workload awareness, and an additional policy with neither feature. Our evaluations at scale show up to 7% system time reduction and up to 11% CPU energy savings, versus the policy that has neither feature.

Our initial opportunity analysis relies on an expectation that all job types submitted to a cluster have prior-known power-performance properties. Such properties may not be known for all job types, or may change over time. To address challenges where job properties are not fully known in a production setting, we design an end-to-end design framework to enforce cluster power policies while exploiting awareness of node-specific application properties (Section 3.3). The end-to-end framework relies on periodic upward communication of job performance properties from a job runtime to a cluster power manager, and periodic downward communication of job-level power objectives as orchestrated by the cluster power manager. We present a practical implementation of the design framework, integrating a cluster-level policy for demand response job scheduling and power management with a software framework for job-level monitoring and control. We evaluate the performance lost when job power and performance properties diverge from our prior expectations and mitigate those losses through feedback from a job-level performance monitor.

Cluster computing equipment accounts for only part of a data center’s power consumption profile. When power management objectives are driven by relationships with external entities, such as energy pricing in a smart grid, it is useful for the cluster to consider other parts of the data center’s power consumption when making power management decisions over computing equipment. We demonstrate that site-wide demand response participation in a data center enables lower operating costs than server-only demand response participation while using an existing QoS-aware demand response policy (Section 3.4). Our efforts to make demand response policies aware of site-wide power consumption achieve 1.3x cost savings with similar QoS degradation to demand response policies without site power awareness. We analyze the sensitivity of demand response cost savings to site-wide power model selection. In scenarios where there is low confidence in the site-wide power model, demand response policies can still reduce electricity costs, but may need to relax their QoS constraints. We show that batch job resource managers meet site-level demand response objectives and job-level QoS objectives by using a simple site power model on top of QoS-aware power managers that operate with job and server scope.

1.2.2 Performance-Aware Job-Level Power Management

The second part of our work (Chapter 4) focuses more closely on specific performance challenges that are faced in HPC data center power management. Specifically, we look at challenges in using power management to improve efficiency of imbalanced MPI applications, which is a common challenge in HPC systems [36, 27, 55, 29].

We describe a method to improve performance of imbalanced bulk-synchronous parallel applications through performance-guided frequency boosting (Section 4.3). We provide an evaluation of the energy and performance opportunities from rebalancing bulk-synchronous parallel MPI applications solely with P-State (voltage-frequency levels) control or with the new guided frequency boosting interface, and from using both together. Our method combines application-awareness with firmware-level power-management capabilities, bal-

ancing MPI applications by configuring the CPU power management interface. Our work includes a discussion of takeaways for using performance-guided turbo in software power management algorithms and application-level work rebalancers.

Chapter 2

Power Management in Data Centers

One of a data center's key goals is to ensure power is delivered everywhere it is needed to keep user workloads running. As a result, data centers need to provide cooling systems, networking, the computers themselves, and infrastructure to deliver sufficient power to everything, potentially with one or more layers of redundancy in any of those systems to ensure resilient operations. Typical large data centers spend up to an additional 20% more than their computing power demand for all of their supporting infrastructure [7].

Top HPC sites seek holistic solutions to manage their infrastructure, with several key use cases in power management [63]. Common power management objectives relate to power capping, power prediction, and energy cost reduction. Holistic solutions collectively consider factors at the facility or site level (such as power delivery and cooling systems), the cluster level (such as job schedulers and resource managers), job level (such as software runtimes), and server level (such as CPUs, GPUs, memory, and other computing resources).

The rest of this chapter provides some background in current approaches to managing data center power for objectives in power capping (Section 2.1), power prediction (Section 2.2), and energy cost reduction (Section 2.3). The work in this thesis utilizes power capping both as an end objective and as a mechanism used along with power prediction to implement policies for cost-reduction objectives.

2.1 Power Capping

Power capping refers to techniques that ensure power demand stays below a specified upper bound. Data centers have multiple motivations driving the need for power caps, and they have many tools at their disposal to enforce those caps.

2.1.1 Motivations for Power Capping

Data centers in the early exascale era consume up to tens of megawatts of power [85]. The large demand for power needs to be controlled in order for a facility to operate under the constraints of its own infrastructure and to ensure that the data center adheres to any requirements set by its electricity service provider.

Components within a computing system are rated with a vendor-defined Thermal Design Power (TDP), which indicates how much power that part can draw under its thermal design constraints. A conservative facility design could assume all components are always able to consume power up to their TDP. In such a design, the facility would provision enough cooling and power delivery infrastructure to support everything running at TDP concurrently. In practice, an HPC facility's typical power consumption is often much lower than its rated power capacity, even under high compute node utilization [66, 81, 10]. Alternatively, data centers may provision their system for *typical* power consumption, reducing the share of a design budget spent on unused power capacity. Under such an *over-provisioned* design, a data center with fully-utilized hardware may use power capping mechanisms to direct the available power to the facility's components in a performance-efficient manner [68].

Data centers may elect to utilize power caps as a mechanism to adhere to contractual obligations with their electricity service providers. For example, the Institute of Physical and Chemical Research (RIKEN) data center enforces site-level power caps to prevent scenarios where penalties must be paid to the facility's electricity service provider [63].

2.1.2 Power Capping Controls in Data Centers

Data centers need to be able to modulate power demand in order to implement their power management decisions. Data center power consumption can be broadly impacted by software control through job scheduling (i.e., changing when and where work goes to servers) and through hardware configuration settings that are exposed to software controls on servers. Many modern processors in HPC systems support both frequency control and power capping mechanisms for software-driven power management.

Server level power management challenges are similar to the previously discussed facility-level motivations for power capping an over-provisioned system. With the breakdown of Dennard scaling, we are hitting a *utilization wall* that prevents us from fully utilizing all parts of a chip at one time [87]. To combat that challenge, CPU designers have come up with several ways to achieve improved performance under power density constraints [83]. Frequency boosting is one such technique. When fewer cores are active, higher (*turbo*) frequencies can be achieved on the active cores. The highest turbo frequency configuration, the *single-core turbo* frequency, is achievable when only one core in the processor is active.

Most hardware vendors provide some degree of firmware support for power management features. Examples of software interfaces to such features include DVFS (Dynamic Voltage Frequency Scaling) and power capping through vendor-specific interfaces such as Intel's RAPL (Running Average Power Limit) [24], IBM's PSR (Power Shifting Ratio) [74], NVML (NVIDIA Management Library) [61], and AMD's APM (Application Power Management) [3].

Software frequency control can be exposed through Advanced Configuration and Power Interface (ACPI) Performance States (P-States). P-States enable hardware vendors to expose a fixed set of frequency-voltage configurations that may be selected by software. For example, on Intel processors, P-State settings range in 100 MHz steps from a lowest-frequency P_n setting to the processor's base frequency P_1 setting, with additional P-States

that opportunistically exceed the base frequency depending on the processor’s design constraints (e.g., thermal and current limits) [43].

While many power-related controls are exposed to software, the challenge is to choose a configuration that ensures efficient or performant application execution. Other power management strategies discussed in this chapter may utilize these power capping controls while including additional insights about how the system’s behavior reacts to power caps.

2.2 Power Prediction

Both daily operations and facility planning benefit from power prediction. Power prediction enables better-informed interactions between a data center and its electricity provider, and better power-aware scheduling decisions. Power prediction also helps facility designers determine how to provision their next-generation systems. Several works proposed using job power properties to infer power properties from information available in a job queue (e.g., user identifiers, submission time, processes requested, job name, etc.) using machine learning models for power consumption [9, 66, 78]. Job queue information is desirable as an input to power prediction models since it is available *before* execution begins.

Proactive power-aware scheduling techniques aim to minimize the use of mechanisms that increase job execution time. By predicting each job’s power properties, such techniques arrange the job start times in order to stay under a power bound [28, 69]. By treating energy as a scheduler constraint (much like execution time), these approaches can use existing *backfilling* scheduler heuristics, which improve system utilization by opportunistically filling in schedule gaps with jobs that are expected not to impact deadlines of current top-priority jobs.

Data centers may also need to predict their power consumption to aid interactions with their electricity service providers. For example, Lawrence Livermore National Laboratory (LLNL) informs their energy provider when they expect large swings in upcoming power

demand [63]. Fast and accurate predictions help ensure grid stability by preparing grid operators for fast-changing demand.

Power prediction also enables data centers to plan their upgrades based on expected long-term demand. Effective power prediction has helped facilities plan properly-sized upgrades to existing infrastructure, and to avoid provisioning too much power capacity for a new-generation system [63]. In both examples, power prediction helps data centers spend their provisioning budget to more efficiently meet their end users' computing needs.

The theme among power prediction use cases is to enable proactive power management decisions. Energy-constrained schedulers use predictions to keep a data center under an average power cap, electricity service providers can prepare for power swings based on a data center's predictions, and facility upgrade budgets can be spent more efficiently by using power predictions.

2.3 Energy Cost Reduction

Power management is an effective tool to reduce energy costs in a data center. Typical approaches to cost reduction include techniques to improve the data center's efficiency and applications in grid-aware power management.

2.3.1 Energy-Efficient Power Management

Energy efficiency broadly refers to reducing the amount of energy required to finish some amount of computational work. This is frequently managed at the facility level by optimizing how cooling systems are configured to interact with computing infrastructure [63]. A cluster can improve its energy efficiency by making power management decisions based on how software is using the cluster. Such decisions may be automated through software implementations in batch job schedulers and resource managers that control how computing resources are assigned to user job requests.

A general technique to increase energy efficiency is to reduce the overall power consumption when there is low risk to introduce performance degradation. At a coarse level, this can be achieved by turning off idle compute nodes whenever there are sufficiently long idle periods in a job schedule [54]. When jobs (or phases of jobs) can be characterized into classes (e.g., compute intensive, memory intensive, etc.) based on their power-performance properties, it is possible to pre-determine efficient power settings to apply whenever the job class is detected [32, 48].

The job scheduler may not have sufficient knowledge to predict every job's power and performance trade-offs. A data center may opt to use human-guided policy selection to enable energy-efficient power management without relying on predicting which type of job is running. For example, the SuperMuc facility at Leibniz Supercomputing Center (LRZ) is configured to run jobs with the CPU at lower than its base frequency, except for job types that are known to efficiently use higher-frequency configurations [5]. Supercomputer Fukagu at RIKEN allows users to select one of multiple server power management policies based on the user's knowledge of how the job will perform under each policy [47].

HPC applications often have varying power and performance trade-offs, as they may use different parallel software design patterns, switch between phases of computation, or follow different execution branches depending on their input data. Power management runtimes aim to resolve such challenges by modifying power controls in direct response to events that occur within a job. Several solutions improve efficiency of *imbalanced* applications, where some parallel processes have more work than others, by directing power toward resources that improve the job's critical path [13, 76, 55, 29]. Another common runtime approach is to search for efficient power configurations within detected regions of job behavior [22, 48, 1]. The Global Extensible Open Power Manager (GEOPM) HPC runtime [29] provides an abstract interface to runtime-based power management with performance awareness.

2.3.2 Grid-Aware Power Management

Electricity providers² have many market tools at their disposal to balance power supply against demand, such as time-of-day tariffs, electricity contracts with power bands, and different pricing for a customer's average versus peak power demand [20]. Data centers can react to variable pricing through techniques like peak shaving [35], which reduce power consumption at times of greater cost. Demand response programs allow market entities to offer power regulation services back to the grid as an additional mechanism that helps balance power supply and demand. In demand response programs, the regulation service provider increases or decreases its power demand at the request of the electricity provider.

Data centers are particularly well-suited to be regulation service providers due to their large capacity for power demand and due to their flexibility to modulate power consumption through software power management [15, 67]. A key challenge is to ensure that the data center still achieves its expected levels of performance. Although a data center can simply defer or slow down computing tasks to reduce power consumption, doing so negatively impacts the quality of service offered to end users of the data center. Many HPC systems operate at high resource utilization, and unchecked performance degradation is perceived as unplanned depreciation in the facility's value [8], so it is important to design demand response policies that limit the overall performance degradation.

The works included in this thesis investigate opportunities for data centers to participate in regulation service reserve programs while offering quality of service guarantees. In such programs (illustrated in Figure 2-1), the regulation service provider (for example, a data center) places a bid to offer a range of achievable power demand ahead of each hour. Within each committed hour, the data center receives a regulation signal every few seconds that indicates how much power the data center is expected to demand within the committed power range.

²This work broadly refers to an electricity provider as any grid entity that may offer demand response programs, such as a regional transmission organization or an independent service operator.

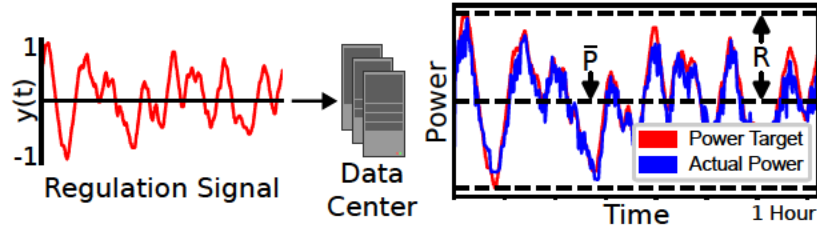


Figure 2-1: Overview of regulation service components. \bar{P} and R are selected by the data center in advance of each hour. Within each hour, those selected values determine how the data center will interpret the $y(t)$ regulation signal that is updated every few seconds.

Each hourly bid for regulation service includes two offerings from the data center: average power and the reserve capacity. The average power (notated as \bar{P} throughout works in this thesis) indicates the mean power consumption the data center wishes to consume over the next hour. The reserve capacity (notated as R) is the maximum change in power the data center is willing to go *above or below* the mean power throughout the upcoming hour.

Within each hour of regulation service, the data center is expected to match its power demand to that requested by the electricity provider. This work follows a model used by the *PJM* independent service operator [72], where the independent service operator broadcasts a regulation signal to entities offering demand response services. The regulation signal $y(t)$ is a time-varying function ranging from -1 to 1 , with a mean value of 0 . At any point in time, the data center must modulate its power demand to be equal to $y(t)R + \bar{P}$. In other words, the regulation signal indicates how high the data center's power demand must be within its bid for power and regulation capacity.

Cost savings are a key incentive to participate in demand response programs. In general, the entity offering regulation service (e.g., a data center) pays some base price for average power and is paid some value for the amount of regulation service it offers, less any penalties for failing to offer regulation service quickly or accurately enough.

The exact cost structure is specific to any given demand response program. Since the works in this thesis follow a model based on that used by the *PJM* independent service

operator, we base our cost model on theirs. We model the cost of an hour of energy as $\Pi_P \bar{P} - \Pi_R R + \Pi_\varepsilon R \varepsilon$, where $\Pi_P \bar{P}$ indicates the cost of energy purchased for the hour, $\Pi_R R$ represents the value of reserve capacity offered for the hour, and $\Pi_\varepsilon R \varepsilon$ is the cost penalty for ε percent error in tracking the regulation signal. The tracking error penalty is designed to map to the *precision score* component in PJM regulation service programs [72].

Chapter 3

Multi-Level Power Management

High-performance computing systems often employ a job scheduler and resource manager (JSRM) to ensure that system resources are well-utilized without exceeding any constraints defined by a data center operator. Power is an example of a manageable resource in a data center. Although it is possible to manage power at the cluster scope typically visible to a JSRM, there are additional concerns both above and below that level of scope. Data centers also consume power in the non-computing components of their facilities, and decisions that are made at the cluster level may have unknown performance impacts at the application level.

This chapter includes an outline of related work in multi-level power management in Section 3.1. The remaining sections demonstrate that multi-level power management solutions achieve their objectives more effectively than siloed solutions that only operate in a single level. Section 3.2 provides an opportunity analysis demonstrating that cluster power management policies make more efficient decisions when unified with prior knowledge about job power and performance properties. Section 3.3 proposes a framework that integrates online performance feedback into a cluster power manager, enabling multi-level power management without complete prior knowledge of power and performance properties. Lastly, Section 3.4 demonstrates that data center demand response policies can achieve greater cost savings by integrating a simple site power model into their cluster power management mechanisms.

3.1 Related Work

There is broad interest in reducing the carbon impact and cost impact of electricity demand in data centers while maintaining high-level quality of service commitments. A survey of energy-aware Top500 supercomputing sites indicates that facilities are interested in job scheduling, resource management, and facility design strategies to meet their high-level power objectives [53]. Another survey reveals that a lack of QoS guarantees is a strong reason that some data centers do not consider demand response in their electricity procurement strategies [67]. The rest of this section outlines related power management work with application-level and system-level components, as well as combined solutions and system-level policies that consider demand response objectives.

3.1.1 Application-Level Power Control

Application-level power management techniques benefit from awareness of application characteristics and design patterns. Typically these approaches leverage monitor and control knobs exposed by the underlying hardware in addition to software-level controls like the number of threads, logical core count, environment variables, etc. Multiple efforts fall into these categories. These efforts include PUPiL by Zhang and Hoffmann, for power-constrained resource allocation [95], PShifter by Gholkar et al. which dynamically adjusts the ratio of compute to network time for improved energy efficiency [34], EAR by Corbalan et al. which detects application loops and scales frequency for reduced energy consumption [21]. Additionally, efforts like Adapt&Cap by Hankendi et al. [37] and the online algorithm [25] by De Sensi and Danelutto leverage software control knobs to boost efficiency. All of these efforts support optimization algorithms that are application-aware, but remain oblivious of the site-specific requirements.

3.1.2 System-Level Power Control

State-of-the-art system-level power management techniques benefit from visibility into energy and power constraints of HPC facilities. Resource managers like HPE's Cray-ALPS [38] and Altair's PBS [2] offer static power management interfaces. Dynamic system-wide power management efforts like SchedMD's Slurm integrated power management [82] and POW by Ellsworth et al. [31] demonstrate the utility of steering power from system components with low power demand to components with high power demand. The RMAP power manager accepts user tolerance for job slowdown as an input to a backfilling scheduler algorithm that sets job power caps to improve throughput in over-provisioned cluster configurations [69]. The fundamental shortcoming of these system-level-only solutions is that the system-level power management mechanism remains incapable of responding to application-level design characteristics.

3.1.3 Multi-Level Power Control

In recent years, it has been well established within the HPC community that there exists a need for feedback-driven end-to-end power management control across the system stack. The HPC PowerStack Initiative [4, 94, 11] is working toward a standardized methodology for end-to-end data center power management from site down to hardware. They outline the architecture of system software within HPC systems that interact with each other to drive efficiency at different levels of granularity within the stack: at facility-level, system-level, node-level, job-level, and platform-level. Saba et al. propose a machine learning approach to predict job performance through feedback from performance counters while under CPU and GPU power caps, as part of a combined scheduling and power-capping solution [77].

Dynamo, by Wu et al. [93], proposes a framework which utilizes hardware power monitoring and controls at all layers of their data center's power delivery hierarchy. While Dynamo supports end-to-end power management, the framework relies on knowledge that

specific nodes are limited to running particular workloads. It does not directly apply to typical shared HPC scenarios where compute nodes are subjected to previously-unseen workloads by multiple users. Our work captures variable power characteristics by ensuring that our search space includes a range of system power limits and different mixes of concurrently running applications. Another multi-tier power management solution focuses on increasing the power over-provisioning ratio of hyperscale data centers by exploiting the flexibility of non-production workloads [79].

Some efforts share work across data centers to meet power objectives across data centers. For example, the Zero-Carbon Cloud project [18] relocates virtual computing workloads so that computing power demand can follow changing supply. That type of solution offers high capacity to match changing power supply, particularly when many small data centers are distributed near different energy supplies. However, that type of approach is not suitable for workloads that are expected to operate within a single HPC data center.

3.1.4 Demand-Response-Driven Power Control

Demand response programs are an avenue for data centers to reduce their energy costs, by offering regulation service on a smart grid. Many efforts underline the importance of task scheduling and power management policies for data center carbon footprint reduction and participation in demand response programs [50, 51, 98, 45].

Existing works about demand response in data centers often investigate QoS awareness without pursuing site-level power awareness. The Adaptive policy with Quality Assurance (AQA) [97] operates on server-level power metrics and controls to enable QoS-aware bidding and power management in demand response in HPC environments. The ECOGreen policy [64] focuses on QoS awareness in virtualized computing environments, and includes on-site power storage as a control.

Other work [33] explores site-level monitoring and control to meet target power levels in demand response programs. While that work does include QoS estimates as part of the bidding strategy, it does not include end-to-end QoS feedback from bidding through scheduling and power management.

Various studies have investigated job power consumption in data centers to develop better power management and resource utilization policies while considering QoS constraints. A recent study demonstrates a power consumption analysis from the system, job, and user perspectives, and touches on using job queue metadata to predict job power properties [66]. Saillant et al. further investigate job power forecasting techniques, underlining the possible use case in cluster management policies [78]. Other studies introduce QoS-aware power management policies by considering job-level features [26, 17]. Those studies continuously monitor the performance of workloads to ensure whether they meet the QoS constraints.

3.2 Job Performance Awareness in Cluster Power Management

Sizing a data center's power supply involves a trade-off between peak performance of individual workloads, and the total number of hosts available to run those workloads. Invest too little in power, and you run the risk that a cluster will perform poorly because it needs more power than can be delivered. But if you invest too much in power delivery equipment, you sacrifice capital and floor space that could have gone to acquiring more computing infrastructure. Several examples involving under-utilization of procured power exist in the real world. For example, Figure 3-1 shows the average power draw of the Quartz system at LLNL over a period of one year. The system is rated to consume 1.35 MW as indicated by the dashed line, but the average power draw remains at 830 kW. Another example is the Cori system at NERSC which is rated to consume 5.7 MW, but its peak power has only reached 4.6 MW, and it typically only consumes 3.9 MW [10].

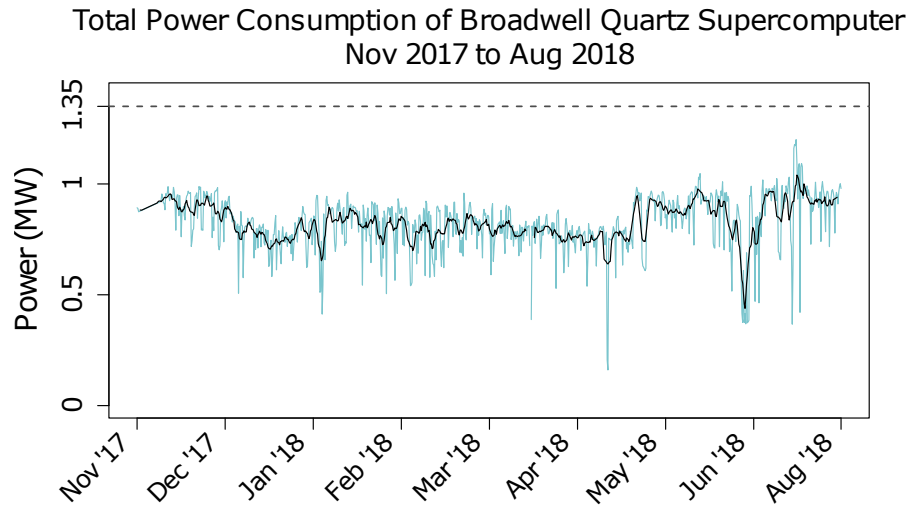


Figure 3-1: Power usage of Quartz system at LLNL over a period of one year. The dashed line shows the peak power rating of the system. The solid blue line shows actual instantaneous power usage whereas the solid black line shows moving average of the instantaneous power draw over a window of one day.

Power delivery infrastructure must ensure that a site’s total power consumption does not exceed the deliverable power capacity. The site’s power draw depends on non-uniform demands of its compute nodes, servicing different applications. State-of-the-art approaches for power management of compute nodes can be divided into two categories: (1) system-level control based on the *observed* consumption (e.g., [31, 82]), and (2) application-level control based on the *required* power (e.g., [29]). Solution-(1) has visibility into the site-level constraints, but is agnostic to the actual application demands. Solution-(2) is aware of the application design requirements, but remains oblivious to the available power at the system level. We present an opportunity analysis of approaches that leverage the best of both solutions by integrating system-level and application-level power management policies.

The **key takeaways** from this work are as follows:

- A siloed power management strategy, isolated to a single layer of the system stack, does not sufficiently optimize system performance or efficiency objectives. Moreover,

multiple such solutions running simultaneously without coordination leads to unpredictability. For example, if power limits are controlled through the same hardware interface by both a resource manager and a job runtime environment, one layer may unintentionally overwrite limits set by the other layer.

- The solution is for HPC vendors to use holistic approaches that apply power management techniques that run in tandem to address both system-level and application-level power requirements.
- Power delivery infrastructure can handle more aggressive power limits with less impact to quality of service by coordinating between layers. Such coordination can ultimately increase the amount of *science per watt*.

We present an opportunity analysis for introducing *interoperability* among multiple layers of the system software stack. The rest of this section is organized as follows. We first motivate the implementation of a multi-level system-wide power management stack. Next, we describe the power management policies, along with the synthetic benchmark and cluster environment used for evaluation. Then we describe the matrix of workload combinations and power cap levels, and the resulting impact of different power management policies across that matrix.

3.2.1 Motivation for a Unified Solution

HPC sites have different power and energy requirements, depending on their environment and the types of workloads they run. The Energy Efficient High Performance Computing Working Group (EEHPC-WG) performed a survey of Top500 sites working with energy- and power-aware job scheduling and resource management [53], showing a need for awareness of dynamic factors in both power supply and demand at a site.

Within a site, a system can be composed of many nodes that react differently under the same workloads. Even with a homogeneous system containing all nodes with the same

configuration, hardware variation can have a significant impact on workload efficiency and performance [56].

Workload characteristics impact performance and efficiency [68]. For example, workloads often have compute-intensive and non-compute-intensive phases. Non-compute-intensive phases could result from pipeline stalls due to dependencies on memory, network resources, storage devices, accelerators, thread scheduling, etc. Since CPU activity is a major contributor to system power, and can be controlled with low-latency interfaces, we study the impact of controlling CPU power to meet system power and performance constraints.

Even *within* a phase of an application, workloads can require varied demand for compute resources across processes, called *workload imbalance* in this work. Workload imbalance in bulk-synchronous workloads can make the application’s overall performance largely insensitive to the performance of some of its parts, since only the process in the workload’s critical path will have an immediate impact on aggregate performance. Several efforts have identified software critical paths for optimization in development and for modifying an execution environment to exploit that insensitivity [29, 34].

The **theme among our motivations** is that some performance and efficiency factors are most visible with a system-wide perspective, and others are more visible within the scale of a single workload execution. We evaluate power management policies with varying degrees of visibility into system constraints and workload properties. Through our evaluation, we demonstrate an opportunity for the HPC community to work toward standardization of power management policies across layers of a system stack. This is in alignment with the charter of the HPC PowerStack consortium [11].

3.2.2 Adaptive Power Capping Policies

To evaluate the potential for performance improvement, we design a new power management policy, *MixedAdaptive*. We also implement baseline power management policies for comparison purposes.

Performance-Aware System Power Management

The proposed `MixedAdaptive` policy enables a resource manager to share power across jobs in a power-aware manner. This policy’s power awareness is made available to the resource manager by a job runtime, which can search at execution time for the distribution of available power that minimizes elapsed time per iteration in a workload.

Our experiments utilize the GEOPM [29] job runtime to apply energy- and performance-aware power management algorithms. GEOPM is a software framework that provides interfaces to read platform state (e.g., energy counters, retired instructions, or CPU core frequency) and write platform settings (e.g., power caps or CPU core frequency limits). The GEOPM project comes with an HPC job runtime that also exposes MPI application state, such as time spent in MPI functions or user-annotated regions of code, as well as time spent between user-annotated loop iterations. The GEOPM HPC runtime includes multiple plugins (called *agents*) to monitor and manage power for HPC applications.

While existing job runtimes are able to utilize performance awareness, they are not currently able to establish an execution-time feedback loop with a resource manager. We emulate such a feedback loop in our experiments by first running our workloads under the GEOPM *power_balancer* agent and identifying the steady-state power consumption for each workload host. The *power_balancer* agent reduces the power limit where it does not impact performance, and redistributes that power where it can improve performance, all during execution. We use the distribution of power caps discovered by the *power_balancer* to select how much power the resource manager allocates to each job, and to determine how a job runtime would distribute that allocated power within each job. We refer to this pre-execution-time emulation of a performance feedback loop as *pre-characterization* since we are characterizing the power and performance properties of each job type. As a result of this pre-characterization step, we learn how much power is minimally needed across individual hosts running the job while remaining near baseline performance.

The **MixedAdaptive** power distribution steps are as follows:

1. Uniformly distribute the system power limit among hosts across all jobs.
2. Reduce the allocated power of each host to the amount of power needed on that host, as determined by the previously described *power_balancer* pre-characterization runs. The total amount of decreased power is now considered deallocated. If there is a significant enough power shortage, the amount of deallocated power can be as low as zero watts.
3. Uniformly distribute the deallocated power among hosts that need more power to meet their characterized performance, at most up to the characterized power. Repeat this step until no deallocated power remains, or all hosts have been assigned their needed power.
4. If there is remaining power was deallocated and not yet allocated somewhere else, allocate the remainder of power across all hosts with a weighted distribution. The weight of each host is determined by the distance from the host's minimum settable power limit to the host's allocated power from previous steps.

Baseline Power Management Policies

In addition to the previously-described *power_balancer* characterization, the baselines we compare against also use workload characterization data from the GEOPM *monitor* agent, which simply reports requested metrics of interest, such as energy and time, without modifying system behavior. We evaluate baseline policies that represent scenarios with varying mixes of knowledge about system-wide power availability and job-specific power-performance properties. Table 3.1 illustrates which types of power or performance visibility influence each power management policy, including the system-wide power budget, visibility

Table 3.1: Knowledge available to each power management policy

Policy	Budget	System Power	Job Performance
Precharacterized	X	X	✓
StaticCaps	✓	X	X
MinimizeWaste	✓	✓	X
JobAdaptive	✓	X	✓
MixedAdaptive	✓	✓	✓

into power usage across all jobs running in the system, and knowledge of performance impacts that power caps have on individual jobs.

For the **Precharacterized** policy, a user *pre-characterizes* a workload, and submits the job with a cap equal to the average power consumption at the most power-hungry node. This policy represents a scenario where there may be a system-level power cap, but users may be interested in capping their jobs without harming performance (e.g., if users are encouraged to reduce their energy consumption).

For the **StaticCaps** policy, system power is uniformly distributed to all nodes in the cluster. A *static cap* is applied for each job, using the max of average powers from all nodes in the job's *monitor* characterization run. Note that this policy's final state is the same as the initial state of the **MinimizeWaste** and **MixedAdaptive** power-sharing policies. This policy represents a scenario where the cluster power manager adheres to a total power budget (e.g., by broadcasting a single power cap to all compute nodes) but does not have knowledge about job-specific power usage or power-performance relationships.

MinimizeWaste shares system power across hosts, to minimize unused power budget. This policy statically emulates the *dynamic* approach documented in SLURM's real-time power management feature [82], which *is full-system-aware*. Our policy first distributes power caps across jobs. It then reduces the budget for low-power jobs to minimize unused (wasted) power budgets, and evenly redistributes power to high-power jobs. The power is removed from and added to jobs based on the observed *performance-agnostic* power usage (obtained from GEOPM reports) for each workload. Remaining unused power budget is

redistributed, weighted by the difference between minimum settable power and currently assigned power.

For the **JobAdaptive** policy, system power is *dynamically* shared within jobs to maximize performance, but power cannot be shared across different jobs. In other words, the policy is *not full-system-aware*. The system power cap is initially distributed uniformly across jobs. Power is further distributed among hosts within each job, based on the *performance-aware* characterization data (from GEOPM reports). If any of the nodes are assigned a power limit that exceeds an evenly-distributed power cap, then all nodes in the job have their power caps reduced by the percentage of their current power consumption that corrects that violation.

3.2.3 A Micro-Benchmark For Varied Imbalance and Power Properties

We use a synthetic kernel to provide fine-grained control of factors that dictate the energy/power signatures of a target platform. This section gives an overview of the different application characteristics that are captured by the kernel, followed by results of a characterization study that identifies energy/power bounds of the kernel. We also present a roofline analysis of the benchmark, which shows that the kernel exhibits the expected range of memory- and compute-boundedness compared to the memory bandwidth and floating-point performance capabilities of our evaluation platform.

Compute Intensity Benchmark

Our synthetic kernel³ is derived from a benchmark used by Choi et al. to evaluate a roofline model of energy [19]. Each bulk-synchronous parallel iteration of the kernel is illustrated in Figure 3-2. The typical application characteristics that impact the energy/power signature of a CPU application are as follows:

³The source code for the micro-benchmark is hosted on a public repository at <https://github.com/dannosliwcd/arithmetric-intensity.git>

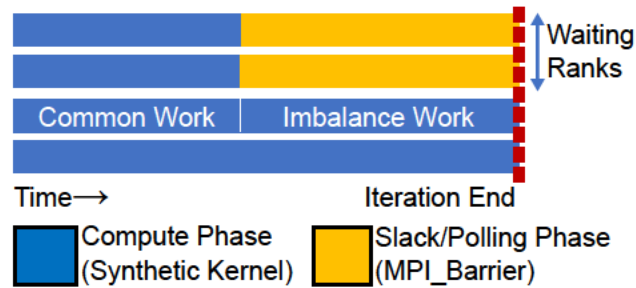


Figure 3-2: Design characteristics of the synthetic micro-benchmark used for emulating typical HPC application properties.

- **Computational intensity:** This is the ratio of compute work to memory work, in FLOPs/byte. Intensity directly impacts the maximum possible CPU throughput [89].
- **Vector length of instructions:** Power-performance trade-offs also vary with the length of vector instructions. We use vector fused multiply-add (FMA) and load instructions to capture a high ratio of instructions per cycle. Instructions with 128-bit operands use xmm registers, 256-bit operands use ymm registers, and 512-bit operands use zmm registers.
- **Excess time on a non-critical path:** Processes in large-scale bulk-synchronous and fork-join applications frequently converge at synchronizing points within the application. Such applications can end up with some processes polling at synchronizing points while making no application progress, but still consuming energy. Our micro-benchmark captures such energy sinks by controlling the work performed on the non-critical path.
- **Percentage of waiting ranks:** This is the fraction of synchronizing processes on the non-critical path. A higher percentage corresponds to a higher number of processes polling at a barrier and consuming energy without making any application progress.

We verify that the kernel covers the full spectrum of achievable throughput of the platform by overlaying the kernel’s performance on top of the system’s roofline plot. Figure 3-3

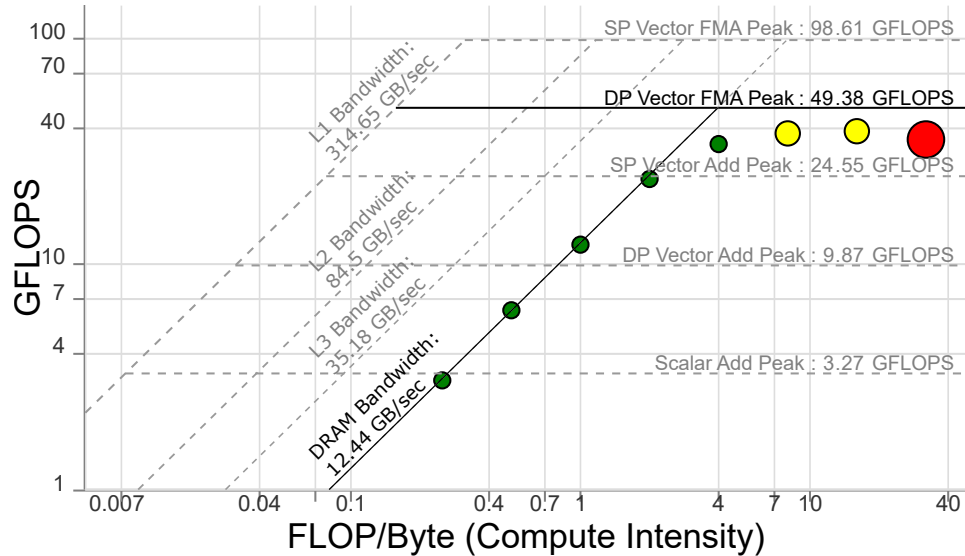


Figure 3-3: The roofline plot of the synthetic kernel when executed on the Quartz platform (described in Section 3.2.4). The plot shows computational throughput (in GFLOPS) as it relates to computational intensity kernel (FLOPs/Byte). The colored dots correspond to the runs with the synthetic kernel. The bold continuous black line corresponds to the maximum achievable performance on the target platform, given our workload configurations.

shows that the kernel’s performance at each configuration reaches the peak performance (in GFLOPS) of the system, bounded by the DRAM bandwidth and double-precision floating point vector fused multiply-add operations. The data points corresponding to the kernel behavior are depicted by colored dots in the graph, which is generated by the Intel Advisor tool [41].

Workload Characterization

For our experiments, we need to know: (a) the maximum power each workload consumes under no power constraints, and (b) the minimum power each workload needs to complete execution. Greater differences between these metrics indicate more opportunity for leveraging application awareness in power-constrained scenarios.

We obtain Metric-(a) by executing each workload with the GEOPM *monitor* agent across 100 test nodes. The heat map in Figure 3.4 summarizes our measurements. We

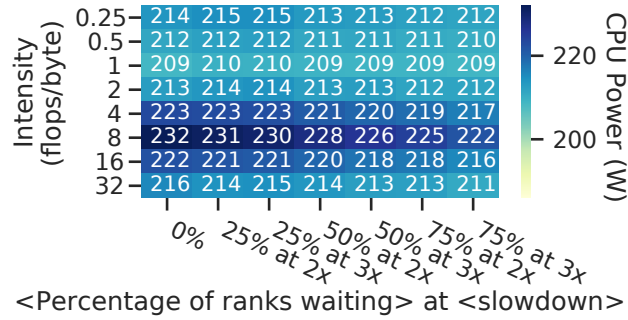


Figure 3.4: Total CPU power per node for different workload configurations, when running the benchmark variant that uses 256-bit (ymm) vector registers with no power limit under the GEOPM *monitor* agent. This non-capped power consumption is largely insensitive to imbalance.

obtain Metric-(b) by observing the actual power consumed by each workload under the performance-guided GEOPM *power_balancer* agent when subjected to an average power budget equal to the total TDP (Thermal Design Power) of each node. The resulting measurements are shown in Figure 3.5.

3.2.4 Experimental Environment

This section describes methods we follow to design our evaluation grid. This setup is divided into three parts. First, we explain the computing environment in which we run our experiments. Next we describe how we select combinations of workloads to evaluate in that environment. Lastly, we outline how we select power budgets for those workload mixes.

We limit the scope of our investigation to focus on workload CPU power. Other work may explore a holistic data management system by also considering other major contributors to power and performance, such as network fabric, memory hierarchy, secondary storage, accelerator offloads, and cooling infrastructure. While we do not consider hardware variation in our comparisons, we control for it by selecting similarly-performing nodes for our experiments.

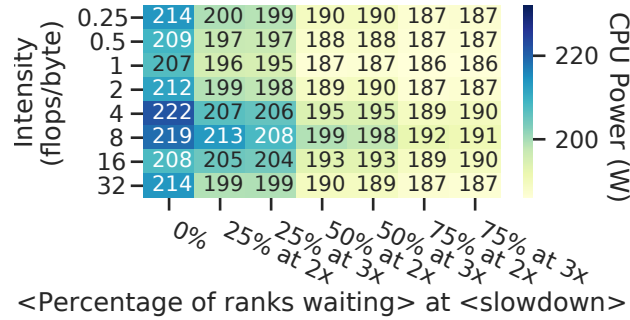


Figure 3-5: Total CPU power per node for different workload configurations, when running the benchmark variant that uses 256-bit (ymm) vector registers under the GEOPM *power_balancer* agent. The most significant reductions in power from the *monitor* case are in the mid-intensity range. The apparent vertical bands indicate that the percentage of waiting ranks have a strong effect on average needed power, which motivates using a performance-aware power management policy in those cases.

Quartz Computing System

Our experiments are executed on the LLNL Quartz computing system, controlling for hardware performance variation by dropping nodes with outlier performance properties from our tests.

Compute Nodes in Quartz The properties of Quartz are summarized in Table 3.2. This system provides large-scale access to fine-grained power management knobs via the MSR-safe Linux Kernel module [52]. Our experiments utilize 34 cores per node for the benchmark, leaving the remaining two cores for monitoring processes and system services. Although these experiments are executed on a single Intel architecture, they can be ported to other architectures (Intel and non-Intel) by leveraging GEOPM’s portable plugin infrastructure. We use CPU-package-scoped RAPL power controls on this system. Systems that also have GPUs may benefit from using platform power controls (such as PSYS RAPL) that are scoped to include CPUs and GPUs.

Table 3.2: Quartz System Properties

Operating System	TOSS 3 (Red Hat Enterprise Linux 7.8)
CPU	Intel Xeon E5-2695, dual-socket
Thermal Design Power	120 W per CPU socket
Minimum RAPL Limit	68 W per CPU socket
Base Frequency	2.1 GHz
Cores Per Node	36

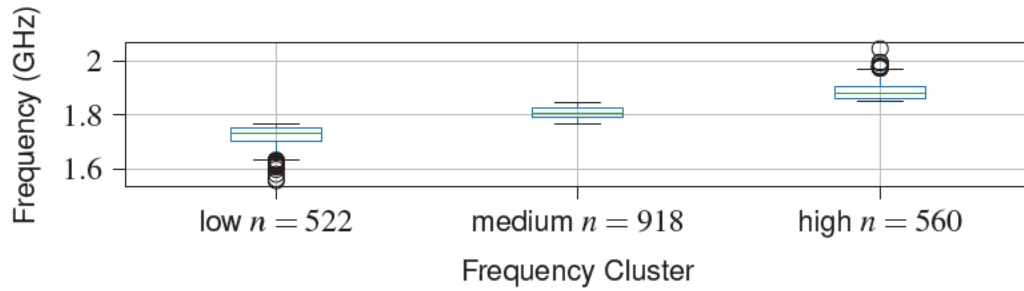


Figure 3-6: Achieved frequencies of 2000 nodes in the Quartz cluster, under 70 watt CPU power limits. Whiskers extend to the nearest samples beyond the inter-quartile range (IQR) by $1.5 \cdot \text{IQR}$. Circles show samples beyond the whiskers.

Hardware Variation in Quartz Many of our experiments are running under tight power constraints, so it is important to consider the impact of hardware variation when running under a power cap [57, 71]. To reduce the impact of hardware variation on our final results, we identified a subset of the cluster’s nodes that perform similarly with our workload. To do this, we first monitored the achieved frequency of each node in the cluster while running our most power-hungry workload configurations under a low power limit. We used k-means clustering over the achieved frequencies to partition the nodes into three groups. We used the 918 *medium* frequency nodes in Figure 3-6 for our experiments so that our results reflect a central tendency of performance in a significant portion of our test system. The unused 1082 nodes achieved CPU frequencies spanning a 400 MHz range under a 70 W per-socket power limit.

Workload Mixes

Our experiments include workload mixes that are composed of multiple configurations of the benchmark described in Section 3.2.3. The combinations of workloads in the mixes are summarized in Table 3.3. We include some mixes that are expected to demonstrate best-case behavior for each of the baselines, as well as high power, low power, and random job mixes.

`NeedUsedPower` is characterized by some jobs with low average power and one job with high compute intensity, and where all used power is needed for performance, as determined by our *power_balancer* characterization runs. This workload mix is expected to be the best-case mix for the `MinimizeWaste` policy, which should decrease the power allocated to the many low-power jobs and use that spare power to help the high compute intensity job finish more quickly. Since the observed power is similar to the needed power for performance, policies that depend on performance-awareness are not expected to have an advantage.

`HighImbalance` is characterized by a single, imbalanced job across all nodes. This workload mix is expected to be the best-case mix for the `JobAdaptive` policy, which should decrease the power to nodes within the single job that do not impact elapsed execution time while increasing the power to nodes that do impact elapsed execution time. Since the workload is highly imbalanced within its bulk-synchronous loops, policies without performance awareness are not expected to have an advantage.

`WastefulPower` is similar to the `NeedUsedPower` mix in the sense that it exhibits a range of average power levels. But it is different from that mix because the average power consumed by an unconstrained workload run significantly differs from the power consumed when the workload is balanced for performance under a generous power cap. This workload mix is expected to be the best-case mix for the `MixedAdaptive` policy, which should redistribute power within each job to maximize performance. As a result, it may reduce an imbalanced job's total allocated power to less than the total allocated power of `MinimizeWaste`, and it can share power across jobs unlike `JobAdaptive`.

Table 3.3: Workloads in Each Workload Mix

Workload Description		NeedUsedPower	HighImbalance	WastefulPower	LowPower	HighPower	RandomLarge
xmm (128-bit) vector registers	Balanced	No waiting ranks	x	x	x	x	✓
		0.25 FLOPs/byte	✓	x	x	x	✓
		16 FLOPs/byte	✓	x	x	x	x
		32 FLOPs/byte	x	x	x	x	x
		25% waiting ranks					
		0.5 FLOPs/byte	x	x	x	x	✓
		16 FLOPs/byte	x	x	x	✓	x
		32 FLOPs/byte	x	x	x	✓	x
		50% waiting ranks					
		0 FLOPs/byte	x	x	x	x	✓
		0.25 FLOPs/byte	x	x	✓	x	x
		0.5 FLOPs/byte	x	x	✓	x	x
		1 FLOP/byte	x	x	✓	x	x
		8 FLOPs/byte	x	x	✓	x	x
		16 FLOPs/byte	x	x	✓	x	x
		32 FLOPs/byte	x	x	✓	✓	x
		75% waiting ranks					
		32 FLOPs/byte	x	x	x	✓	x
		25% waiting ranks					
		16 FLOPs/byte	x	x	x	✓	x
	32 FLOPs/byte	x	x	x	✓	✓	
	50% waiting ranks						
	16 FLOPs/byte	x	x	✓	x	x	
	32 FLOPs/byte	x	x	✓	✓	x	
ymm (256-bit) vector registers	No Imbalance	No waiting ranks	x	x	x	x	✓
		0 FLOPs/byte	x	x	x	x	✓
		0.25 FLOPs/byte	x	x	x	x	✓
		4 FLOPs/byte	x	x	x	x	✓
		8 FLOPs/byte	x	x	x	x	✓
		32 FLOPs/byte	✓	x	✓	x	x
		25% waiting ranks					
		4 FLOPs/byte	x	x	x	x	✓
		8 FLOPs/byte	x	x	x	x	✓
		50% waiting ranks					
		8 FLOPs/byte	x	x	x	x	✓
		75% waiting ranks					
		8 FLOPs/byte	x	x	x	x	✓
		25% waiting ranks					
		0 FLOPs/byte	x	x	x	x	✓
		0.25 FLOPs/byte	x	x	x	x	✓
		2 FLOPs/byte	x	x	x	x	✓
		8 FLOPs/byte	x	x	x	x	✓
		32 FLOPs/byte	x	✓	x	x	x
		50% waiting ranks					
	8 FLOPs/byte	x	x	x	x	✓	
	75% waiting ranks						
	16 FLOPs/byte	x	x	x	x	✓	

`LowPower` has the nine lowest-power workload configurations, `HighPower` has the nine highest-power workload configurations, and `RandomLarge` has nine jobs selected from a random shuffle. Each of these mixes has 100 nodes per job.

In order to select the appropriate microbenchmark configurations and system power caps to use in the above mixes, we characterize the power and performance properties of the microbenchmarks on the nodes that we identified as similar earlier in this section.

Selection of Power Budgets

We evaluate each power management policy at three different system-wide power budgets (*min*, *ideal*, and *max*) representing degrees of over-provisioning. The range of power caps we test covers the power capping region within which policies produce different power allocations relative to the needs of the workload mix. Power caps less than *min* result in all policies producing the same configuration as `StaticCaps`. Power caps greater than *max* result in all policies allocating at least as much power as `Precharacterized`. The quantitative values of the three power budgets are described in Table 3.4.

The *min* power budget represents an aggressively over-provisioned system, where there is little capacity to share power across workloads. Its budget is selected by determining which workload in the mix has the least power consumed by a single node under the *performance-aware* characterization described in Section 3.2.3. The system is allocated enough power to provide that amount to each node.

The *ideal* power budget represents over-provisioning that is ideal for a given workload mix. This budget is used in our experiments to evaluate cases where there is significant opportunity to improve performance by sharing power across workloads. Its limit is selected by summing the power used by each node for all workloads in the mix, as determined by the *performance-aware* characterization described in Section 3.2.3. The system is allocated that summed total of power.

Table 3.4: Power Budgets for Each Workload Mix

Workload Mix	min	ideal	max*
NeedUsedPower	167 kW	171 kW	209 kW
HighImbalance	141 kW	163 kW	209 kW
WastefulPower	136 kW	144 kW	209 kW
LowPower	138 kW	152 kW	209 kW
HighPower	140 kW	177 kW	209 kW
RandomLarge	139 kW	164 kW	209 kW

*TDP of all CPUs is 216 kW

The *max* power budget represents a conservatively over-provisioned system, where there is little need to share power across workloads. Its budget is selected by determining which workload in the mix has the most power consumed by a single node under the *uncapped* characterization described in Section 3.2.3. The system is allocated enough power to provide that much to each node.

Enforcing Power Budgets

Some of the policies evaluated in this work utilize uniform power caps across all compute nodes in a job. We use those power caps as inputs to the GEOPM *power_governor* agent for those experiments. That agent does not support non-uniform power caps, which are needed to implement some of our power budgeting policies. So we implement a new GEOPM *host_power_governor* agent⁴ that allows us to set per-host power caps for each job. We use the *host_power_governor* to enforce power limits in policies that non-uniformly assign power across hosts in a job.

3.2.5 Results & Takeaways in Unified Cluster Power Management

In this section, we evaluate the impact of various factors on elapsed time, energy, and power usage. Specifically, we discuss:

⁴Available at https://github.com/dannosliwcd/geopm/tree/dcw/host_power_governor/tutorial/host_power_governor

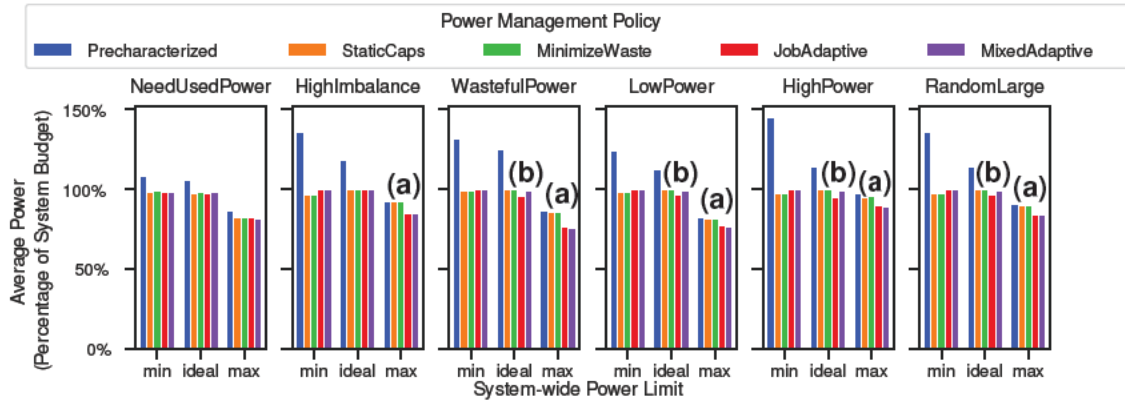


Figure 3-7: Mean power used by each policy, across workload mixes (Section 3.2.4). Bars above 100% mean a policy exceeds the system-wide power budget. Bars below 100% show opportunity for more aggressive over-provisioning, or a missed opportunity to distribute power where it can be utilized. Annotations show where policies had large differences in power usage. Marker-(a) highlights max-power-limit cases where performance awareness enables less power usage under relaxed limits. Marker-(b) highlights ideal-power-limit cases where system power awareness enables more power utilization under a moderate limit.

1. How system power limits impact performance under different policies.
2. Trade-offs of different levels of visibility into a system's resources and a workload's performance.
3. How different workload mixes react to power management policies.

Impact of System Power Cap

Support for hardware over-provisioning is one of the use cases for a system-wide power budget. We evaluate the impact of the system-wide power budgets described in Section 3.2.4, which are meant to represent different levels of over-provisioning.

As shown in Figure 3-7, lower system-wide power limits, which correspond to more aggressively over-provisioned systems, result in greater utilization of power capacity. The Precharacterized policy is omitted from further plots since it can only stay within the budget in the high power cap case. Note that system-unaware policies may under-utilize the available power in cases with balanced supply and demand or with a surplus of power.

In cases with balanced power demand (marker-(b)), the `JobAdaptive` policy is unable to utilize all of the budgeted power. This is because some power is set aside for workloads that cannot use all of their allocated power, while other workloads remain power-bound. The policies with system awareness avoid this by sharing the budget across jobs.

In cases with high power budgets, annotated with marker-(a), the amount of power being provisioned is higher than needed, regardless of performance-awareness in the policy. The difference between policies in these low-power-utilization cases is observable when we run the workloads and evaluate their energy impacts. In Section 3.2.5, we note that when job awareness enables reduction of a power budget, it results in improved energy efficiency. This efficiency improvement grows stronger at higher power limits where there is more opportunity to decrease power without impacting performance.

The impact of power caps is also visible as trends in mean time savings and energy savings in Figure 3.8. The time-saving opportunity decreases as system-wide power budget increases, with a maximum opportunity at 8% (marker-(e)) in the *min* power case for the `HighPower` workload mix. In that maximum opportunity scenario, the `MixedAdaptive` approach achieved about 7% time savings over the `StaticCaps` baseline (marker-(e)). The energy-saving opportunity increases as power budget increases, with a maximum opportunity of 11% energy savings in the `WastefulPower` workload mix, when subjected to the maximum power budget (marker-(d)).

Takeaway: Sites incorporating dynamic power management policies benefit from energy savings. These savings increase with the amount of surplus power budget.

Impact of Job Awareness

We evaluate a static power management policy and three policies (Section 3.2.2) that depend on workload characteristics. Of the dynamic policies, only `JobAdaptive` and `MixedAdaptive` are aware of performance of running workloads. We note takeaways about job awareness from traits that these policies share with each other, and not with other policies.

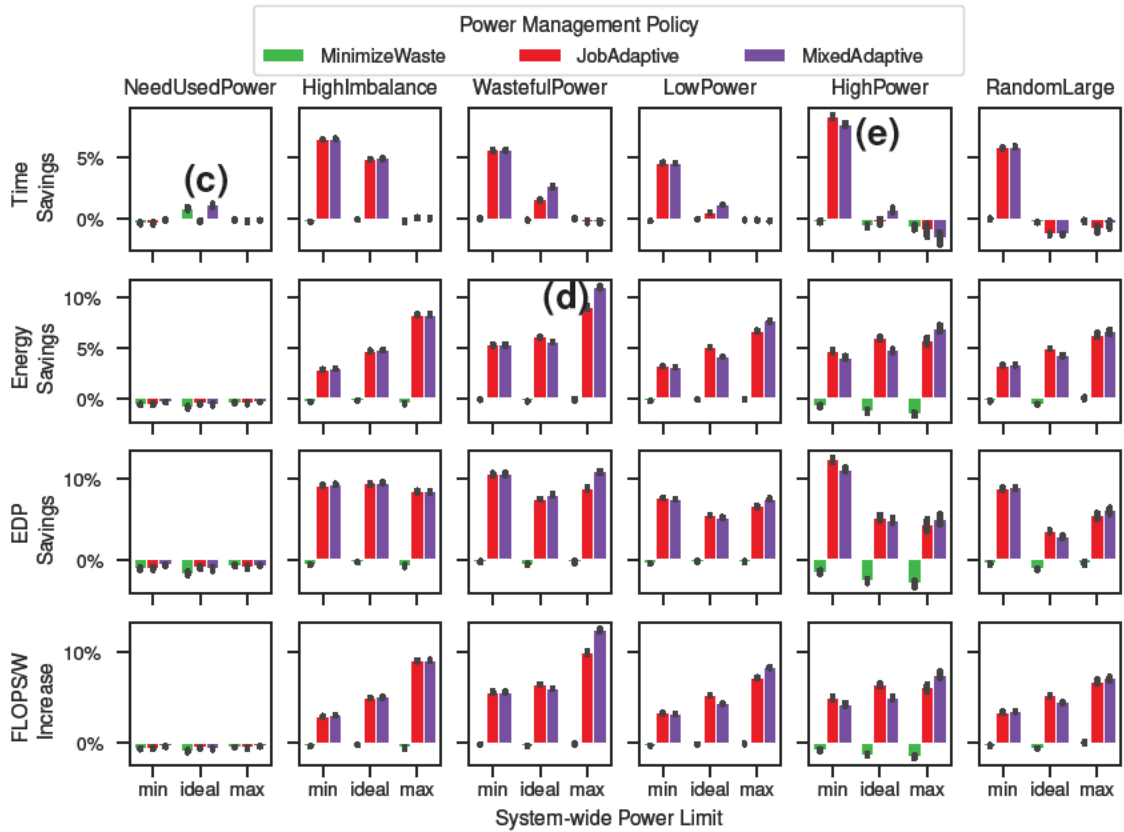


Figure 3-8: The grid above depicts the impact of three system-wide power management policies (Section 3.2.2) on different workload mixes (Section 3.2.4) and power budgets (Section 3.2.4). The height of each bar is quantified as a mean percentage decrease from the StaticCaps baseline measurements. Each row reports a metric of efficiency or performance. Each column reports a workload mix. Marker-(c) highlights an ideal-power-limit case where MinimizeWaste saves more time than JobAdaptive. Marker-(d) highlights a max-power-limit case where MixedAdaptive saves more energy than JobAdaptive. Marker-(e) highlights the case with the most time savings opportunity. Error bars show the 95% confidence interval, calculated over measurements from 100 iterations per benchmark configuration.

The plots in Figure 3.8 show how energy, elapsed time, energy-delay product, and FLOPS per watt are impacted by each of the policies for different workload mixes. All metrics are reported as a percent improvement from the `StaticCaps` policy, which makes no workload-dependent changes to power allocation. The `Precharacterized` policy is omitted since it is unable to operate within the budgeted power in most cases.

One trend to note is that `JobAdaptive` and `MixedAdaptive` policies tend to perform similarly in the *min* (aggressively over-provisioned) and *max* (conservatively over-provisioned) power levels. Since the *min* power level has no opportunity for power sharing, both policies remain in their initial state, where power is distributed uniformly across hosts (refer to step 1 in Section 3.2.2).

Those policies do have a chance to decrease their allocated power in the *max* power level case, which is why they are able to noticeably reduce total energy consumption in those runs. But, the cross-job power-sharing capability of the `MixedAdaptive` policy does not differentiate it from the `JobAdaptive` policy since there are no power-bound jobs that need to make use of that surplus (the lack of power-bound jobs is a design property of the *max* power level).

Takeaway: HPC sites incorporating application awareness have increased opportunities for energy savings under a system power limit, compared to sites with application-agnostic solutions.

Impact of System-Wide Resource Awareness

The `MinimizeWaste` policy also depends on workload characteristics. But unlike the `JobAdaptive` policy, it is not aware of the workload's relationship between performance and power consumption. Instead, it relies only on observing and limiting power consumption from a resource manager's level of visibility in a cluster. `MixedAdaptive` also has this level of cluster power visibility, in addition to job awareness.

`MinimizeWaste` and `MixedAdaptive` both excel when workloads are all balanced with

high compute intensity, and some workloads are more power-hungry than others. They save time in the `NeedUsedPower` mix by steering unused power toward power-hungry workloads.

The `MixedAdaptive` policy often performs similarly to the `JobAdaptive` policy. The cases where it stands out (e.g., marker-(**d**)) indicate scenarios where added resource awareness offers new opportunities. Cases that favor resource awareness offer additional energy savings when time savings are near equal. They also save time in the `WastefulPower` workload mix.

The energy savings of the `MixedAdaptive` policy are further improved over the savings of the `JobAdaptive` policy in some max power limit cases, most notably in the `WastefulPower` workload mix annotated with marker-(**d**). These savings are a result of the final stage of the power allocation strategy used for these policies, described in Section 3.2.2. After all needed power has been distributed to the workloads, the `JobAdaptive` policy continues to distribute the remainder power within each workload to the nodes that need the most power. In contrast, the `MixedAdaptive` policy uses its system-wide resource awareness to distribute the remaining power *across* all workloads to the nodes that need the most power. The `JobAdaptive` policy is not able to take the most efficient actions with the remainder of the power budget because it does not have system-wide power awareness.

Takeaway: Resource awareness alone has small benefits, but when combined with application awareness, can have greater benefits than either application awareness or resource awareness alone.

Impact of Workload Mixes

The running workloads significantly impact savings opportunities. While most of the mixes show significant time savings and energy savings for both `MixedAdaptive` and `JobAdaptive` policies, the `NeedUsedPower` mix shows no opportunity for energy savings, and small time savings for the `MinimizeWaste` and `MixedAdaptive` policies.

Both differences occur because the mix of jobs is composed of workloads that actually

need all of their consumed power to avoid loss of performance. There is no opportunity to save energy for those performance-constrained policies because any reduction in power consumption will also cause the applications to take more time to finish running. Workloads under the `MinimizeWaste` policy can run faster as indicated by marker-(c) because the high power jobs are able to use the slack power budget from the low power jobs. The `JobAdaptive` policy can not see a benefit to elapsed time from re-balancing because the workloads are already balanced.

Takeaway: Application characteristics dictate the amount of surplus power available within for that application. Based on the mix of applications within a job schedule, this surplus power can be steered towards improving the efficiency of the entire system.

3.3 Integrating a Performance-Aware Job Runtime in Cluster Power Management

The previous section demonstrates opportunities for more effective power management when a cluster power cap is distributed across jobs using both cluster-level power visibility and job-level performance awareness. That work pre-characterizes the power-performance properties of jobs, and distributes power to jobs based on a pre-selected cluster power cap. In this section, we enable performance-aware power management policies to work with time-varying cluster power objectives and to improve their effectiveness when job power-performance models are inaccurate or absent. We achieve that goal by using an online feedback mechanism between the cluster and job power management layers.

Global clean energy investment is expected to double in the next decade [40]. Increased reliance on renewable energy supply increases grid exposure to nature-driven supply volatility. Electricity service providers have many tools at their disposal to combat imbalance between electricity supply and demand, including demand response programs where consumers cooperatively adjust their demand at the request of electricity providers in exchange

for monetary incentives. Participation in such programs helps increase grid flexibility, but the current rate of adoption of energy storage and demand response programs needs to double in order to offer sufficient grid flexibility for 2050 net-zero carbon emission goals [39].

Prior work has demonstrated that data centers are suitable capacity providers in demand response programs, and have proposed policies to maximize cost savings while meeting a data center’s quality of service (QoS) commitments to its users [16, 97, 96]. Those prior works assume that application power and performance properties are known in advance and do not change after profiling the applications. However, power and performance properties may not be known (e.g., if some jobs are not pre-characterized or if they are incorrectly identified as pre-characterized job types) and properties may change from run to run (e.g., due to hardware performance variation, or changes in application inputs).

We design a multi-tiered power management framework that is robust to scenarios where job performance diverges from precharacterized expectations while enabling a high degree of power sharing across jobs. We measure missed performance opportunities from inaccurate power-performance models in a power management policy that is purely driven by precharacterized job properties and demonstrate that our framework enables the power manager to recover the lost performance.

We implement a cluster demand response policy and evaluate our implementation on 16 physical compute nodes. We explore the impacts of power management policy choices made when the power and performance properties of some jobs are unknown. Lastly, we simulate demand response scenarios, augmenting our measured power-performance properties with randomized job performance variation⁵ on a 1000-node data center to explore opportunities for further policy enhancements.

Through evaluations on a 16-node cluster, we show that our multi-level power management framework enables a demand response policy to track time-varying power targets

⁵In this work, performance variation broadly refers to differences in job execution time from one run to another.

with 90th percentile error of 18% while speeding up some jobs by 4% compared to uniform power caps. Through simulations, we show QoS⁶ degradation when the power manager’s job characterizations are unaware of performance variation, and identify opportunities to integrate performance feedback.

The rest of this section begins with an outline of our multi-tiered power management framework for time-varying power objectives in Section 3.3.1. We describe the architecture of an instance of our framework used for demand response in Section 3.3.2. Section 3.3.3 describes the experimental methodology we follow to evaluate our system in Section 3.3.4. We discuss this work’s outcomes and challenges in Section 3.3.5.

3.3.1 A Framework for Multi-Tiered Power Management: ANOR

Our goal is to design a framework that tracks time-varying cluster *power targets* under user QoS constraints where power-performance sensitivity may change over time. Potential applications for changing power targets include grid-aware power management scenarios where data center operators may react to time-varying carbon intensity, changing power tariffs, or demand response events. Time-varying QoS impacts may surface in many ways, such as seeing a new job type execute for the first time or running a previously seen job type under new conditions. In Section 3.3.2, we describe a specific implementation that works toward these goals to enable QoS-constrained demand response.

To that end, we propose *ANOR* (Attach Nested-Objective Runtimes), a multi-tiered design framework that manages a cluster with power objectives and QoS impacts that change over time. At a high level, this framework attaches two tiers of monitoring and control mechanisms, shown in Figure 3.9. A cluster tier orchestrates entities in a job tier to optimize for application performance under time-varying constraints over total cluster power consumption. The tiers interact by sending control messages down to jobs from the cluster tier and by sending status messages up from the job tier.

⁶QoS in this work corresponds to the time a user must wait for a submitted job to complete.

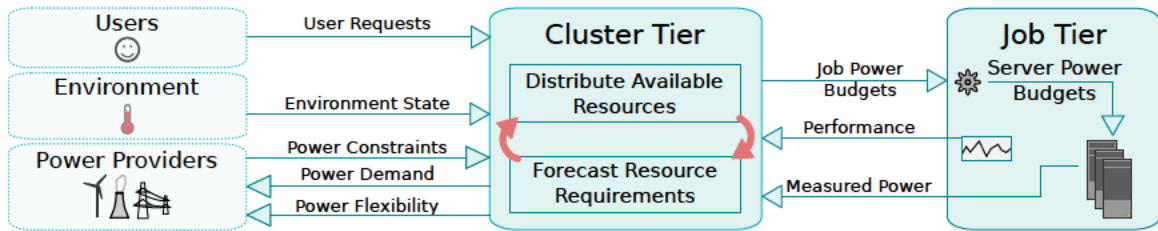


Figure 3-9: Tiers of power management entities in the ANOR framework. The cluster tier distributes the cluster’s resources to members of the job tier. The cluster tier makes its decisions based on external state and aggregated power and performance data that comes from the job tier.

The cluster tier owns policies that relate application performance and cluster resources to external state. Performance and power properties are passed to this tier from the job tier while the job executes. The cluster tier may receive power caps from a higher-level entity such as a facility power manager, or it may receive information that helps determine an appropriate cluster power cap on its own. Examples of higher-level information include power price information, carbon intensity of the power source, the facility’s operational analytics, and demand response power targets. In this work, we demonstrate the efficacy of ANOR in conducting data center demand response.

The job tier manages the relationship between power caps and application performance within each job. This tier receives a power cap from the cluster tier and receives execution-time performance feedback from an application. It uses both sets of data to determine how to distribute the power policy across the job’s compute resources, and sends the job’s power and performance properties to the cluster tier.

This framework enables cluster power management policies to integrate performance awareness by delegating job-specific tasks to the job tier. We outline an example implementation in the next section.

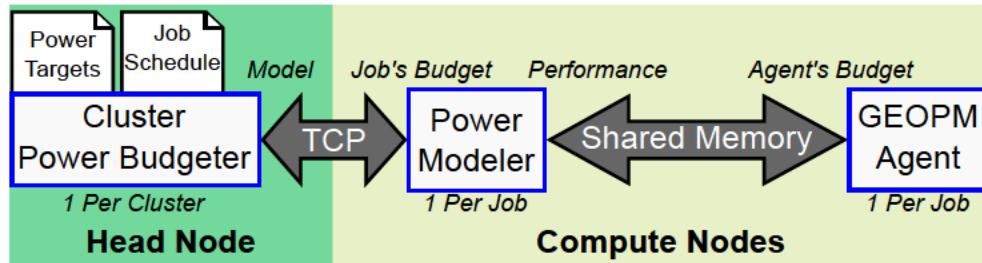


Figure 3-10: Our implementation of ANOR for demand response. A single cluster-tier process communicates over TCP with one job-tier power-modeling process per job, sending down power budgets and receiving power models. The power-modeling process sends power budgets to one GEOPM agent instance per job, over shared memory, and receives performance metrics from the agent.

3.3.2 Implementing ANOR for Demand Response

We demonstrate the ANOR framework by implementing it on the Global Extensible Open Power Manager (GEOPM) framework [29] to realize a performance-aware demand response policy⁷. The demand response mechanism is based on the AQA [97] demand response bidder, job scheduler, and power budgeter. This section outlines the architecture of our ANOR implementation for demand response.

We use the *GEOPM HPC runtime* to monitor and control node power consumption and to monitor job performance in our cluster. GEOPM provides *signals* to monitor applications (e.g., a count of times a region of code was entered) and hardware (e.g., power and energy), and provides *controls* for the hardware platform (e.g., CPU power caps). GEOPM also includes a software framework to define *agents* that periodically respond to observed signals and update controls in response to those signals while a job executes. Agents on multi-node jobs interact across nodes through a hierarchical communication interface. The root level of that agent hierarchy has a software interface, called the *GEOPM endpoint interface*, that can be used to dynamically write new objectives and read summarized state updates from agents.

⁷Available at <https://github.com/dannosliwcd/geopm/tree/dcw/site-power>

We implement a software layer that bridges the GEOPM endpoint to a job-tier power and performance model, illustrated in Figure 3.10. This job-tier endpoint communicates through a TCP connection to a job-tier management process running on the head node in our cluster. The cluster-tier manager periodically reads new cluster power targets from a file, reads received messages from nodes running jobs, calculates how to distribute available power to nodes, and sends messages to inform each job-tier endpoint of the job's updated power cap.

Cluster Power Budgeter

In this work, we consider two methods to allocate a cluster's available power to its compute nodes. We refer to the allocation mechanism as a power budgeter. An effective power budgeter ensures that the available power is sent to components that use their power to improve system performance.

A single process balances the cluster's power budget from the head node in our cluster. For experimental repeatability (discussed in Section 3.3.3), this process reads power targets and a job submission schedule from files. Other implementations may elect to schedule jobs through an independent scheduling mechanism, which is also compatible as long as new jobs initiate connections with the power budgeter.

The two budgeting policies we evaluate in this work are illustrated in Figure 3.11. Different power caps are selected by each of these policies when two jobs with different power-performance relationships run at the same time. The *power-balancing* approach aims for both jobs to operate on the same percentage of power consumption across their range of achievable power (the vertical dashed line). The two jobs have different performance impacts at that power ratio. In contrast, the *time-balancing* approach aims for both jobs to degrade performance by the same amount (the horizontal dashed line), but each job consumes a different amount of power.

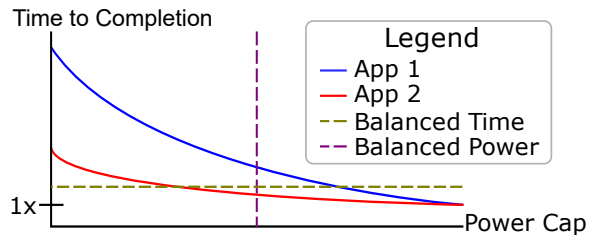


Figure 3-11: Comparison of balancing mechanisms used by two power-capping methods. Solid lines show the power-performance curves of two hypothetical applications. Dashed lines show how a time-oriented power balancer and a power-utilization-oriented power balancer would select power caps to assign to the applications.

Power Modeler

The cluster power budgeter uses power models to determine how to efficiently distribute a power budget. Each model relates a job’s rate of progress to a CPU power cap. The modeler receives an *epoch* count (i.e., count of main loop iterations) from the GEOPM agent layer via the GEOPM endpoint interface. The modeler records the time since the last epoch update, and the average power cap applied over that time span. We fit $T = AP^2 + BP + C$ for T seconds per epoch and power cap P watts below TDP. We re-train the model when at least 10 new epochs have been recorded. Jobs that report no epochs or that have yet to build a model use a default model. Our contributions include an evaluation of performance impact from different default models (Section 3.3.4).

GEOPM Agent

The GEOPM agent enforces the budgeted power caps and feeds application performance to the modeler. The agent resides on the same node as the modeler, and they communicate with each other over shared memory through the GEOPM endpoint interface. There is a single agent per job, and an agent runs one process on each compute node used by the job. When the endpoint sends a new power cap to a job’s GEOPM agent on one node, the agent sends the power cap over GEOPM’s internal communication tree to the other agent instances.

The agent reports the current GEOPM *epoch count* to the endpoint interface. The epoch count is incremented each time all processes in the monitored program reach an instrumented `geopm_prof_epoch()` call in the monitored program's main compute loop. We modified the GEOPM *power_governor* agent to write epoch count to the endpoint.

Cluster-Tier Policies to Track Power Targets

The goal of our cluster policy is to achieve a target level of power consumption across the cluster while distributing performance loss across jobs. This policy enables performance-aware power capping without knowing the relationship between every job's power and performance in advance. We achieve that goal by delegating power-performance modeling to the job tier, as described in Section 3.3.2.

Cluster tier policies define how the job tier and external entities interact with each other. We explore two types of policies that work with each other to meet QoS objectives while participating in demand response programs. A resource-forecasting policy aims to maximize the cluster's power flexibility under job performance constraints, and a power distribution policy maximizes job performance under time-varying cluster power constraints. The three main cluster tier tasks to implement those policies are power forecasting, job scheduling, and power capping.

Forecasting Power Demand

In our demand response scenario, the resource-forecasting policy determines how much *average power* the cluster should request and what range of power flexibility the cluster should offer as *reserve* for demand response. The bidding decision is made once per hour, influencing the range of power targets that will be received until the next bid. The data center must enforce time-varying power caps that span the average power plus or minus the reserve. New power targets arrive once every few seconds.

Scheduling Jobs

We base our scheduling and power management mechanisms on those introduced by the AQA policy for demand response with QoS assurances [97]. AQA assigns each job type to its own work queue. Each queue is assigned a weight of node allocations that is tuned over simulations of expected power-constraint and job-submission scenarios. Compute nodes are allocated so that so queues with greater weight are assigned more nodes.

AQA searches for queue weights and demand response bids (average power and reserve) that reduce electricity cost under constraints for QoS and power-tracking error. We set a power-tracking constraint allowing no more than 30% error for at least 90% of the time. Error is calculated as distance between the measured power and the target power, divided by the reserve. Power caps are applied uniformly across active nodes.

AQA assumes that all jobs are classified into precharacterized job types with known power-performance relationships. We use the existing job-weight training mechanism in AQA to support jobs with types that are not yet known at the time AQA is trained. For each unknown job type in the user submission queue during AQA training, we simulate a known minimum execution time (which may be provided at launch time, similar to setting a job's time limit). We simulate the job's achievable power-demand range and maximum slowdown (i.e., at the minimum power cap) to be randomly sampled from those of known job types.

Controlling Power Consumption

We evaluate two cluster-tier power-budgeting policies that utilize information from the job tier. One policy is performance-unaware, and the other utilizes performance awareness to balance the expected slowdown of each job in the cluster.

The performance-unaware balancer follows the power capping rules used by AQA [97]. In this approach, we select node power caps that are scaled equally between the minimum and maximum power achievable for all jobs. We select a single γ value that lets total power equal the power budget for:

$$p_{j,cap} = \gamma(p_{j,max} - p_{j,min}) + p_{j,min},$$

where each $p_{j,cap}$ is a job's selected power cap, $p_{j,min}$ is a job's minimum power consumption, and $p_{j,max}$ is a job's maximum power consumption.

The performance-aware balancer follows a similar approach, but balances the expected slowdown of each job instead of balancing the expected power consumption of each job. That is, we select the expected-slowdown limit s that lets the total power cap utilize the full power budget for:

$$p_{j,cap} = P_j(sT_j(p_{j,max})),$$

where each P_j function models a job's power caps as a function of execution time, and each T_j function models that job's execution time as a function of power caps.

3.3.3 Experimental Methodology

Our experiments are designed with two goals in mind. First, we quantify the real-system impact that a multi-tiered power control mechanism has on a cluster. Next, we evaluate the implications of using such a mechanism at larger scale by simulating a 1000-node cluster. We perform both evaluations in demand response scenarios where the cluster is subjected to power targets that vary with time.

This section describes the benchmarks used in our evaluations, describes how we select QoS constraints, describes how we generate job schedules, describes the tools we use for performance and power measurement, and describes the system on which we execute our cluster experiments.

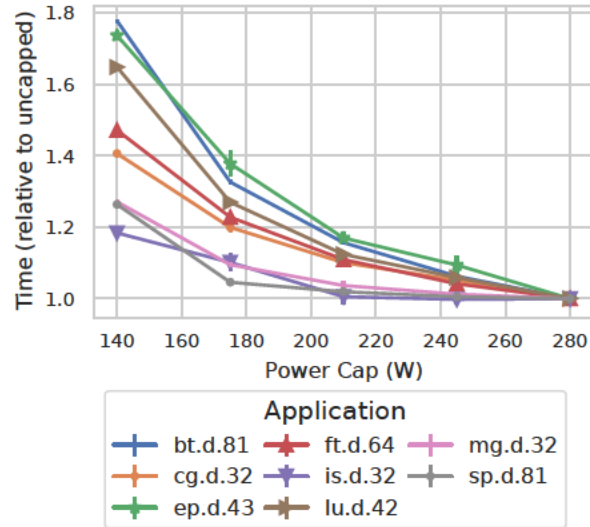


Figure 3-12: Execution time of each job type under varied power caps, relative to the execution time at a 280 W CPU power cap per node. Error bars show standard deviation over 10 runs.

Benchmarks

We evaluate multiple combinations of concurrently executing benchmarks from the NAS Parallel Benchmark suite [60] across our simulated and real-cluster experiments. The power-performance relationship of each job type is shown in Figure 3-12. The name of each job type follows a format of *benchmark-name.input-problem-class.process-count*.

We insert a `geopm_prof_epoch()` function call once per iteration of the main outer loop in each benchmark. This function call indicates how long each iteration takes to complete. Once this function is called by all processes across all nodes running the benchmark, an *epoch count* is incremented. Each job’s latest epoch count is reported whenever the cluster tier requests performance data from the job tier.

Our experiments use a combination of job power-performance curves that are precharacterized or learned at execution time. We precharacterize jobs by fitting them to the model described in Section 3.3.2. Most job types have training R^2 scores of at least 0.97. The exceptions are IS (0.92), MG (0.94), and SP (0.84).

QoS Constraint Selection

We use probabilistic QoS constraints as inputs to the scheduling and power-capping modules. We define a job's QoS degradation limit Q in terms of that job's sojourn time (i.e., elapsed time between job submission and job completion), T_{so} , and the amount of time that job takes to execute when it is not power limited, T_{min} :

$$Q = \frac{T_{so} - T_{min}}{T_{min}}.$$

The QoS objectives should be configured for a data center based on the QoS requirements of its users. We set the QoS objective for all job types in our experiments to be within $Q = 5$ with 90% probability. To justify this experimental design decision, we measured the queue wait time and execution time of jobs from a month of real-world job queue data [66]. The 90th percentile of job wait time divided by execution time is larger than 22, which makes our selected constraint more aggressive than the observed properties of that real-world queue trace.

Job Arrival Rates

We evaluate combinations of co-scheduled jobs across a range of power sensitivity. Some of our experiments (Section 3.3.4) do so through a randomly generated job schedule, with job arrivals generated as a Poisson processes. Job arrival rates are selected so that nodes are expected to be in use for a percentage of time equal to the target utilization. In other words, we relate a target utilization η to job type j 's arrival rate λ_j and non-power-capped time to completion T_j over N nodes by the following equation:

$$\sum_{j=1}^J \lambda_j T_j = \eta N.$$

Power and Performance Measurement

The cluster power manager gets periodic CPU power readings from the GEOPM HPC runtime's `CPU_ENERGY` signal, which aggregates the `PKG_ENERGY_STATUS` MSR across CPUs. Our GEOPM instance uses the *msr-safe* kernel module for MSR access.

We measure application performance in two ways. For our hardware experiments, we report the time that the job spends running its benchmark, as reported by the *Application Totals* section of GEOPM reports that are generated after the job finishes executing. For simulation experiments, we report the QoS metric described in Section 3.3.3.

Test Platform

The goals of our real-cluster experiments are two-fold. First we aim to demonstrate that offline job performance model analyses sufficiently describe real opportunities in cluster power management. Then we evaluate the efficacy of ANOR as a way to mitigate job performance loss when offline analyses do not accurately represent an executing job. We run both sets of evaluations on 16 nodes with dual-package Intel® Xeon® Gold 6152 CPUs and 100 GB RAM. The Thermal Design Power (TDP) is 140 W per CPU.

Adding Performance Variation to a Cluster Demand Response Simulator

Some of our evaluations simulate a 1000-node cluster. We introduce random performance variation to simulations in order to evaluate the impact it has on enforcement of QoS and power-tracking requirements in demand response scenarios. Furthermore, we investigate whether alternative power-capping mechanisms can help mitigate any resulting impacts.

Our simulator's high-level organization is shown in Figure 3-13. The simulator receives configuration files with cluster and job-type properties, and produces a time series of cluster power consumption and a job queue with submission, start, and end time of each job. Input cluster properties include average idle power per node, total node count, average node

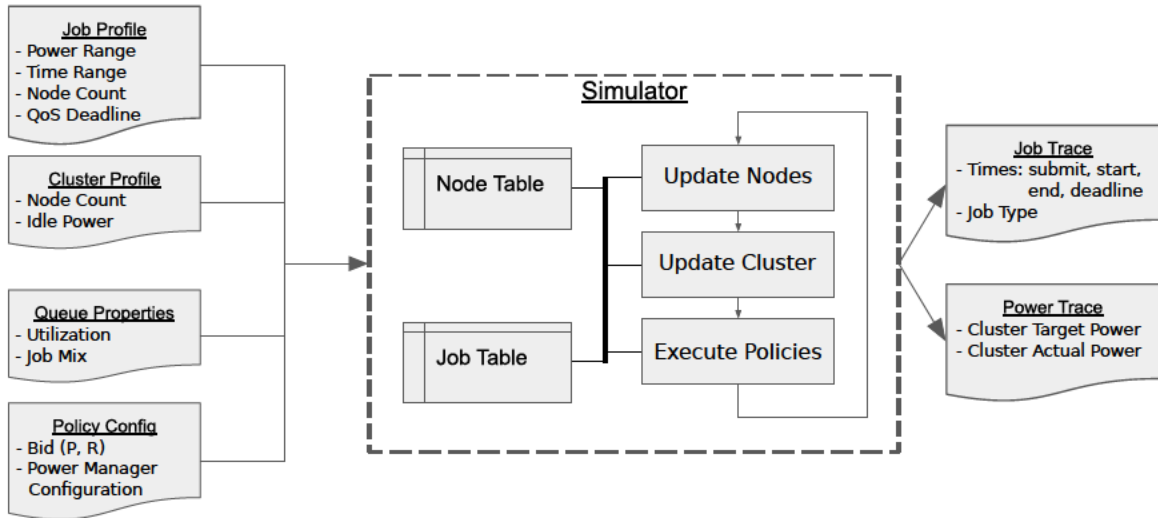


Figure 3-13: High-level organization of our data center demand response simulator.

utilization, and demand response parameters. Properties of each job type include a threshold for the maximum acceptable QoS degradation as defined in Section 3.3.3, required nodes per instance of the job type, maximum achievable power per node running the job, minimum power per node, and the elapsed execution time when the job runs with a cap at either of those power levels. Demand response parameters include average power demand \bar{P} , reserve power R offered by the simulated cluster, and a time-varying regulation signal $y(t)$. The regulation signal ranges from -1 to 1 , indicating the cluster power target $P_{\text{target}} = \bar{P} + Ry(t)$.

The simulator is implemented as a collection of tables that store the current state of nodes and jobs in the cluster, as well as the simulator configuration. The node table indicates whether a given node is idle, or which job it is executing, and tracks the current power consumption and current cap applied to each node. The job table keeps track of timestamps for queue entry, job start, and job end, as well as the type of job. The simulator keeps track of the minimum and maximum power and time configured for each job type, in order to simulate a simple linear power-performance relationship.

The simulator works over 1-second time steps. In each time step, we update the state of the node table, then update the simulated cluster's view of the nodes (i.e., as a job scheduler

and power manager would see them) and external state (e.g., power targets for demand response), then execute a scheduler and a power manager policy. The policy updates inputs to the node table that will be processed in the node-update stage of the next time step. Lastly before starting the next iteration, we append the current state of all tables to a file.

One task in the node table update is to set the estimated percent completion of the current job executing on each non-idle node. The simulator calculates a rate of progress for each time step by linearly scaling between the job type’s fastest and slowest precharacterized rate of progress, based on the server’s current power cap. Some of our experiments introduce performance variation in that rate-of-progress calculation by assigning a random multiplier to each simulated server for the duration of the simulation. We treat a simulated multi-node job as finished executing when all nodes executing the job have reached 100% progress.

3.3.4 Results

In this section, we discuss the impacts of multi-tiered power management on the real cluster’s power-tracking effectiveness, and on the simulated cluster’s demand response bidding and tracking effectiveness.

Opportunities to partially precharacterize cluster job types

The goal of this set of experiments is to identify cases where accuracy of job power-performance models is expected to impact job performance under a cluster power cap. We estimate the slowdown of each job under a given power cap, as modeled in Section 3.3.2. We evaluate the slowdown of jobs under ranges of cluster power caps following the performance-unaware (even power caps) and performance-aware (even slowdown) balancer policies described in Section 3.3.2.

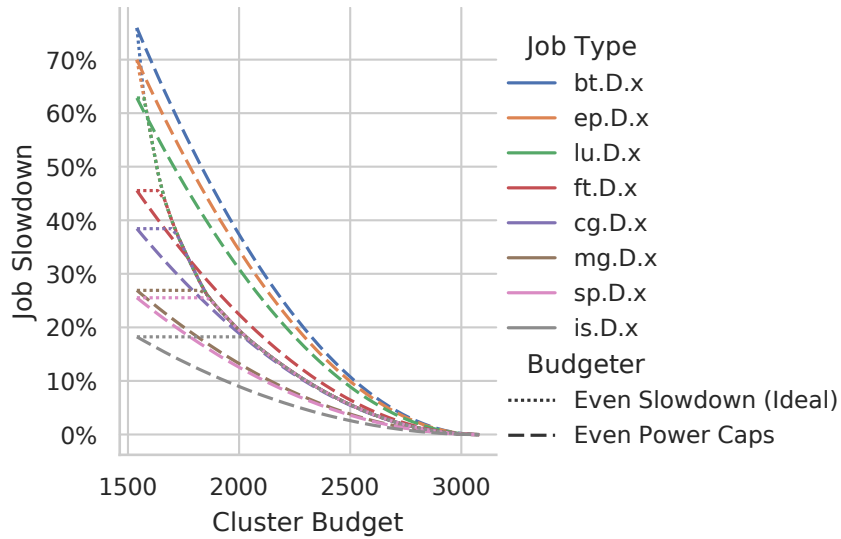


Figure 3-14: Estimated job slowdown when 8 job types are each executing one instance concurrently under a range of shared power budgets.

Impact of integrating precharacterized performance models in a power budgeter

We first estimate the expected slowdown of each job running in a cluster, assuming one of each job type from Section 3.3.3 is executing in the cluster at the same time under a shared cluster power cap.

Figure 3-14 shows the estimated slowdown of each job under a policy that distributes cluster power budgets for even slowdown across jobs and under a policy that distributes budgets for even power caps across nodes. As expected from the discussion in Section 3.3.2, we observe varied slowdown across jobs when applying the even-power budgeting policy, with a range of job slowdown that increases as the cluster budget decreases.

Under the *even-slowdown* policy, all jobs have equal expected slowdown at high cluster power budgets. But as the cluster budget decreases, the jobs with lower power-performance sensitivity level off. This happens because the low power-sensitivity jobs receive less power than high-power-sensitivity jobs in order to achieve equal slowdown, and the low-sensitivity jobs reach the system's minimum-allowed power cap (70 W per CPU package in our evaluation system).

The net impact of the even-slowdown power-budgeting policy is a reduction in the slowdown of the most severely impacted job across currently-scheduled jobs. There is no opportunity for reduced slowdown at minimum or maximum budgets since neither policy has flexibility to assign power caps beyond the range allowed by the power-capping interface. But there is opportunity in the middle region of power budgets.

Impact of misclassifying applications in a performance-aware power budgeter

It may not be practical for a data center to require that all of its users' job types are characterized with power-performance sensitivity models in advance of their execution. Some jobs may be executed before they are characterized, and some jobs may be misclassified as a job type with different power-performance characteristics (e.g., due to user error with user-specified job classes, or due to prediction error in an automated system that has job information that exists in submission-queue metadata). This set of experiments evaluates the performance cost of misclassifying a job's power-performance sensitivity, and explores power management mechanisms that cope with misclassification.

Figure 3-15 shows slowdown in scenarios where jobs (IS, FT, and EP) of different sensitivities are scheduled. In these scenarios, the power-performance curve of FT is unknown to the power budgeter. The left subplots show performance of jobs if the budgeter employs a policy where unknown jobs are assumed to follow the sensitivity curve of the least-sensitive known job type (IS). The right subplots show the opposite policy where the budgeter assumes unknown jobs follow the curve of the most sensitive job type (EP). Upper subplots show cases where the unknown job is smaller (2 nodes) than the known jobs (4 nodes each), while lower subplots show cases where the unknown job is larger (8 nodes) than the known jobs (1 node each). The solid lines show the ideal slowdown when the budgeter knows the true sensitivity of all jobs, while the dashed lines show the slowdown if a performance-agnostic policy is employed, and dotted lines show the slowdown when the budgeter relies on a default sensitivity assumption for the unknown job type.

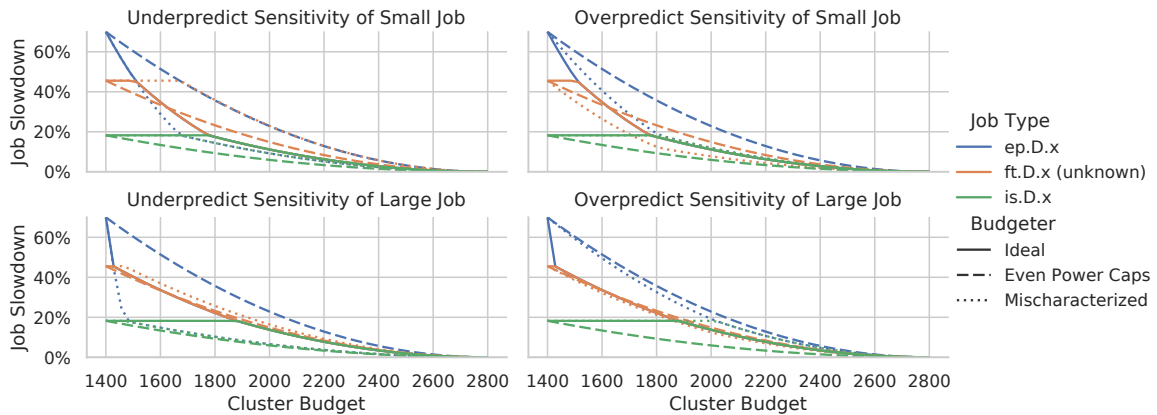


Figure 3-15: Performance impact when a medium-sensitivity job is misclassified as one with higher or lower sensitivity than its true behavior, while co-scheduled with both high-sensitivity and low-sensitivity jobs. Upper subplots model the unknown job as requiring more compute nodes than the known jobs, while lower subplots model the unknown job as requiring fewer compute nodes than the known jobs.

There are two key takeaways from comparing this set of scenarios. First, an *underprediction policy* slows down the unknown job while an *overprediction policy* slows down power-sensitive co-scheduled jobs, as compared to the ideal budgeter. This means that a precharacterization-guided power budgeter must classify unknown job types based on whether it is preferred to degrade performance of the unknown job or other running jobs.

Second, the impact of misclassification depends on the *size of the misclassified job* relative to the size other jobs and the *direction of mischaracterization*. Data centers should account for this to minimize the risk of performance degradation when some jobs are not precharacterized. Small jobs are slowed down when their power sensitivity is underpredicted. Large jobs slow down their co-scheduled jobs when the job's sensitivity is overpredicted.

Regardless of which sensitivity assumption is used for unknown job types, there is a risk of misclassification, which either limits the performance of the unknown job or the other jobs running in the cluster. In case of misclassification, it is important to select a policy for unknown job types that exposes whichever type of cluster performance risk is preferred in the data center, or to have a method to detect misclassification and adjust the power budget.

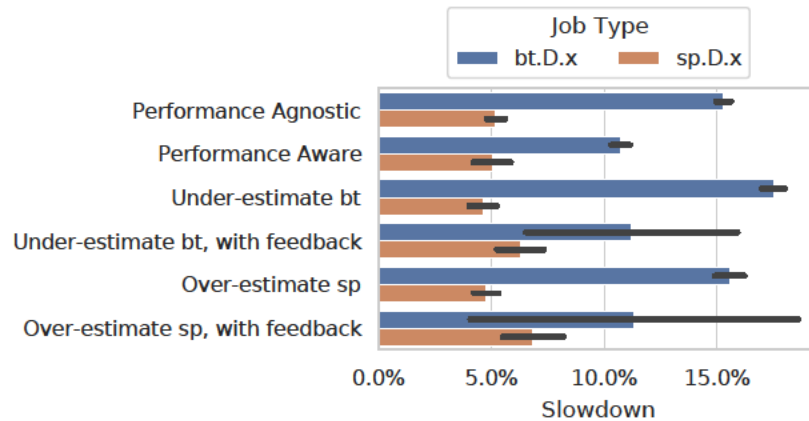


Figure 3-16: Job slowdown when BT and SP are scheduled under a shared power budget with 75% of TDP. Slowdown is shown as a percentage of each job type’s execution time under no power cap. Error bars show standard deviation over 3 runs of SP.

Performance Under Shared Power Caps on Real Hardware

These experiments demonstrate how the offline analysis results translate to a real cluster. We measure the slowdown of jobs in schedules with different mixes of job types. We consider 3 scenarios where there are two jobs with low power sensitivity (both SP), two jobs with high power sensitivity (both BT), or one each of high and low (BT and SP) power sensitivity.

We measure job execution time under a static power budget shared across 4 nodes by a power-balancing policy and by a performance-balancing policy that relies on precharacterized power-performance curves. We use an 840 W budget, mid-way between the maximum and minimum power caps supported by our platform. We consider cases where each job’s type is known and where a job is misclassified as one with different power sensitivity.

Figure 3-16 shows job slowdown under multiple scenarios of model accuracy given a job mix that executes BT and SP in the cluster at the same time. As expected from the offline model analysis, performance of power-sensitive BT degrades from the fully characterized case when that job is misclassified as one with less power sensitivity, or when the other executing job is misclassified as high sensitivity. The budgeter reduces BT’s slowdown in both misclassification cases when it re-trains job models from performance feedback.

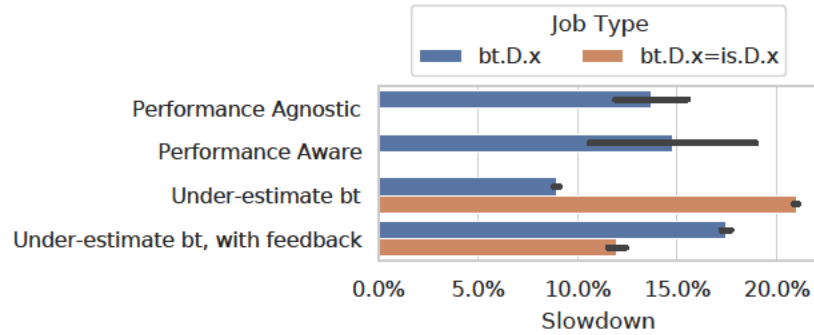


Figure 3-17: Job slowdown when two instances of BT scheduled under a shared power budget with 75% of TDP, with one instance misclassified as IS. Slowdown is shown as a percentage of each job type’s execution time under no power cap. Error bars show standard deviation over 3 back-to-back runs.

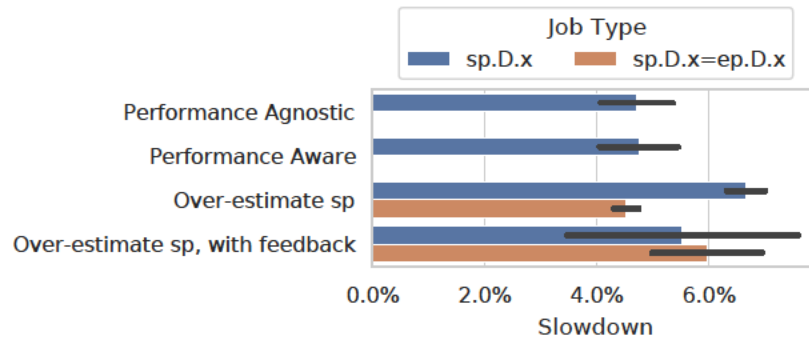


Figure 3-18: Job slowdown when two instances of SP are scheduled under a shared power budget with 75% of TDP, with one instance misclassified as EP. Slowdown is shown as a percentage of each job type’s execution time under no power cap. Error bars show standard deviation over 6 back-to-back runs.

Although the runs with prior performance knowledge do not completely close the slowdown gap between BT and SP, they do reduce the gap in comparison to the performance-agnostic policy. One way this might be addressed is by improving the accuracy of the characterization model. A more robust control system may also help by reacting to modeling error as part of including performance feedback.

Figures 3-17 and 3-18 both show that performance-agnostic solutions perform similar to the precharacterized power budgeter. This happens because all scheduled job types have the same power-performance trade-offs. In Figure 3-17, we see increased slowdown when a high-sensitivity job type is misclassified. By misclassifying one of the low-sensitivity

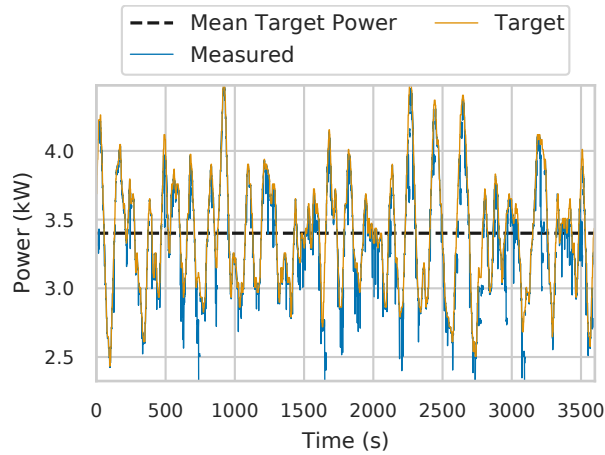


Figure 3-19: Time-varying cluster power targets and measurements using ANOR over an hour of job arrivals from 6 job types. The y axis spans the cluster’s committed range of power flexibility around the requested mean target power.

jobs in Figure 3-18, we see a small slowdown for its co-scheduled job. In both cases, we are able to recover some of the lost performance by communicating information about the unexpected slowdown from the job tier to the cluster tier’s power budgeter.

Performance Under Time-Varying Power Caps

In this section, we evaluate job performance under a time-varying power cap through a 1-hour job schedule. We measure how well our power-capping mechanism follows a moving power target at the cluster level while reacting to unknown job power-performance properties. The power cap changes once every 4 seconds, staying within the range of 2.3 kW to 4.5 kW, as shown in Figure 3-19.

We measure the execution-time impact of different power capping techniques summarized across 6 job types arriving for 95% node utilization as described in Section 3.3.3, using the scheduler described in Section 3.3.2. The slowdown of each job type under each power budgeting policy is shown in Figure 3-20. These results show that the slowdown of jobs with greater power sensitivity (BT, LU, and FT) under a uniform power distribution policy is greater on average than the slowdown of other co-scheduled jobs. The performance-aware

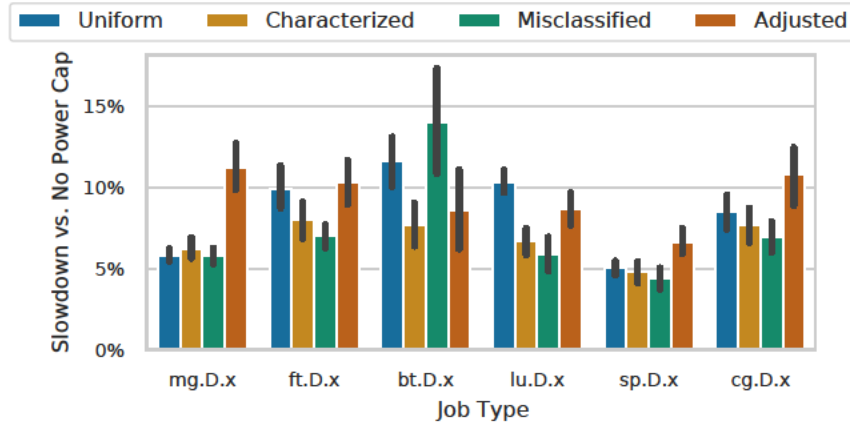


Figure 3-20: Mean execution-time slowdown of job types under a 1-hour schedule with time-varying cluster power caps. Slowdown is shown as a percentage of each job type’s mean execution under no power cap. Error bars indicate the 95% confidence interval. The *uniform* set uniformly distributes the cluster power budget across compute nodes, while *characterized* budgets for uniform slowdown by using precharacterized job properties. In the *misclassified* set, BT is incorrectly classified as a job type that is less power-sensitive than any of the other scheduled job types. The *adjusted* set corrects that mischaracterization by using execution-time performance feedback from the job.

(characterized) balancer improves the worst-job slowdown by steering power toward power-sensitive jobs. As a result, the three most-sensitive job types achieve less slowdown on average, reducing the slowest job type from 11.6% slowdown to 8.0%.

The misclassified runs represent the case where BT (high power sensitivity) is misclassified as IS (low power sensitivity). This slows down BT on average, but performance feedback from the job tier enables the adjusted policy to recover much of the lost performance.

Power is within our constraints (exceeding 30% error no more than 10% of the time) in the worst case (misclassified, without job feedback). All other cases are under 17% error.

Impact of Performance Variation on QoS Degradation

We evaluate our policy under different levels of performance variation to observe the impact of variation on the QoS degradation of jobs. To that end, we generate performance coefficients from a normal distribution with a mean of 1, and adjust the standard deviation

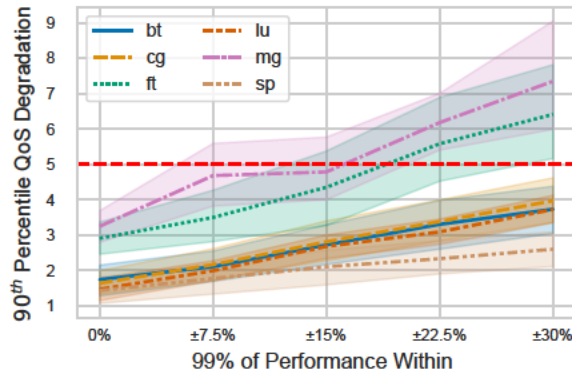


Figure 3-21: QoS degradation under different levels of performance variation. Each line presents the mean degradation over 10 trials. The shaded region indicates the 90% confidence interval. The horizontal dashed line indicates the QoS target of all applications.

to change the level of performance variation. The performance coefficients are randomly generated for each of 1000 compute nodes at the start of each of 10 simulations per variation level. Each simulation uses a different random seed that impacts performance coefficients and job arrival times of 6 job types at 75% utilization. All jobs are scaled to use 25x as many nodes as the job types used in the 16-node cluster experiments. Under each level of performance variation, our method’s power tracking error is within our constraint for less than 30% error 90% of the time.

Figure 3-21 shows the 90th percentile of QoS degradation across performance variation levels. The results indicate that, across all applications, a greater degree of performance variation causes more QoS degradation. However, since different applications have varying levels of sensitivity to performance variation, some of them more rapidly degrade beyond the QoS threshold of 5.

While we are able to avoid capping power on jobs that application feedback indicates are at risk of QoS degradation, we found that this does not improve QoS significantly because the AQA policy already lightly caps power on jobs in this schedule, primarily reducing power by refraining from scheduling jobs to idle nodes.

3.3.5 Discussion

In this work, we evaluate the cluster-power-tracking accuracy and job QoS impacts of a multi-layered power capping method, and we investigate the opportunities and challenges that a multi-layered solution introduces to a demand response power management scenario. We have several takeaways from this investigation, as well as some remaining questions that should be investigated in future work.

Scope of Power Management

We limit this work's investigation to CPU power monitoring and control for simplicity of the initial implementation. The same framework can be used with additional scope through modeling or additional monitoring. For example, the job tier can either model or directly monitor additional devices and components in the nodes. Alternatively, the cluster tier can apply a model of the cluster's total power consumption as a function of the job tier's reported power, as well as environment state and other state within the cluster. A data center facility may also be able to use facility models and metering infrastructure directly for higher-level power monitoring.

Practical Implementation Challenges

We encountered practical challenges implementing and evaluating a performance-aware power-capping system. Specifically, we encountered challenges with asynchronous management of samples and summarized metrics across power management tiers, and challenges transitioning from ideal and static cluster scenarios to end-to-end scheduled scenarios.

For asynchronous sample management, we initially needed to gather many samples from the job runtime to consistently map power caps to job performance metrics. To tackle this issue, we introduced timestamps so that different tiers of the power management system can be mapped to each other when the tiers are not executing their control loops at the same rate.

We initially observed unexpected performance *improvements* in all power management policies when we moved from ideal power-capping scenarios to real-world experiments. Ultimately, those improvements ended up being due to two job types (IS and EP) that have very short execution times (less than half a minute). The time spent setting up and tearing down those short jobs (both in the batch system and in the job itself) represents a major share of the total time those jobs hold compute node resources in the batch submission system. During that time, the compute node's power consumption is low, which enables all policies to reallocate extra slack power to all other active jobs for most of the time the short job is active. We omit those job types from schedules in our final evaluation since they hide the slowdown that would be expected in a system with mostly minutes-or-longer jobs.

3.4 Cluster-Level Policies For Site-Level Power Objectives

Electricity markets are trending toward supply mixes with increasing proportions of renewable energy sources. Most states in the U.S. have adopted renewable portfolio standards to increase their renewable energy generation, and 12 states have put plans into action to achieve 100% clean energy within the next 30 years [75]. The U.S. Energy Information Administration forecasts that 70% of the renewable energy supply will consist of solar and wind power in that time frame [44]. Solar and wind power supplies have time-varying availability, so there is a growing need for power management solutions that adaptively balance supply and demand.

Regulation service programs are one solution to manage imbalances between electricity supply and demand. In such programs, an electricity consumer offers regulation capacity to an independent service operator (ISO) in the grid. The offered service is a promise that the consumer will modulate power consumption to help the ISO balance electricity supply and demand. The consumer receives cost incentives depending on constraints specified by the ISO. For example, the PJM ISO has a program that pays for regulation service based

on changes in hourly rates, on the magnitude of reserve offered by a consumer, and on the consumer’s quick and accurate response to the ISO’s requests for power consumption modulation [72]. In this work, we refer to a *bid* as the amount of average purchased power for an upcoming hour, and the reserve capacity offered for that hour.

Data centers are well-suited to offer regulation service because they consume a lot of electricity and they can quickly modulate their power consumption through job scheduling and server-level power management. An ACM tech brief estimates that data centers consumed 3% of the total energy supply in 2021, and are likely to demand more as power-hungry cryptocurrency and artificial intelligence workloads increase in popularity [46]. Prior works have demonstrated that data centers can effectively manage trade-offs between power consumption and application performance to meet system-wide power objectives [14, 64].

Previous work toward using data centers as reserve capacity in a smart grid has focused on enhancing trade-offs between quality-of-service (QoS) and server power consumption, such as by co-optimization of battery-based and server-workload-based power reserves [64], offering probabilistic guarantees on QoS degradation [97], and enhanced bidding strategies to exploit properties of long-running workload mixes observed in real-world data centers [96].

There is currently a gap in efforts to design QoS-aware demand response policies that operate at the site level (i.e., including all components of the data center). Beyond server power in a data center, cooling and power delivery infrastructure contribute toward significant additional power demand. In a 2021 survey, large data centers used on average over a third of their power consumption for non-server needs [30]. While prior work proposes methods to co-optimize frequency control and cooling system parameters [33], there isn’t a single solution that integrates QoS-aware decisions along the entire path of bidding, job scheduling, and site-scoped power management. Our work aims to close the research gap by evaluating the opportunity to use site-level power consumption in cluster-oriented demand response power management policies.

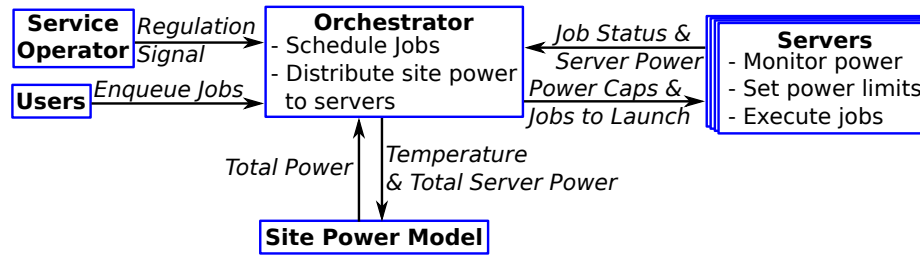


Figure 3-22: The simulator places control logic in an orchestrator that schedules work and applies power limits to servers to satisfy user and independent service operator objectives.

The remaining subsections describe our side-wide demand response simulation infrastructure, explain our experimental methodology, and evaluate our simulation results.

3.4.1 Site-Wide Demand Response

Our high level goal is to identify the cost-saving opportunities of demand response bidding and power management policies in an HPC data center, while meeting QoS constraints of jobs in a work queue. This section describes the data center simulator and the demand response policies we use to analyze our cost-saving opportunities.

Adding a Site Power Model to a Cluster Demand Response Simulator

To evaluate the quality of demand response policies, we simulate a data center's work queues, power consumption across the data center, performance of running applications, and incoming regulation service control signals. As shown in Figure 3-22, simulation logic resides in a central orchestrator, which acts as the data center's job scheduler and resource manager.

The overall execution flow of the simulator begins by loading application and data center properties, then executing the bidding policy, and lastly performing fixed-time-interval simulations of servers, the regulation signal, and the demand response policies for scheduling and power management.

For each one-second time step simulating data center servers, the simulator updates

its queue of incoming job requests from users and reads the regulation signal from the independent service operator. The simulator updates the cluster-level view of server power consumption prior to executing scheduling policies and power-capping policies, which update control signals on the simulated servers.

Job submissions are randomly generated for each simulation, as a Poisson process for each job type. The average arrival rate of each job type is selected to evenly distribute a target data center utilization across all job types in a workload mix.

Policy for Site-Wide Demand Response Through Cluster Power Control

We simulate the *AQA* policy [97] for demand response control decisions, additionally making it aware of site-wide power consumption. We use a site-wide power model to inform the policy how much server power needs to increase or decrease to meet the site-wide power target, and the policy makes its QoS and server-power-aware control decisions in the same manner as described in its original work. This design choice ensures that existing policies can make site-level power control decisions by using a simple model of site-wide power consumption. We refer to the updated site-wide *AQA* policy as *AQA-SW*.

The *AQA* algorithm applies weights to individual job types in a workload mix. Those weights are used to influence which queued jobs get sent to servers, as well as how the power budget is distributed across active servers. Weights and bids are learned prior to evaluating each workload mix by running many simulations of a workload mix through a gradient descent search [97].

In this work, we use the same search used by *AQA*, but we run the *AQA-SW* policy in each gradient descent step. The search incorporates electricity cost, QoS constraints, and power-tracking constraints in the gradient descent cost function. Some search steps occasionally enter regions of violated constraints, then adjust in later steps to find viable bids and weights. We stop searching after 120 iterations of gradient descent, prune non-viable search steps, and select the bid and weights with the lowest cost.

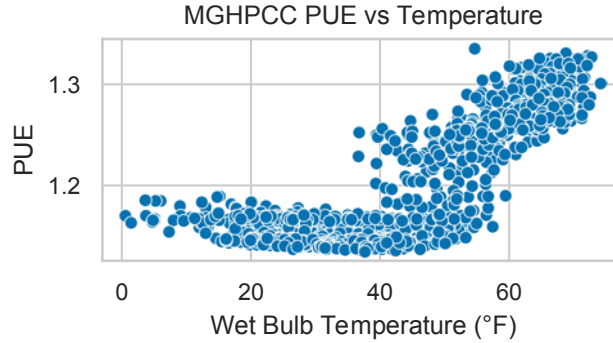


Figure 3-23: PUE of the MGHPCC data center as a function of outdoor wet bulb temperature.

Site Power Model

The simulator estimates power consumption at the site level by using a power usage effectiveness (PUE) model. PUE is defined as $PUE = \frac{\text{Total Power}}{\text{IT Power}}$, so our simulator estimates total power as the product of the modeled PUE and the sum of power consumption across servers.

For this work, we construct a piecewise-linear model of PUE as a function of outdoor wet-bulb temperature, where hotter and more humid days will place a greater cooling load on the data center for a given computational workload. The PUE model is used with the definition of $PUE = \frac{\text{Total Power}}{\text{IT Power}}$ to translate between IT, non-IT, and total power where needed by the simulator. The model of PUE as a function of wet bulb temperature is:

$$PUE(x) = \begin{cases} A(x - B) + C & x < B \\ D(x - B) + C & x \geq B \end{cases}$$

We use least-squares regression to fit the model against power and wet bulb temperature data provided by the Massachusetts Green High Performance Computing Center (MGHPCC) [59]. The MGHPCC is a megawatt-scale computing center with hundreds of thousands of CPU cores and millions of GPU cores. We use data from 2018 to 2020, shown in Figure 3-23. To work with the available data, we approximate PUE as the ratio of the data center’s power consumption to the power used by computing racks in the center’s clusters. The resulting parameters are $A = 5 \times 10^{-12}$, $B = 42.1$, $C = 1.16$, and $D = 5.5 \times 10^{-3}$.

The model predicts the source data’s PUE with 1.1% mean absolute error. We do not focus on developing a robust model in this work. Instead, we evaluate how server-only demand response policies meet their QoS objectives when they are augmented with a simple model to enable site-wide power awareness. We evaluate sensitivity to modeling error in Section 3.4.3.

Power consumption of simulated servers follows a simple model based on activity. If no workload is executing on the server, the server is assumed to consume its idle power. Idle power is assumed to be 169 watts in our experiments as measured in previous experiments on a subset of servers in that cluster [97]. If a workload is executing on the server, then it is assumed to consume the lesser of the server’s current power limit (as set by the orchestrator) and the current workload’s maximum power consumption.

3.4.2 Methodology

Our experiment plan consists of two stages. First, we evaluate the cost savings of a site-wide demand response policy in comparison to the savings from a server-power-only policy. Second, we evaluate the sensitivity of those cost savings with respect to the choice of site-wide power consumption model.

Both the cost-savings evaluation and the sensitivity analysis are performed over a range of simulated external wet bulb temperatures from 40°F to 70°F, in order to exercise data center efficiency levels between the corner cases of observations from the PUE model described in Section 3.4.1.

The properties of our simulated data center are based on sampled data from real servers in the MGHPC. As described in Section 3.4.1, our site-wide power model is fit against site power data. The execution times and power trends of our simulated applications are sampled from application runs on Boston University’s Shared Computing Cluster, which is hosted in the MGHPC.

Table 3.5: Membership of applications within workload mixes. Each application follows a pattern of <name>.<input class>.<process count>. The m_j column shows the number of servers used per job. T_{min} (T_{max}) is the minimum (maximum) processing time in seconds, and p_{max} (p_{min}) is the corresponding power consumption of a server in watts. Q_{thres} is the threshold for each workload mix’s acceptable level of QoS degradation.

App	m_j	T_{min}	p_{max}	T_{max}	p_{min}	Q_{thres}	W1	W8	W9	W12	W13
bt.C.16	1	73	339	86	249	2.5		✓			
mg.D.16	1	84	380	105	266	2.8			✓		
sp.C.16	1	54	375	62	267	7.1			✓		
is.D.32	3	42	249	42	241	5.6	✓	✓		✓	✓
bt.C.36	2	38	343	46	249	3.1		✓			✓
bt.D.49	2	551	391	729	250	5.6	✓		✓	✓	✓
ep.D.64	3	54	353	70	237	3.9			✓		
sp.D.100	4	343	399	381	264	3.3			✓		
ep.D.28	1	124	383	175	238	5.9			✓		
cg.C.4	1	28	238	28	239	4.0		✓			
bt.D.25	1	1022	402	1370	254	3.2			✓		
mg.D.8	1	141	297	151	258	2.9	✓	✓		✓	✓
is.D.4	1	122	204	123	194	7.3	✓			✓	✓
cg.D.16	1	743	326	823	253	7.3		✓			
ep.D.56	2	64	383	90	238	2.0					
cg.D.128	6	231	336	242	246	4.0		✓			
cg.D.32	3	364	281	390	246	5.5		✓			
ep.D.100	4	36	366	49	238	4.5	✓			✓	✓
is.D.64	4	27	287	28	228	3.1	✓			✓	
lu.D.112	4	164	405	222	251	4.1	✓			✓	✓
mg.D.32	2	49	378	58	266	5.0	✓		✓	✓	✓

We select workloads from the workload mixes in the work that introduced the AQA policy [97]. The applications composing each selected workload are shown in Table 3.5. We pick workloads that exhibit mixed properties (W1), as well as corner cases such as low power (W8), high power (W9), low utilization (W12), and high utilization (W13).

Cost Savings

Our goal in the cost-savings experiments is to evaluate the cost opportunities that are offered by using site-wide demand response instead of server-only demand response in an HPC data center. We simulate the AQA policy for server-only demand response and the AQA-SW policy for site-wide demand response (both described in Section 3.4.1).

Cost Model

We model the total cost of electricity as the estimated cost of an energy purchase, minus the value of the regulation service provided by the data center. As with other demand response efforts in this thesis, we model cost as

$$\text{Cost} = \Pi_P \bar{P} - \Pi_R R + \Pi_\epsilon R \epsilon,$$

where $\Pi_P \bar{P}$ indicates the cost of energy purchased for the hour, $\Pi_R R$ represents the value of reserve capacity offered for the hour, and $\Pi_\epsilon R \epsilon$ is the cost penalty for ϵ percent error in tracking the regulation signal. The key difference from other works is that we calculate the cost using *site-wide* power consumption for \bar{P} and R .

Sources of Cost Savings

The site-wide demand response policy accounts for all the site's power in the above cost model. The average purchased power of the AQA and AQA-SW policies is derived from the total power consumed by the site. Note that although the cost equations appear the same across AQA and AQA-SW cases, the actual cost differs because the magnitude of R

increases in the site-wide policy, since it is derived from both server and non-server power in that case.

As a baseline, we estimate non-demand-response cost based on the total energy consumed, times the cost per kilowatt-hour of energy. As a result, the baseline cost does not directly depend on the \bar{P} and R cost components describe above, but the baseline cost does depend on the scheduling and power management decisions made in the data center.

Sensitivity Analysis

We analyze the demand response policy's sensitivity to the choice of site-wide power model. The purpose of this analysis is to identify how well the policies respond when the selected model has prediction errors.

Our simulated *actual* PUE measurements at each temperature come from data provided by the MGHPCC. We group PUE measurements in 5°F bins of wet bulb temperature, selecting the 5th percentile, median, and 95th percentile PUE measurements from each group. We run simulations of the data center where the bidding policies are not aware of the actual PUE. As a result, the \bar{P} and R bids are unable to reflect the resulting increase or decrease to power consumption, so all adjustments to the incorrect site-wide power estimate must be managed through scheduling and power capping. This evaluation emulates the range of power prediction error that may occur if the data center operates at different efficiency shortly after placing a bid (e.g., due to cooling system changes or weather forecast error).

3.4.3 Evaluation

In this section, we evaluate the results from our experiments in cost-saving opportunities under an ideal site-wide power model, as well as our experiments in the effects that site-wide power prediction errors have on a demand response policy.

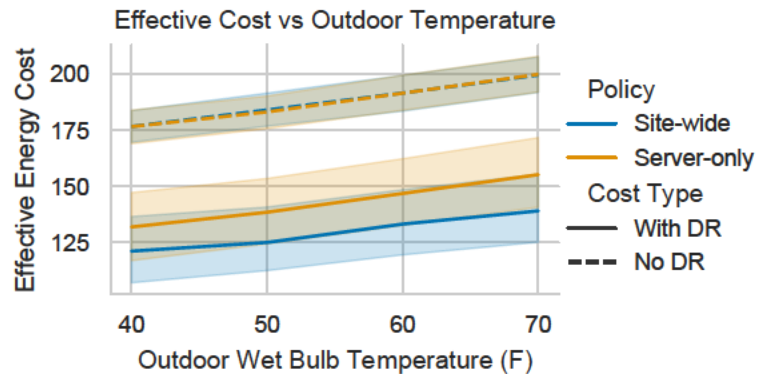


Figure 3-24: Cost of demand response (DR) policies with and without site-wide awareness, compared to the cost of electricity purchase policies that do not participate in demand response programs, averaged across 3 simulations of the workload mixes described in Section 3.4.2. Shaded regions indicate the 95% confidence interval.

Cost Savings

Figure 3-24 shows the simulated costs of demand response policies with and without site-wide power models, compared with the costs of electricity-purchasing policies that do not take advantage of demand response programs. Regardless of the presence of demand response participation, costs trend upward with the simulated wet bulb temperature. That trend exists because more electricity is needed for non-server components, such as site-wide cooling systems, when outside temperature is higher.

The cost-saving opportunities of the site-wide demand response policies consistently outperform the opportunities of the server-only policies. The average improvement is 1.3x savings in the site-wide policies, relative to the savings in the server-only policies.

Sensitivity Analysis

We first evaluate the high-level trends in our cost and performance metrics when there is an error in site-wide power consumption estimation. Next, we look at the properties of the worst-performing scenario underneath those trends.

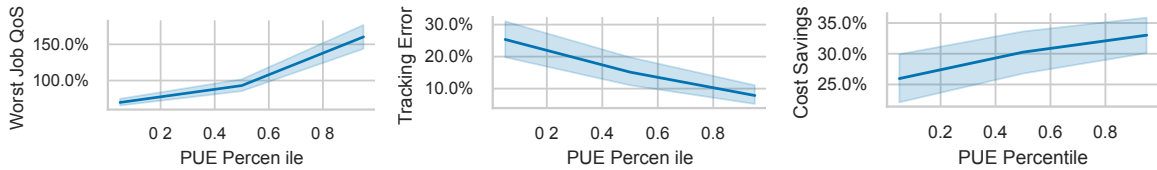


Figure 3-25: Sensitivity of worst-case QoS, regulation signal tracking error, and cost savings as a response to corner cases of PUE modeling error. Shaded regions indicate the 95% confidence region across 3 trials of simulation over 4 temperature points and 5 workload mixes. *Cost savings* are relative to the cost of electricity without demand response. *Worst QoS* is the worst QoS degradation (relative to target QoS) across job types. *Mean tracking error* is the mean absolute error of the data center’s actual power, with respect to the time-changing power target specified by the independent service operator.

High-Level Trends

Figure 3-25 shows the sensitivity of demand response cost and performance as the AQA policy over-predicts or under-predicts the site-wide power consumption. Low PUE percentiles indicate scenarios where the PUE is less than typical PUE values used in the site-wide power model, which cause the bidding policy to bid higher than its needed power. Similarly, the higher PUE percentiles indicate scenarios where the policy is likely to under-bid for power.

As the actual power consumption trends from over-bidding to under-bidding scenarios, the degradation of the worst-performing job increases in mean and in variance. Both trends occur because it is easier to let jobs run at full speed when there is a larger power budget.

Although over-bidding scenarios meet QoS more easily, higher power requests may not be achievable if the data center is not running high-power workloads. Because over-bidding cases are more difficult to achieve all requested power targets, we see that lower PUE percentiles cause greater tracking error in the demand response power management policy.

These experiments cover the case where \bar{P} and R selection are based on incorrect estimates of the site’s power properties, so tracking error is the only remaining cost component described in Section 3.4.2. This means that although cost savings increase with the site’s actual PUE, that is simply because the tracking error decreases in that case, and it comes with the previously-discussed impact on QoS degradation.

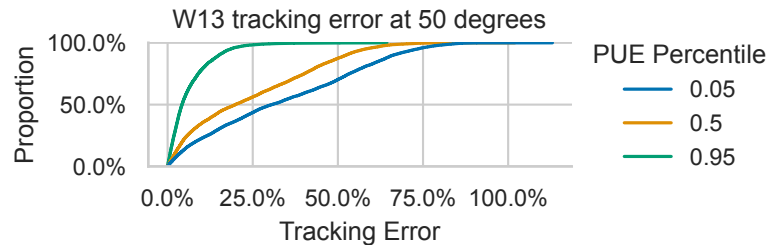


Figure 3-26: Cumulative distribution function of target-power-tracking error as a percentage of the R bid, for the W13 workload mix at 50°F wet-bulb temperature.

Worst-Performing Scenario

The worst-performing scenarios occur when the external wet-bulb temperature is near 50°F, where there is a transition between the data center’s free cooling mode of operation and its active cooling mode of operation, as shown in Figure 3-23. The model indicates the PUE is 1.2 at 50°F, but the 5th, 50th, and 95th percentiles of PUE at 50°F are 1.15, 1.17, and 1.26, respectively. We evaluate behavior of the demand response policies on mix W13 at 50°F.

Workload mix W13 has higher system utilization than the other mixes. This gives the data center less room to modulate its power level without causing work to accumulate in the job queue. Figure 3-26 shows the power-tracking error of the data center when applying the site-wide power management policy in different PUE prediction error scenarios. This case results in better tracking error when the site-wide power is under-estimated, and struggles to track the target power near the median case. This suggests that the workload mix could benefit from standby jobs and preemptible jobs as described in the original AQA work [97].

In contrast to the more spread out tracking error distribution in the over-bidding case, the QoS degradation plots in Figure 3-27 show more spread-out distributions in the under-bidding case. Furthermore, not all applications are impacted equally, as some have more drastic changes in QoS degradation across the PUE percentile cases.

The `lu.D.112` and `mg.D.32` applications are the most sensitive to under-predicting the site power demand. They transition from far under threshold degradation in the 0.05 PUE percentile case to above threshold for 50% of runs in the 0.95 PUE percentile case. Both of

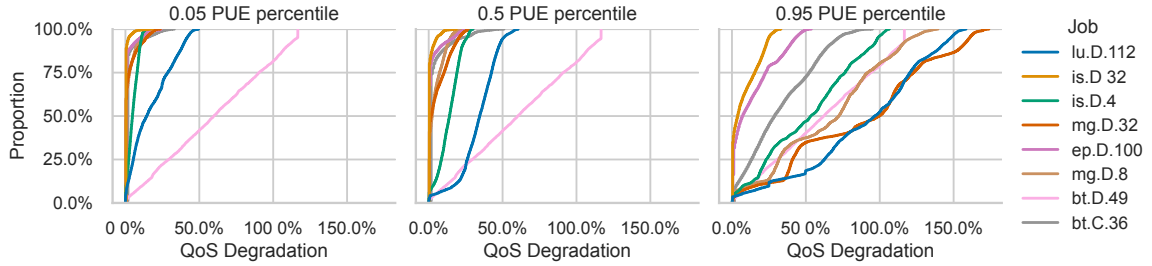


Figure 3-27: Cumulative distribution functions of QoS degradation as a percentage of each application’s QoS threshold, for the W13 workload mix at 50°F wet-bulb temperature.

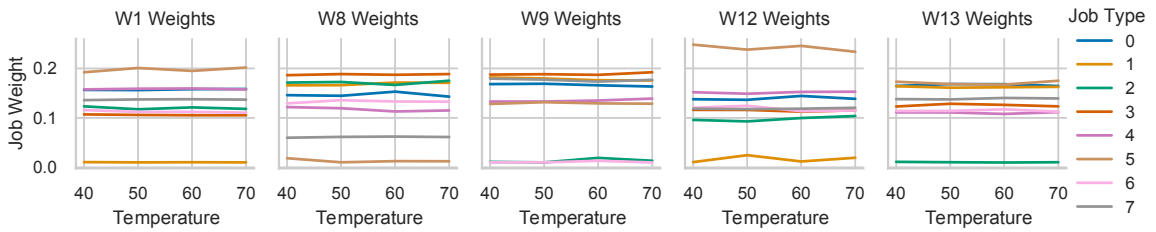


Figure 3-28: Job weights selected by training the AQA demand response policy against each workload mix. Job types refer to the *App* column in Table 3.5. Job indices are presented here since they do not map to the same job names across workload mixes.

those applications stay well under their QoS thresholds in all temperatures evaluated in the experiments with ideal site-wide power prediction (Section 3.4.2 experiments).

Application Weights

Job weights used by the AQA policy are largely insensitive to the data center’s site-level efficiency, as indicated by nearly flat lines for all workload mixes in Figure 3-28, suggesting that it is sufficient to fit the weights against a single simulated relationship between server power and site-wide power consumption for each workload mix. The learned weight could be used as an initial state in future searches for \bar{P} and R across other PUE scenarios.

3.5 Concluding Remarks in Multi-Level Power Management

This chapter demonstrates that multi-level power management solutions more effectively achieve their objectives than siloed solutions that only operate in a single level. Our

opportunity analysis in Section 3.2 demonstrates that cluster power management policies make more efficient decisions when unified with prior knowledge about job power and performance properties. We highlight the need for online feedback between cluster and job tiers in Section 3.3, where our proposed ANOR framework enables multi-level power management without complete prior knowledge of power and performance properties. In Section 3.4, we demonstrate improved cost savings in demand response policies by integrating a simple site power model into cluster power management mechanisms. Through these works, we show improved efficiency, performance, or cost by introducing collaboration between a cluster power manager and a neighboring power management layer.

Chapter 4

Performance-Aware Job-Level Power Management

HPC job performance challenges may surface in many ways. For example, job performance properties may change from one run to the next, or they may not be uniform throughout the job's execution. By including performance awareness in a job-level power manager, we can react to job-specific performance challenges when they surface.

Chapter 3 designs methods for multi-level power management as a way to loosely couple cluster power management policies (such as those used in demand response) with job-level power management policies. Our proposal for the ANOR framework opens the door to provide performance feedback from job-level power managers to a cluster power manager. This chapter focuses on job-level power management that incorporates job performance awareness with knowledge of trade-offs in hardware power management controls.

This chapter begins with a description of one challenge where power properties are different across processes within a job, *imbalance* in bulk-synchronous parallel applications (Section 4.1). We then outline related work in performance-aware and hardware-aware power management (Section 4.2). Lastly, we introduce a method to configure a new CPU power management interface in response to measured application imbalance (Section 4.3).

4.1 Imbalance in Bulk-Synchronous Parallel Processing

Many bulk-synchronous parallel applications run in HPC systems every day, including solvers and simulators that iteratively work over large problems. Such applications contain

synchronization points, where multiple processes need to achieve some program objective before the general program flow can proceed. Many processes independently solve local problems and periodically communicate with each other at synchronization points to work toward a global solution. If a single process encounters significant delays to reach a synchronization point, the performance and efficiency of the entire program will suffer. Such mismatches in time to reach those synchronization points are called *imbalance*.

Imbalance may result from both hardware and software causes, such as hardware variation [56], and application inputs that cannot be perfectly distributed among processes [84]. Imbalance introduced in one way does not necessarily need to be mitigated by removing the source of imbalance. For example, an application may adjust for imbalance in its inputs by partitioning those inputs with consideration for the expected time to finish working on each partition of the problem. It is also possible to mitigate the effects of imbalanced application inputs by slowing down the processes that have less assigned work.

4.2 Related Work in Performance-Aware Power Management

This section discusses related works in hardware and application efficiency tuning, including both preventative and reactive mechanisms to efficiently use computing resources.

4.2.1 Hardware-Level Efficiency Tuning

Hardware-level efficiency tuning techniques use hardware power management controls to improve the efficiency of executing workloads. Some related approaches use P-State controls to throttle processing units when the current workload does not efficiently benefit from higher frequency states. Other related works extend their range of frequency control to influence a hardware power manager's turbo-boosting decisions.

P-State Control

Countdown [12] throttles frequency when in a long-lasting network region. This and some of its related works use non-MPI time as a metric of interest when making decisions, as we do here. Later work [13] separately models slack time, networking wait time, and networking copy time. The model is used to balance the non-networking time and to select efficient configurations for networking code through frequency control.

Cuttlefish [48] searches core and uncore frequency limits by region. Regions are identified by signatures from measured TOR inserts per instruction. As a result, it searches for ideal frequencies per level of memory-intensiveness in an application.

EAR [22] identifies periodic regions by patterns in observed MPI function calls. EAR searches the control space of core and uncore frequency across periods to select efficient frequency configurations.

Turbo Boosting

Recent work demonstrates how SST-BF can be used to give high base frequencies to cores running higher-priority workloads [86]. SST-BF operates under a similar principle of exchanging lower frequency on many cores for higher frequency on a few cores. The difference is that SST-BF exchanges *base* frequency on *platform-defined* sets of CPU cores, whereas SST-TF exchanges *turbo* frequency ranges on *user-defined* sets of CPU cores. These differences impact the kinds of problems where each solution is directly applicable. Our proposed solution targets bulk-synchronous HPC workloads, which may exploit bursts of high frequency sensitivity on individual cores to correct for imbalances across those cores within bulk-synchronous iterations.

The Poseidon algorithm [58] increases turbo frequencies of OpenMP workloads by packing work into a smaller number of cores. Some cores go idle and enter deeper C-States as a result of the thread packing, which allows the active cores to enter higher P-States in

the turbo range. Our work speeds up workloads by correcting for imbalances across MPI processes, whereas Poseidon targets workloads with varying degrees of thread parallelism. Many HPC systems disable C-States, and some imbalances may not be drastic enough to ensure that frequent thread reaffinitization will have more performance benefits than costs. Our proposed solution targets such scenarios by monitoring, but not controlling, how much work is performed on each core, and allowing higher turbo frequencies on the cores that lag behind in progress.

Wamhoff et al. investigate strategies to improve efficiency in multi-threaded workloads by using DVFS to slow down threads that are waiting for locks while speeding up threads that hold locks [88]. In cases where baseline behavior was power-bound, these policies are able to increase the extent to which turbo frequencies can be used by the lock holders, resulting in performance improvements. Our work similarly aims to increase turbo utilization by applying frequency limits on cores that are not on the critical path. We also aim to increase the peak achievable turbo frequency on the critical path by lowering the range of turbo frequencies allowed on non-performance-critical cores. Our work applies frequency controls to balance MPI applications before bulk-synchronous loop iterations complete, instead of only applying the frequency trade-offs while a worker is in a non-progressing state.

4.2.2 Application-Level Efficiency Tuning

Applications can balance themselves by assigning sub-problem to processors based on estimates of each problem's amount of work. For example, the NPB benchmarks evaluated in these experiments statically balance their work by decomposing the problem into many smaller sub-problems of varying size. The sub-problems are assigned weights based on the amount of work they contain, then the sub-problems are distributed to processes based on those weights.

The LAMMPS application distributes sections of its simulation space across MPI processes by recursive coordinate bisection [84]. Their balancer can also be invoked

dynamically while a LAMMPS simulation is executing, by specifying how often to check for imbalance, and at what threshold of imbalance to pause simulation to execute recursive coordinate bisection over the current state of a simulation.

4.3 Guiding a CPU Power Manager to Rebalance Parallel Applications

In this section, we propose a method to guide a CPU's power management decisions for better HPC application performance, and to improve energy efficiency of servers as a result. We evaluate new opportunities to work with Turbo Boost technology as a tool to improve performance on imbalanced, compute-bound applications.

Turbo Boosting enables a CPU to exploit bursts of high compute needs by temporarily increasing the core frequency above the CPU's base frequency as long as power, current, and temperature design constraints are met [62]. Higher-frequency bursts are possible when more cores are in sleep states (i.e., not in the ACPI C0 state). Consequently, a CPU cannot reach as high of a turbo frequency when all cores are active (*all-core* turbo frequency limit), as compared to when only one core is active (*single-core* turbo frequency limit).

The Ice Lake family of Intel CPUs support a new set of features that allow increased turbo frequencies even when all cores are active [65]. The *Intel(R) Speed Select Technology Core Power* (SST-CP) feature lets a user specify a priority level for each core in a CPU package. The CPU's power management unit uses those priorities to distribute power among the cores when the CPU is power-constrained.

The SST-CP feature can be combined with the *Intel(R) Speed Select Technology Turbo Frequency* (SST-TF) feature, which enables different peak turbo frequencies based on the core priority configuration, as illustrated in Figure 4.1. Low-priority cores get a decreased peak turbo frequency in exchange for allowing increased peak turbo frequency on high priority cores. When more cores are configured with low priority, the remaining high-priority cores are afforded greater turbo frequency limits. As a result, some cores on a CPU package

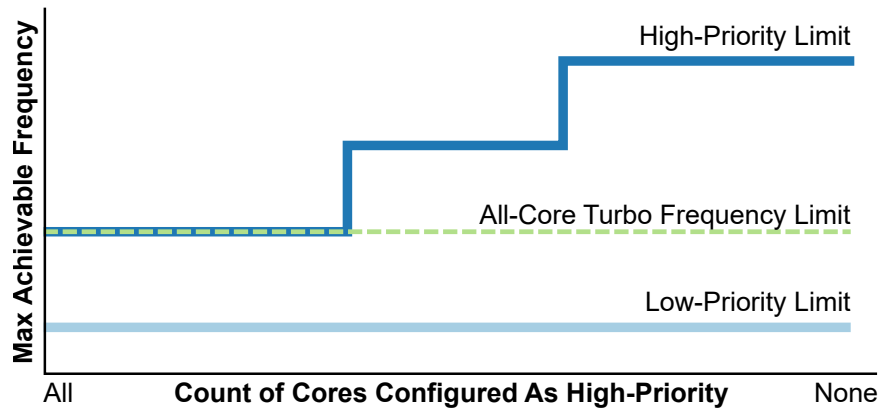


Figure 4-1: SST-TF CPU core frequency trade-offs. SST-TF allows a user to exchange lower all-core turbo frequency limits on some cores for increased limits on other cores. The actual frequency limit trade-offs vary by part and can be read from the CPU.

can be configured to achieve greater than the processor’s regular all-core turbo frequency limit, even when all cores are active.

We propose a technique for a job runtime to configure these turbo frequency features to improve performance in *imbalanced bulk-synchronous parallel* jobs. Workload imbalance is understood to be a source of efficiency loss in HPC applications. Middleware solutions may decrease the energy consumption of imbalanced workloads by matching the allocated computing resources (such as power or maximum frequency) to the needs of a running application [29, 13]. The high-level objective of prior solutions is to minimize energy consumption with minimal performance degradation. However, these new turbo frequency prioritization features enable us to convert some of those energy savings back into performance *improvements* by selectively speeding up the application’s critical path.

Although thermal and electrical constraints prevent all CPU cores from concurrently operating at peak turbo frequency, imbalanced applications do not require that all cores concurrently operate at that peak frequency. For example, a subset of cores far from the application’s critical path can be aggressively throttled to allow the performance-critical cores to more closely reach the processor’s maximum turbo frequency limit. The system

under evaluation in this section allows all cores to concurrently achieve up to 2.7 GHz, or can alternatively allow 8 cores to reach 3.3 GHz as long as the rest of the cores are restricted below 2.1 GHz.

Our goal in this section is to rebalance compute-bound parts of HPC applications by speeding up the lagging processes, rather than solely slowing down the leading processes. We demonstrate that the new SST-TF feature can help achieve this goal by providing a power management policy that combines awareness of a job’s time spent in parallel processes with knowledge of the hardware performance trade-offs that are exposed through SST-TF and SST-CP. The combined knowledge guides configuration of core priorities in SST-CP.

We start Section 4.3.1 with a description of our method to configure P-States and SST-TF in response to imbalance. Next, Section 4.3.2 describes our experiment platform and the applications we evaluate under our proposed policy. Section 4.3.3 discusses the outcomes of using our power management method with the experimental applications.

4.3.1 Configuring CPU Core Priority in Response to Imbalance

Our proposed balancing technique explores how turbo frequency limits can be exchanged across CPU cores. This solution balances the time spent outside of MPI function calls in a parallel application implemented with MPI synchronization, referred to as *non-networking time* in Figure 4.2. We define *leader* processes as the processes that reach the end of an epoch before the other processes. *Lagger* processes are those that take longer to reach the synchronization point (corresponding to the application’s critical path). We aim to balance the time in compute phases by guiding turbo frequency limits to speed up the lagger process while slowing down leaders, thus reducing the time to reach the end of each iteration.

Non-networking time is assumed to be compute-sensitive time. In reality, it may be composed of many regions of code with varying degrees of compute intensity. This assumption causes our method to behave conservatively with respect to performance impacts that result from frequency throttling decisions.

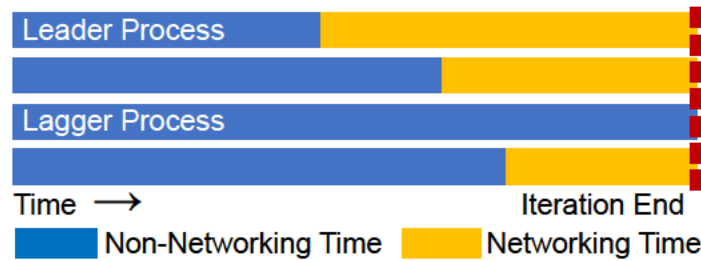


Figure 4-2: Leading and lagging processes in an iteration of a bulk-synchronous loop. Leader processes spend less time per iteration in their compute phases of work. Lagger processes spend the most time in their compute phases of work.

Application performance is monitored by inserting a single function call within the application’s main outer loop of computations. Each time this function is called, the performance monitor increments a counter and immediately returns to continue executing the application. When the counter has incremented across all processes in the application, the *epoch* count increments. In other words, increases in epoch count typically correspond to iterations of the application’s outer main loop. Our objective is to reduce the time between epochs by enabling higher turbo frequencies on the CPU cores that spend the most non-networking time per epoch.

We initialize power and frequency controls to a state that is expected to result in baseline performance for the application. Specifically, all cores are set to high priority in SST-CP, and are given frequency limits equal to the processor’s maximum allowed frequency.

After initialization, the control loop samples program state once every 5 ms. Whenever a new epoch is detected, we apply frequency controls based on the predicted critical path and the frequency-performance trade-offs of the completed epoch. Frequency controls are also adjusted when unexpected application state is detected, which may be due to phase changes or rapidly-changing imbalance. The overview of data flow across these components is shown in Figure 4.3. Each component is described in the following subsections.

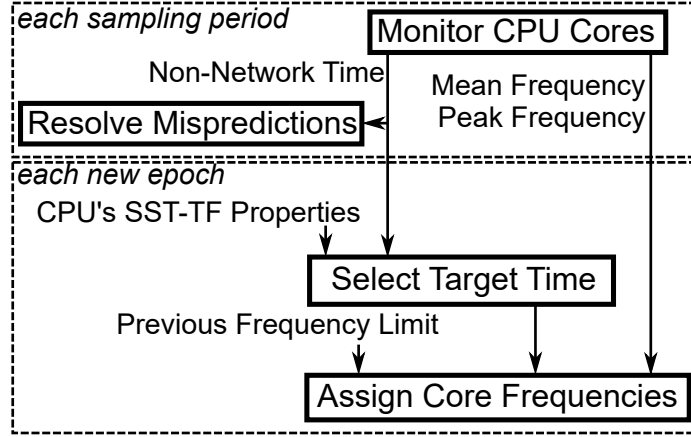


Figure 4-3: Data flow for performance-aware SST-TF configuration. The data flow consists of monitoring and misprediction resolution in each sampling period, as well as predictive rebalancing whenever a new epoch is detected.

Monitor CPU Cores

Every sampling period, we update a running counter of time spent in non-networking regions of the application. We use the PMPI profiling interface for MPI applications to set an indicator while the application is in MPI function calls. If that indicator is not set for a given CPU core when a sample is taken, then the time since the previous sample is added to the non-networking time accumulator for that core.

We monitor `IA32_APERF` and `IA32_MPERF` model-specific registers (MSRs) to track peak and mean achieved frequencies. Average achieved frequency between two points in time is calculated as:

$$f_{\text{achieved}} = f_{\text{base}} * \frac{APERF_t - APERF_{t-1}}{MPERF_t - MPERF_{t-1}}$$

We approximate each epoch's peak achieved frequency as the maximum observed frequency between any two sampling periods. The mean achieved frequency is calculated from the first sample of the monitored epoch to the first sample of the next epoch.

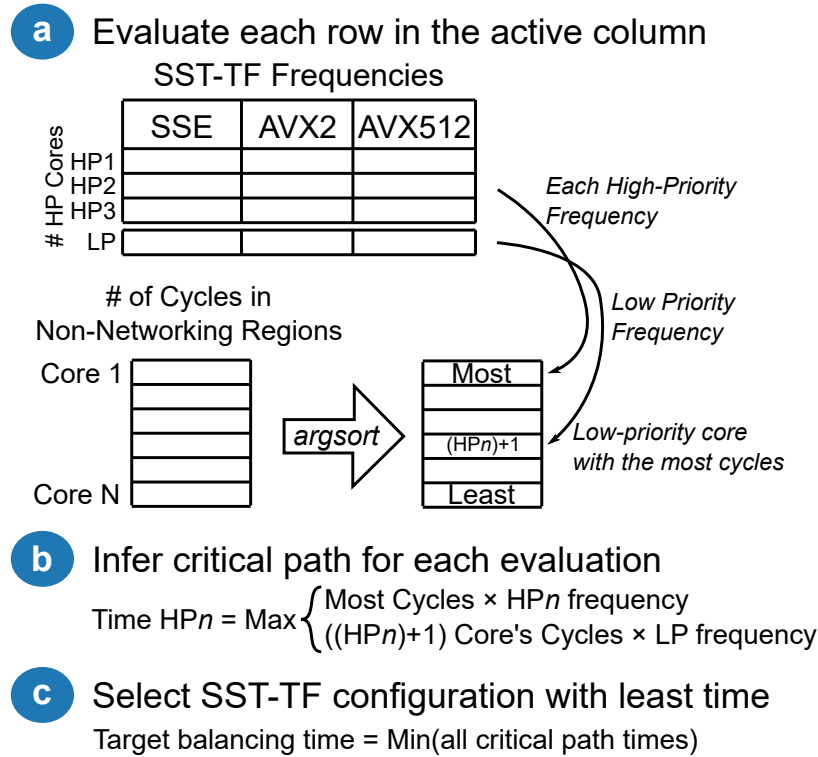


Figure 4-4: Target time selection searches for the SST-TF configuration that is estimated to minimize the time spent in an application’s critical path.

Select Target Time

Whenever a new epoch is observed, we select a new target non-networking time for upcoming epochs. If we select too long of a target time, then we slow down the application’s critical path. On the other hand, if we select too short of a target time, then we will not be able to throttle cores significantly enough to enable higher turbo frequencies on the critical path. Figure 4-4 shows the steps to select a target time for rebalancing via CPU frequency controls.

The performance risks of poor target time selection are asymmetric, since over-targeting harms performance but under-targeting merely misses opportunities and achieves baseline performance. Poor decisions may be compounded by basing future balancing decisions on previous incorrect decisions, where each subsequent epoch may balance to a slightly longer target time than the previous epoch.

We mitigate the asymmetric risk in target time by ensuring that frequency controls are never configured to throttle *all* cores in a CPU package. Then we balance the non-network time of all cores to match the *expected* non-network time of a reference core that was not frequency limited in the previous epoch. Expected non-network time is determined from the expected frequency of the reference core, which depends on the active turbo license level (i.e., the SSE, AVX2, and AVX512 frequency limits reported by the SST-TF interface), as well as how many other cores are also configured as high priority in SST-CP. Since this covers multiple possible combinations of SST-TF configuration, we apply a search for a candidate SST-TF configuration that will minimize an estimate of the critical path’s execution time.

The SST-TF configuration search begins with a target time equal to the non-network time of the selected reference core. Then, we iterate over each of the high-priority core counts that result in a different maximum achievable frequency (i.e., each row in part **(a)** of Figure 4.4). We estimate the critical path time at each stepping point (part **(b)**), and select the SST-TF configuration that results in the least estimated non-networking time (part **(c)**). Each critical path time is approximated as the greater of times spent in the expected laggiest high-priority core and the laggiest low priority core. We apply a simplifying assumption that the laggiest cores are those with the most non-networking compute cycles, and that the inverse of their non-networking time scales linearly with frequency, as defined below.

$$T_{\text{estimated,core}} = T_{\text{measured}} * \frac{f_{\text{measured,core}}}{f_{\text{expected,core}}}$$

For an example of a critical path estimate, suppose we are evaluating a candidate configuration with 8 cores set to high priority. In that case, we estimate that the core with the most CPU cycles spent in non-networking regions will execute with the high-priority SST-TF frequency that is allowed for a high-priority core count of 8. We assume that the core with the 9th-most CPU cycles in non-networking regions will execute at the SST-TF

low-priority frequency. The estimated time of each core is calculated as in the equation above, and the greater of those two times is assumed to be the time of the critical path.

Estimating Achievable Frequency

The time estimates described in Section 4.3.1 assume that we know the maximum frequency that each core can achieve. The achievable frequency of each core depends both on how many cores are configured as high priority, and on which turbo license level (i.e., SSE, AVX2, AVX512) is currently active. The high priority core count is known because we configure the core priorities in each iteration of the rebalancing method. For the current high-priority core count, we look up the maximum achievable frequencies of each license level, as reported in the SST-TF programming interface. We assume that our workload's license level is the nearest one greater than the core's recently-achieved frequency.

Assign Core Frequencies

After selecting a target non-networking time to balance against, we compute a set of per-core frequencies that are expected to make each core spend the desired amount of non-networking time in the coming epoch. We scale the previous epoch's average frequency by the ratio of *measured* non-networking time to *desired* non-networking time, biased higher by the length of the sampling period with respect to the desired time. The added bias exists to reduce the risk of performance loss due to the impacts of noise over a small count of samples. The desired frequency is calculated as:

$$f_{\text{desired,core}} = f_{\text{measured,core}} * \frac{T_{\text{non_network,core}}}{T_{\text{desired}}} * \left(1 + \frac{T_{\text{sample_period}}}{T_{\text{desired}}} \right)$$

Target time selection in Section 4.3.1 assumes that there is always at least one core that we do not throttle. We ensure that invariant by scaling the highest-limited core frequency all the way to the processor's maximum frequency.

The resulting configuration assigns the processor's maximum frequency to the core with the most non-networking cycles measured in the previous epoch. The remaining cores are assigned lower frequencies based on how much less time they spent in non-networking regions of code.

The desired frequency is controlled by either P-States, or SST-TF, or both in combination. When P-States are in use, the desired frequency settings that land between P-States are rounded up to the next P-State, then applied via the `IA32_PERF_CTL` MSR. When SST-TF is in use, the desired frequencies are used to guide the selection of core priorities for use in the SST-CP interface.

We select a high-priority SST-CP class of service for a core if the desired frequency exceeds the expected low priority frequency (using the same license level inference method as described in Section 4.3.1). Otherwise, a low priority class of service is selected. In the SST-CP interface, priorities range from 0 to 3, with 0 as the highest priority. We use priorities 0 and 3 for high and low priority, respectively. The interface allows us to specify ranges of desired frequencies for each priority level. We conservatively set the low-priority level's limit to the CPU's base frequency, though it will often actually be restricted to lower frequencies depending on the workload-dependent low-priority frequency enforced by SST-TF. We set the high-priority level to allow the entire turbo range of frequencies.

Resolve Mispredictions

Mispredictions of target time and target frequencies may occur for multiple reasons, such as inaccurate approximations of the frequency-performance relationship, sampling noise, and rapidly changing application imbalance. In addition to applying prediction-based frequency controls in each observed epoch, we also react to prediction errors during each sampling period. Specifically, we react to cases where the high-frequency cores were given more resources than necessary

We aim to balance time spent in a region of interest. In this version of our method, we apply a simplifying assumption that each epoch consists of a networking region and a non-networking region, using the latter as our region of interest. In reality, a single epoch may contain many instances of entering and exiting networking functions. While the simplifying assumption does allow us to balance the aggregate time spent in the region of interest, there may be missed opportunities if significant time is spent in the intermediate networking functions within an epoch.

We mitigate the above scenario by deprioritizing cores that are in a networking region of code for multiple samples in a row. We only apply this heuristic when *multiple* such samples are observed, based on the performance evaluations by Cesarini et al. [13].

Furthermore, if we observe that all the frequency-unlimited cores are currently in networking regions of code (again, for multiple samples), this may either indicate that we are within an intermediate networking region, or that there was a misprediction resulting in over-restricting the frequency of our low-priority cores. In that case, we remove the frequency restrictions applied to the cores that are still in non-networking regions of code.

4.3.2 Environment and Tools For Imbalance Experiments

This section outlines our computing environment, tooling, and applications used in our experiments. A key requirement of our computing environment is that it supports SST-TF. We select tooling that enables simple monitoring and control of hardware state while measuring application performance. We evaluate MPI applications with a range of imbalance and frequency-performance trade-offs.

Computing Environment

Our experiments require Intel Ice Lake or later CPUs that support the *Intel(R) Speed Select Technology - Turbo Frequency* feature, in order to utilize SST-CP and SST-TF. The server properties used in our experiments are outlined in Table 4.1.

Table 4.1: System Properties

Operating System	CentOS Linux 7, Kernel 5.10.0
CPU Model	Intel Xeon Gold 6338T
CPU Packages Per Node	2
Cores Per CPU Package	24
Thermal Design Power	165 W
Min Frequency Limit	0.8 GHz
Base Frequency	2.1 GHz
Max All-Core Turbo Frequency	2.7 GHz
Max Single-Core Frequency	3.4 GHz

Table 4.2: System Package SST-TF Frequency Limits

High-Priority Core Count	Max Frequency (GHz)		
	SSE	AVX2	AVX512
up to 8	3.3	3.3	3.2
up to 12	3.0	3.0	2.9
up to 16	2.8	2.8	2.7
<i>(Low Priority)</i>	2.1	1.8	1.5

The processor makes its SST-TF frequency limits available to the user through a Linux driver [65]. Table 4.2 illustrates the maximum all-core turbo frequency under different workload types. The frequency limits of low-priority cores are constant across different counts of high-priority cores.

Measurement Tools

Both measurement and control of the test servers are implemented in the Global Extensible Power Manager (GEOPM) job runtime [29]. GEOPM includes *IO groups*, which define a software interface to sample and control system state, and *agents*, which implement power and performance management algorithms. As part of this work, we introduce a new IO group⁸ to interact with SST-CP and SST-TF from GEOPM, and we add a new agent⁹ to balance imbalanced MPI applications using SST-TF.

⁸Available as part of the main GEOPM project at <https://github.com/geopm/geopm>

⁹Available at <https://github.com/dannosliwcd/geopm/tree/single-node-rebalancer>, with user documentation at [./service/docs/source/geopm_agent_frequency_balancer.7.rst](https://github.com/dannosliwcd/geopm/blob/master/service/docs/source/geopm_agent_frequency_balancer.7.rst)

We use the existing *monitor* agent to perform baseline measurements of energy and performance. This agent reads the requested metrics and does not write any changes to system power and performance controls. As part of this work, we introduce a new *progress_balancer* agent to implement the algorithm described in Section 4.3.1. When evaluating applications, we reserve one core to execute the monitoring infrastructure separate from the applications.

Applications

We evaluate how the power management policies respond to imbalanced benchmarks from the NAS Parallel Benchmarks suite [60], the Mantevo suite [23], and a micro-benchmark that demonstrates controllable degrees of imbalance across different levels of vectorization. We also evaluate a real-world application in the form of a molecular dynamics simulation in LAMMPS [84].

NPB IS The NPB IS benchmark applies a bucket sort over a collection of integers. The collection is partitioned into buckets, which are sorted in parallel before being merged into the final sorted collection. Imbalance manifests in this type of sort when buckets end up with significantly different sizes after partitioning the collection. The benchmark partially-addresses imbalance by assigning small buckets to share MPI processes.

NPB BT-MZ The NPB BT-MZ benchmark solves the Navier-Stokes partial differential equations in 3 dimensions by dividing the space into a 2-dimensional mesh of zones that can be solved in parallel, with periodic exchanges of boundary values between neighboring zones. The zones increase in size the further they are from the origin. The benchmark statically balances the work by distributing the zones MPI processes based on their size. For example, many small zones might all be assigned to one process to counterbalance the amount of work performed by a few larger zones in other processes. However, if the MPI

process count is sufficiently large, then there may not be enough small zones available to combine to the amount of work performed by the larger zones, resulting in imbalance. We perform our experiments with class *B*, which exhibits high imbalance on our server.

Mantevo MiniFE MiniFE is a mini-application that serves as a proxy for the main phases of unstructured finite element applications. Most of the time spent in this application is in performing sparse matrix multiplication. MiniFE rebalances work very effectively at the application level, but it includes a command-line option to intentionally leave some imbalance present to simulate scenarios where finite element applications may not be able to resolve all imbalance. We run MiniFE in a configuration that sets the imbalance parameter to 50.

LAMMPS The Large-scale Atomic/Molecular Massively Parallel Simulator is often used for molecular dynamics simulations. LAMMPS partitions the simulation space into subdomains that are assigned to unique processes. LAMMPS supports multiple partitioning schemes, which can be configured by users to help address imbalance at the application level. Imbalance in LAMMPS depends heavily on the input data and scripts that are used to configure the simulator. Imbalance is likely to surface in cases where the input problem cannot be evenly distributed among processes, or where particle density changes significantly throughout simulation. In our experiments, we simulate the collapse of a water column in a container with obstacles. In this case, the simulation begins in an initially-balanced state, but rapidly changes the distribution of work across CPUs as the water flows through the simulation space.

Arithmetic Intensity Micro-benchmark We use the micro-benchmarks introduced in Section 3.2.3 to evaluate floating-point operations per second across workloads that result in different workload-dependent turbo frequency limits, or *license levels* [42]. These micro-

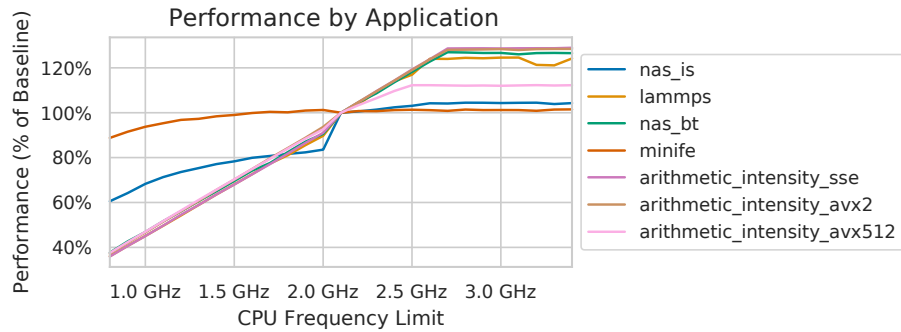


Figure 4-5: Performance of selected benchmarks under various frequency limits. Performance is shown relative to performance at the CPU’s base frequency.

benchmarks are executed in an imbalanced configuration that results in twice as much work on 8 processes per CPU package. This configuration is sized such that only the application’s 8 slowest-progressing processes need to be configured as high-priority in SST-CP. If our algorithm sub-optimally places at least one core in the wrong configuration, then the application’s performance will be impacted. Either the application’s critical path will grow worse as it receives the SST-TF low-priority frequency, or we will fail to achieve the highest possible frequency in Table 4.2 if too many cores are configured with high priority.

Frequency Sensitivity Measurement

Figure 4-5 shows the frequency sensitivity of each application. We repeatedly execute each application under different CPU frequency limits. Performance is measured as the inverse of the time spent in each application’s main compute loop, and visualized with respect to the performance of each application with a frequency limit set to the CPU’s base frequency.

The drop in performance for IS near 2.0 GHz occurs because the power control unit selects a lower uncore frequency setting when *all* cores are configured with low core frequency controls. Future iterations of this rebalancing work may improve by additionally setting controls that guide uncore frequency selection based on application performance.

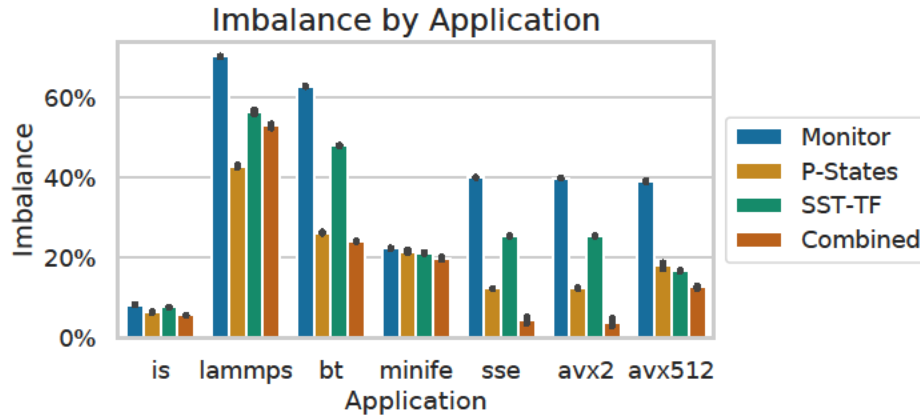


Figure 4-6: Imbalance of the selected benchmarks. 0% indicates perfectly balanced, 100% indicates a lack of concurrent work. Error bars indicate the 95% confidence interval over 5 trials.

However, the current version of the algorithm can safely ignore this behavior since it also ensures that at least one core per CPU package has a high frequency limit.

Applications with greater opportunities in increased frequency limits reach peak performance near their all-core turbo frequency limits. On the other hand, IS and MiniFE do not achieve much improved performance beyond the processor’s base frequency, which is expected from their documented behavior as more memory-sensitive applications. The varying level-off points of the frequency-sensitive applications result from multiple effects. First, they spent different amounts of frequency-sensitive time on their critical paths. Second, different vectorization types have different all-core turbo frequency limits [62].

Imbalance Measurement

Figure 4-6 shows the measured imbalance of each application. We calculate imbalance similar to prior work in load imbalance detection [27]. Imbalance is calculated as $\frac{T_{max} - \bar{T}}{T_{max}} * \frac{n}{n-1}$ for the time T each CPU core spends in non-networking sections of code, over n CPUs used by the application. A value of 0% imbalance indicates a perfectly-balanced application where all processes completed their work in the same amount of time, whereas a value of 100% means that a single process performed all the work.

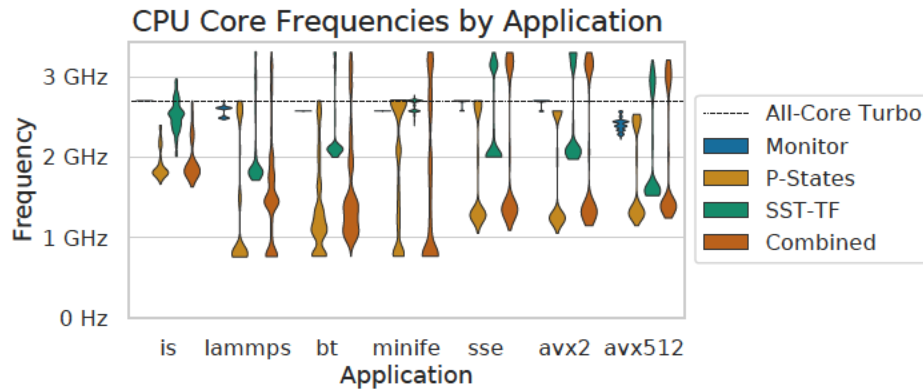


Figure 4-7: Distributions of average frequency across CPU cores and across time in monitor-only runs and under frequency-control variants of the performance rebalancer.

If imbalanced applications are not CPU-frequency-bound, as shown in Figure 4-5, then we only expect energy savings. But if applications are imbalanced and frequency-bound, then we also expect opportunities to improve performance with guided turbo.

4.3.3 Measured Impacts of Performance-Guided Frequency Boosting

In this section, we evaluate how each application responds to the different frequency-control variants of the method described in Section 4.3.1. One variant only applies the desired frequency control setting through P-States. Another only configures SST-CP classes of service with SST-TF enabled. Lastly, a combined variant applies both sets of controls. These are all compared to baseline performance from monitoring-only runs, as described in Section 4.3.2.

Achieved Frequencies

Figure 4-7 shows how each policy results in different achievable frequencies across cores in each application. The monitored runs of most applications are able to achieve the system’s all-core turbo frequency of 2.7 GHz, while the other runs apply frequency controls to achieve varied ranges of core frequencies.

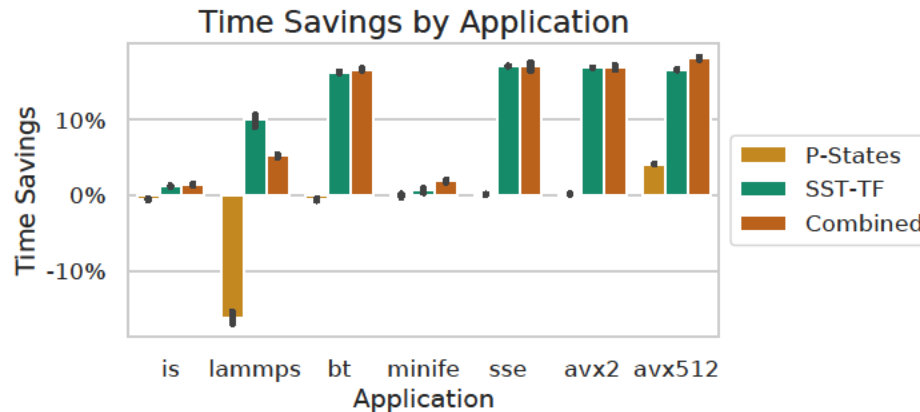


Figure 4.8: Time savings for imbalanced applications under different frequency-control variants of the performance rebalancer. Savings are relative to monitor-only runs. Error bars indicate the 95% confidence interval over 5 trials.

The system is already able to achieve all-core turbo frequency for all cores in most workloads, so the P-States variant is not able to increase frequency on any cores even when it heavily throttles some of the cores, as in `nas_bt`. However, the variants with SST-TF successfully exchange lower frequency on some cores for higher frequency on other cores.

The `AVX512` micro-benchmark is frequently throttled due to voltage regulator design constraints, so it spends a significant amount of samples at lower frequencies. While the upper end of achieved frequencies is lower with the P-State-only approach, the distribution is shifted such that the cores on the critical path are throttled for voltage regulator design constraints less often.

Time Savings

Time savings are summarized in Figure 4.8. The results show at least some performance-improving opportunity in each of the imbalanced application configurations, and highlights a limitation of the current method that motivates potential for future improvement.

All three of the arithmetic intensity micro-benchmarks are able to achieve nearly the same performance improvement with SST-TF because each of their critical paths has about

2.5x as much work as the rest of their processes, they are all highly performance-sensitive to frequency, and they operate in the highest-frequency SST-TF configuration for the duration of their runs. In each case, the highest-frequency configuration results in a similar relative speedup compared to the execution time of the monitor-only run (constrained by regular all-core turbo frequency limits) for that application. Although the BT-MZ class B input results in even greater imbalance than the amount configured in the micro-benchmarks, we do not seize additional performance improvement since we are already achieving the maximum-allowed frequency under this server's SST-TF properties.

The main performance outlier is P-State-only control of LAMMPS, which is the only result with significant performance degradation. The degraded performance in LAMMPS comes mainly from the rapidly-changing behavior of imbalance in that application, where around 8 application epochs finish executing between each sampling period in our method. For example, we may observe little to no non-networking time on a CPU core in one period, and consequently assign a low frequency limit to that core. But new work may shift to that core by the next period, impacting the performance of multiple epochs in the meantime. Although the *combined* variant also suffers this limitation, it is able to mitigate the performance loss by speeding up the cores that are consistently given a lot of work by the application. Future work may seek to improve this case by investigating alternative responses to prediction errors.

On the other hand, the P-State-only variant is able to avoid performance degradation in all the applications that exhibit steady imbalance, and even achieves a performance improvement in the AVX512 micro-benchmark. This is expected since most of the applications are able to achieve their maximum all-core turbo frequency without throttling any of the cores. The AVX512 micro-benchmark is often frequency-limited by the CPU's power management unit when all cores are in the high-power-consuming region of interest. In that case, the P-State-only solution throttles the leader cores, which creates enough headroom for the

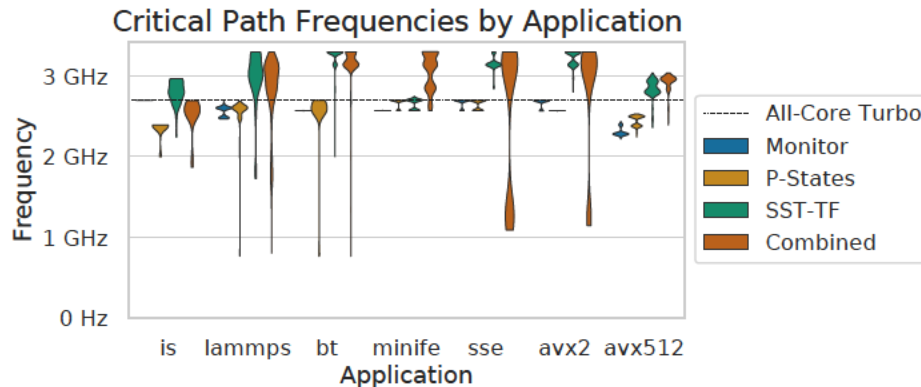


Figure 4-9: Distributions of average achieved CPU core frequency for cores that spend the most non-networking time in each iteration of the application.

larger cores to achieve higher frequencies and speed up the application. The *SST-TF* and *combined* variants achieve even more performance improvement because they enable higher peak turbo frequency limits in the lagging cores' region of interest.

To illustrate the value of throttling the non-critical-path cores as in the P-State-only variant on the AVX512 micro-benchmark, Figure 4-9 shows the achieved core frequencies while ignoring cores that were not on the critical path for each application iteration. In the AVX512 example, we see that selective throttling of non-critical-path cores lets us *increase* the average frequency on cores that end up in the critical path. We increase the frequency of critical-path cores even further across all the applications by adding SST-TF control.

In Section 4.3.2, we observe that the overall behavior of the MiniFE benchmark is insensitive to higher all-core frequencies above the processor's base frequency. However, we observe that increased peak turbo frequencies on the application's critical path do offer some performance improvement opportunities for MiniFE. This benchmark spends a significant amount of time in sparse matrix vector multiplication regions of code. While those regions are typically non-compute bound, they can exhibit moderate temporal locality as values in the vector are repeatedly accessed for the computation. This indicates that there may be performance opportunities for selectively higher turbo frequencies in imbalanced applications even if they do not appear to be largely core-frequency-sensitive at a high level.

Energy Savings

Energy savings in these runs result from power reductions due to frequency throttling, and from time reductions due to increased ranges of usable turbo frequency, as is visible in Figure 4.10. The P-State-only variant reduces energy despite showing little to no time improvement in Figure 4.8, indicating the throttling-induced savings. The additional energy savings on the combined variant are due to the run time reduction that comes with performance-guided turbo utilization in those runs.

The BT-MZ benchmark shows greater energy savings than the other evaluated applications due to its heavy imbalance. We are executing the benchmark's class *B* input, which exhibits imbalance because the application's work cannot be evenly distributed across all application processes. The laggiest process receives about 4 times as much work as the process that finishes each iteration first. As a result, many of the cores executing this application can be throttled to very low frequencies. We observed that BT-MZ did not achieve more performance improvement than the imbalanced micro-benchmarks in Section 4.3.3 because we were already executing the application's critical path at the maximum-allowed SST-TF frequency. However, we do observe greater energy-saving opportunity through throttling the non-critical-path CPU cores.

Since the P-State-only variant can generally only balance by slowing down parts of the application, the desired behavior is to achieve energy savings while minimizing performance degradation.

A useful takeaway from this comparison is that compute-bound, imbalanced applications stand to *reduce* their energy consumption by using performance-guided turbo. Although the more memory-bound imbalanced applications are able to achieve most of their energy savings from P-State throttling alone, we are able to convert some energy savings back to performance improvement when bursts of turbo utilization may improve performance.

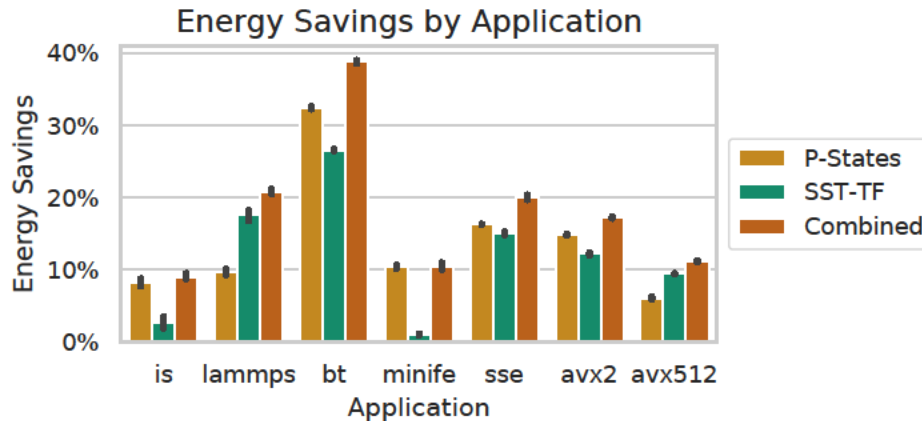


Figure 4-10: Energy savings for imbalanced applications under different frequency-control variants of the performance rebalancer. Savings are relative to monitor-only runs. Error bars indicate the 95% confidence interval over 5 trials.

Takeaways for Power Management Algorithms

The variants of our core prioritization method discussed in this section all utilize the same technique to determine critical path and to determine the CPU core frequencies that would achieve the desired execution time. Although the same balancing decisions are applied across variants, there are significant differences in behavior by applying different combinations of P-State control and SST-TF configuration from those decisions.

The P-State-only approach is typically limited to reducing energy consumption while seeking unharmed performance. However, in cases where application performance is limited by power constraints (*power-bound*) such as the AVX512 micro-benchmark, there is additional value in throttling some cores to enable speedups on other cores. Aside from workloads that are naturally power-bound, this may be useful for algorithms intended to rebalance applications on power-capped systems.

The SST-TF-only approach offers improved performance by increasing the peak turbo frequency on an application’s critical path. Our method applies a conservative limit to the low-priority cores, allowing SST-TF to restrict the frequency based on the active turbo license level. As a result, the SST-TF-only approach responds to the rapidly-changing CPU

demands of the LAMMPS experiment with less risk to performance degradation. However, the conservative approach leaves behind more imbalance (Figure 4.6) compared to the solutions with more aggressive throttling on the low-priority end, and ultimately leaves some energy savings on the table.

Further work in rebalancing with performance-guided turbo may seek to reduce the trade-off between energy and performance opportunities by exploring more of the control space in SST-TF (e.g., by using all four priority levels instead of one high and one low, or by exploring less conservative frequency range configurations in SST-TF).

Takeaways for Imbalanced Applications

Our experiments cover applications with multiple characteristics of imbalance. Overall we have the greatest opportunity when the critical path is co-located on a CPU package with non-critical paths that can be slowed down. Some applications (e.g., LAMMPS) have application-level work rebalancers that are aware of the hardware topology. On systems capable of performance-guided turbo configuration, application-level rebalancers could consider how to spread any unresolved imbalance across CPU packages if it is not in conflict with other resource-oriented rebalancing objectives.

4.4 Concluding Remarks in Performance-Aware Power Management

We investigate imbalance in bulk-synchronous parallel applications as a specific challenge in performance-aware power management, where uneven amounts of work are distributed across a computing platform. In Section 4.3, we introduce a method to speed up imbalanced applications by managing power with combined knowledge of configurable hardware power-performance trade-offs and awareness of application performance. We improve energy savings and performance by informing a job-level power manager of non-uniform power-performance trade-offs across CPU cores made available by a hardware-level power manager.

Chapter 5

Conclusions and Future Work

As we progress through the exascale era of computing, high-performance computing facilities need to manage increasingly large amounts of power. Site-level power constraints are growing in complexity as data centers need to coordinate their power demand with their electricity providers or react to extreme energy-pricing volatility. High-performance computing software interacts with a computing system in diverse ways, exhibiting different degrees of parallelism and power-performance sensitivity across computing resources. Each layer of power management in a data center exposes different capabilities for monitoring and control, and does not always have visibility into what is happening in the neighboring layers.

This thesis claims that multi-tiered power management methods enable data centers to implement site-wide power management policies that accurately respond to changing power constraints and react to application-specific performance impacts. We support our thesis by investigating opportunities in multi-tiered power management and job performance awareness in high-performance computing scenarios.

Our investigations throughout this thesis reveal opportunities to improve a system's energy efficiency and to enable accurate, cost-saving participation in demand response programs under performance constraints. We achieve those benefits by increasing the scope of power and performance visibility within cluster-level power managers and job execution runtimes. Our works investigate methods to incrementally improve power management techniques in a cluster tier or in a job tier. We also show that multiple power management

tiers can work together through a unified software framework that loosely couples tiers through periodic communication of power objectives and measured performance properties.

Each of our investigations opens opportunities for future work that increases the depth or accuracy of introduced power management methods. Furthermore, our proposed ANOR framework offers future opportunities to explore how job-specific performance solutions can be integrated into cluster-wide power managers.

5.1 Multi-Level Power Management

While designing HPC sites under a power constraint, there arises a need to provide a power management policy that works well in the common case, and minimizes the loss of quality of service in exceptional cases. We describe power management solutions that combine cluster power awareness with knowledge from the surrounding site and job tiers in Chapter 3. We show that multi-level power managers enhance cost, energy, and performance opportunities compared to single-level power managers, and we show that feedback between levels improves cases where modeled behavior does not match actual behavior.

We present an opportunity analysis highlighting the need to integrate application awareness with system-level resource awareness (Section 3.2). It is well-established that application-aware power management solutions can exhibit efficiency gains in a power-constrained environment. However, application-awareness alone fails to account for system-level power constraints that dictate HPC site policies. We design a unified power management policy that builds on existing efforts in system-level dynamic power reallocation and in application-aware power management. We present empirical evidence to suggest that integrating application and system-level solutions together leads to improved savings, showing up to 7% reduction in system time and up to 11% savings in energy. Across a variety of workload mixes, we show that a unified power manager can match or improve the energy savings of either a job power manager or a system power manager used in isolation.

We continue beyond our opportunity analysis by motivating the need for a feedback-driven multi-tiered cluster power management stack (Section 3.3). Although a unified power management solution performs well when job properties and system power caps are known in advance, both may actually change over time. We present ANOR, which loosely couples power management frameworks at cluster level and job level in order to meet time-varying system-wide power and performance constraints while reacting to unexpected job power-performance properties.

Job power and performance properties may be estimated in advance of a job's execution through prediction models or as user-provided information. We explore options to budget cluster power across jobs under ANOR by using prior-known power and performance information. Our analysis reveals trends where job performance awareness is expected to improve system performance under a cluster power cap versus a performance-agnostic policy, but also reveals that performance degrades when pre-characterization data is missing or inaccurate. We show that ANOR's performance feedback mechanism enables the cluster power manager to mitigate the resulting performance degradation.

We demonstrate a practical implementation of a QoS-aware data center demand response policy through the ANOR framework, successfully tracking power in many cases, and identifying practical challenges for others. Furthermore, we show a performance variation analysis in 1000-node cluster simulations. Although performance feedback enables the power manager to steer power toward underperforming jobs, it does not have a significant impact on individual job performance in scenarios where the majority of QoS impact comes from job scheduling decisions.

Future work may explore multiple methods to resolve the performance variation challenge in demand response policies. One option is to place more emphasis on power modulation through node power capping, which can be adjusted after jobs are already scheduled. By relying more on node power capping during expected performance behavior, there is

room to *increase* power beyond its initial cap as a way to accelerate an application. Such a decision reduces the range of power control that AQA can offer for demand response, but may result in greater node utilization while offering a reasonable trade-off under tight QoS constraints.

Another option to resolve the performance variation challenge is to re-fit the weights assigned to each job queue when unexpected rates of performance degradation are detected. Our preliminary investigation into this approach suggests that the scheduler may become over-constrained between QoS and power-tracking constraints in some cases. Demand response policies designed to be resilient to these types of scenarios may benefit from incorporating performance-modeling uncertainty as a component of the bidding strategy (in essence, bidding with more power to exchange less cost savings for greater resilience).

Multi-tier power management may introduce scalability challenges when there are many concurrent jobs. Our example implementation of ANOR limits the depth of control needed in the widely-scoped cluster tier by delegating application-aware and node-aware processing to the job tier. The cluster tier needs to know the coefficients of the job performance model to distribute power effectively, and it needs to issue new power caps to all jobs whenever the distribution shifts. Future work may investigate methods to reduce the required communication or to localize it within additional control hierarchy.

Future work may investigate ways to extend the ANOR framework to other types of data centers. We evaluate ANOR framework with an HPC cluster while executing applications with bulk-synchronous parallelism, a common parallel design pattern used in HPC software. Conceptually, the ANOR framework enables performance-aware power sharing across multiple computing units (compute nodes in our implementation) by reacting to performance measurements from worker units (jobs in our implementation). Other types of data center workloads may be compatible after solving additional job-tier design challenges. For example, cloud computing platforms are likely to map multiple virtual machines to a single

physical machine, which can potentially share power-capping domains (which are typically available at CPU package granularity in current hardware). This challenge may be addressed by using alternative power management controls that are available at more fine-grained levels of scope, such as frequency throttling controls. Doing so would require also modeling the power consumption as a function of the chosen controls and the current executing workload properties.

There are many additional opportunities for future work with the ANOR framework by exploring how existing cluster power management solutions and existing job performance solutions may integrate with each other. For example, our implementation demonstrates how performance responds to a CPU power-capping mechanism. Other power cappers, such as GPU, memory, or node-level could similarly be used. Furthermore, existing online policies to optimize job performance could be modified to report performance up to the cluster tier, enabling a cluster with a range of job-specific performance challenges to still achieve performance-aware system-wide power capping.

Lastly, we demonstrate that adding a simple site-wide power model to existing QoS-aware demand response policies for cluster power management can improve energy cost-saving opportunities while still meeting the data center’s QoS constraints (Section 3.4). We perform simulations of multiple mixes of applications on a 6000-server cluster with a simple PUE model for site-wide power consumption. We demonstrate that with accurate PUE-based power predictions, we can seize 1.3x cost savings compared to QoS-aware demand response without site-wide power awareness.

Our evaluation of experimental results with the site-wide power model includes some observations that could be applied for future improvements to QoS-aware demand response management algorithms. First, the job weights learned by the AQA policy’s training mechanism depend on the relationship between server power properties and job performance properties, but not non-server power properties of the site. Future work may use this

knowledge to reduce the search space when working with long-lived workload mixes across changing site-wide power efficiency. Second, we note that there are competing risks in QoS and power tracking, from over-bidding or underbidding due to site-wide power-modeling inaccuracy.

Future work in site-wide or other multi-tier power management policies could investigate methods to proactively adjust bids for reduced risk when operating in a low-confidence region of power or performance models. Our investigation in ANOR (Section 3.3) suggests that performance variation may cause overly aggressive bids when they aren't made with variation in mind. Our investigation in site-wide demand response through cluster-level controls (Section 3.4) suggests that uncertainty in the site power model may result in suboptimal bids. One solution may be to introduce random variations when performing the simulation-driven search for demand response bids. The properties of randomized variation may be chosen to match the degree of resilience a data center wishes to have for modeling uncertainty. Such a solution may increase the time it takes to converge to the preferred bid, so an alternative approach may more efficiently use robust optimization to select a bid by including the random distribution of uncertainty as part of the optimization constraints.

5.2 Adapting Job Schedules for Changing Performance Properties

Each section of Chapter 3 demonstrates a way that a JSRM can use knowledge of pre-characterized job power and performance properties to achieve power, energy, performance, and cost objectives. Section 3.3 further demonstrates how a cluster-wide power manager can mitigate the performance degradation from misinformed characterization-driven decisions when it can use online performance feedback from a job-level execution runtime. However, our results also reveal that power capping alone is insufficient to address the impact that misinformed decisions have on sojourn time in a scheduler that operates under both job performance constraints and cluster power constraints.

Future work may investigate ways to overcome this limitation by enabling the *job scheduler* to make informed decisions when executing jobs behave different from prior expectations. Furthermore, future work may consider how to design demand response policies that enable data centers to maximize their commitment to power flexibility without becoming over-constrained by their cluster power target and job performance constraints.

5.3 Performance-Aware Job-Level Power Management

In addition to sharing information across software power management tiers, it is important to coordinate software power management with the hardware power managers it builds on. Frequency boosting mechanisms enable CPUs to opportunistically achieve greater peak operating frequency without violating design constraints. The Ice Lake family of Intel CPUs introduces software interfaces to ask for information about how the hardware power manager makes boosting decisions, and interfaces to guide the hardware power manager's decisions.

We demonstrate opportunities to use application performance awareness to configure SST-CP and SST-TF for reduced energy *and* increased performance in imbalanced bulk-synchronous MPI workloads (Section 4.3). Our evaluations show performance improvements of up to 18% in highly-imbalanced and compute-bound benchmarks. We also demonstrate up to 10% performance improvement in a real-world application that exhibits compute-bound imbalance. The resulting time savings highlight the value of integrating job performance awareness with knowledge of hardware power management trade-offs.

Further improvements can still be seized in our evaluated workloads. Future works may improve on these results by sharing power across compute nodes, exploring additional parallel design patterns, and using performance models that account for non-compute-bound imbalance.

Opportunities With Power Sharing Across Compute Nodes We present a solution that locally balances the non-networking time within each CPU package, and thus only

improves the application's critical path when *intra*-package imbalance is present within the CPU package that is executing the application's critical path. A distributed power cap may introduce additional opportunities when combined with our solution. That is, any *inter*-package rebalancing performed by distributed power capping will close the gap between packages, improving the chance that any package-local time improvements may impact the total application execution time. Furthermore, executing under a power cap may amplify the benefit of *intra*-package frequency guidance as more cores experience frequency throttling due to power constraints.

Future work may investigate performance-guided frequency boosting at multiple levels. One level of improvement may come from sharing power across nodes within or across jobs. Another level of improvement may come from additionally providing performance feedback through a multi-tiered power management framework such as ANOR. Jobs with different degrees of imbalance expose different responses to performance-guided frequency boosting, which may provide additional opportunities to a performance-guided cluster power manager.

Opportunities With Other Parallel Programming Patterns The current version of the algorithm is designed for bulk-synchronous parallel MPI applications. Future work may consider extending the performance monitoring mechanism to other metrics of imbalance. In order to make the performance-guided frequency boosting algorithm applicable to other parallel design patterns, it will be necessary to detect additional types of non-compute-sensitive regions of code in an application. For example, many HPC bulk-synchronous applications use OpenMP loops within each server. In that example, our solution could be extended to also consider non-OpenMP-barrier time as a metric of interest.

Performance Model Opportunities Our performance-aware power management method shapes CPU core frequency allowances by deciding whether the performance of application regions is likely to be sensitive to CPU core frequency. However, many workloads spend

significant amounts of time in regions of code that are more performance sensitive to other resources, such as cache, memory, and accelerators. Future work may investigate how to apply this method to non-CPU-core resources in imbalanced applications. For example, Veitch et al. accompany base frequency configuration with cache allocation and memory bandwidth steering to prioritize separate applications on shared hardware [86]. Similar benefits might be achievable for separate processes in an imbalanced memory-bound workload.

Future work may extend our power management method to work with systems that have heterogeneous computational components, such as mixes of accelerators and CPUs. In such a system, the frequency selection mechanism can be separately instantiated for each type of controllable resource (e.g., a selector for CPUs and another for accelerators). Each instance of the frequency selection mechanism needs to be aware of the division between potentially frequency-sensitive time and remainder time spent in each application loop iteration. An extension to our method may use device activity metrics (e.g., GPU utilization and activity metrics available in NVML, DCGM, and LevelZero software libraries) to aid that time division step.

Appendix A

Open-Source Software Contributions

Several of the works in this thesis introduce software power management mechanisms built on top of the Global Extensible Open Power Manager (GEOPM) project¹⁰. Our open-source software contributions and their relevant sections in this thesis are listed below, following a description of technical terminology regarding GEOPM.

GEOPM provides an extensible framework for software power management. Key extensible components are *IOGroups* and *Agents*. An IOGroup is an extension that lets GEOPM read current state (e.g., a CPU's power draw, or an application's rate of completion) and write controls (e.g., a power cap). An agent is an extension that implements a power management policy that periodically writes to IOGroup controls, making management decisions that are informed by state read from IOGroups.

Imbalanced micro-benchmark (Section 3.2.3, Section 4.3.2) We provide a microbenchmark with controllable imbalance, vectorization, and license level. Source code is available at <https://github.com/dannosliwcd/arithmetic-intensity>.

GEOPM host_power_governor agent (Section 3.2.4) We add a per-host GEOPM power governor, used for site-wide power management opportunity analysis experiments. Source code is available at https://github.com/dannosliwcd/geopm/tree/dcw/host_power_governor/tutorial/host_power_governor

¹⁰<https://geopm.github.io/>

ANOR (Section 3.3.2) We implement a cluster-tier power management policy that communicates with a job-tier power capper through TCP sockets as an example implementation of ANOR. Source code is available at <https://github.com/dannosliwcd/geopm/tree/dcw/site-power>.

GEOPM frequency_balancer agent (Section 4.3.2) We create a GEOPM Agent that implements our performance-guided turbo frequency policy. Source code is available at <https://github.com/dannosliwcd/geopm/tree/single-node-rebalancer>. Agent documentation is included in that repository at `service/docs/source/geopm_agent_frequency_balancer.7.rst`.

GEOPM SSTIOGroup (Section 4.3.2) This IOGroup enables interaction with SST-CP and SST-TF through GEOPM. Source code is merged into the GEOPM project at <https://github.com/geopm/geopm/blob/dev/service/src/SSTIOGroup.cpp>. Documentation is located at https://geopm.github.io/geopm_pio_sst.7.html.

Bibliography

- [1] Bilge Acun, Kavitha Chandrasekar, and Laxmikant V. Kale. Fine-Grained Energy Efficiency Using Per-Core DVFS with an Adaptive Runtime System. In *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. 2019, pp. 1–8. DOI: [10.1109/IGSC48788.2019.8957174](https://doi.org/10.1109/IGSC48788.2019.8957174).
- [2] Altair. PBSPro Administrator Guide 2020. <https://www.altair.com/pdfs/pbsworks/PBSAdminGuide2020.1.pdf>.
- [3] AMD. Linux Tuning Guide, AMD Opteron 6200 Series Processors. Apr. 2012.
- [4] Eishi Arima, A. Isaías Comprés, and Martin Schulz. On the Convergence of Malleability and the HPC PowerStack: Exploiting Dynamism in Over-Provisioned and Power-Constrained HPC Systems. In *High Performance Computing. ISC High Performance 2022 International Workshops*. Ed. by Hartwig Anzt, Amanda Bienz, Piotr Luszczek, and Marc Baboulin. Cham: Springer International Publishing, 2022, pp. 206–217. ISBN: 978-3-031-23220-6.
- [5] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A Case Study of Energy Aware Scheduling on SuperMUC. In *Supercomputing*. Ed. by Julian Martin Kunkel, Thomas Ludwig, and Hans Werner Meuer. Cham: Springer International Publishing, 2014, pp. 394–409. ISBN: 978-3-319-07518-1.
- [6] Anna Maria Bailey. The Evolution of High Performance Computing (HPC) and the Path to Exascale at Lawrence Livermore National Laboratory (LLNL). <https://drive.google.com/file/d/1DTAE3Qo15TXvzhaR309iUhU7HQu-7h5e/view>. Keynote presentation at the Energy Efficient HPC SOP Workshop, International Green and Sustainable Computing Conference (IGSC). Oct. 2022.
- [7] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*, Third Edition. Morgan & Claypool, 2019. ISBN: 9781681734347. DOI: [10.2200/S00874ED3V01Y201809CAC046](https://doi.org/10.2200/S00874ED3V01Y201809CAC046).
- [8] Natalie Bates, Ghaleb M Abdulla, Gregory A Koenig, Sridutt Bhalachandra, Mehdi Sheikhalishashi, and Tapasya Patki. The Electrical Grid and Supercomputer Centers: An Investigative Analysis of Emerging Opportunities and Challenges. In *Informatik Spektrum* 38 (2015), pp. 111–127. DOI: [10.1007/s00287-014-0850-0](https://doi.org/10.1007/s00287-014-0850-0).

- [9] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. Predictive Modeling for Job Power Consumption in HPC Systems. In *High Performance Computing*. Ed. by Julian M. Kunkel, Pavan Balaji, and Jack Dongarra. Springer International Publishing, 2016, pp. 181–199. ISBN: 978-3-319-41321-1. DOI: [10.1007/978-3-319-41321-1_10](https://doi.org/10.1007/978-3-319-41321-1_10).
- [10] Jeff Broughton. HPC Energy Efficiency in the Exascale Era. <https://datacenters.lbl.gov/sites/default/files/SC%202019%20EE%20HPC%20Keynote%20V2.pdf>. Keynote talk at the Energy Efficient HPC Workshop, SC 2019. Nov. 2019.
- [11] Christopher Cantalupo, Jonathan Eastep, Siddhartha Jana, Masaaki Kondo, Matthias Maiterth, Aniruddha Marathe, Tapasya Patki, Barry Rountree, Ryuichi Sakamoto, Martin Schulz, and Carsten Trinitis. A Strawman for an HPC PowerStack. United States, Aug. 2018. DOI: [10.2172/1466153](https://doi.org/10.2172/1466153).
- [12] Daniele Cesarini, Andrea Bartolini, Piero Bonfà, Carlo Cavazzoni, and Luca Benini. COUNTDOWN: A Run-Time Library for Application-Agnostic Energy Saving in MPI Communication Primitives. In *Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems*. ANDARE '18. Limassol, Cyprus: Association for Computing Machinery, 2018. ISBN: 9781450365918. DOI: [10.1145/3295816.3295818](https://doi.org/10.1145/3295816.3295818).
- [13] Daniele Cesarini, Andrea Bartolini, Andrea Borghesi, Carlo Cavazzoni, Mathieu Luisier, and Luca Benini. Countdown Slack: A Run-Time Library to Reduce Energy Footprint in Large-Scale MPI Applications. In *IEEE Transactions on Parallel and Distributed Systems* 31.11 (2020), pp. 2696–2709. DOI: [10.1109/TPDS.2020.3000418](https://doi.org/10.1109/TPDS.2020.3000418).
- [14] Hao Chen, Michael C. Caramanis, and Ayse K. Coskun. The data center as a grid load stabilizer. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2014, pp. 105–112. DOI: [10.1109/ASPDAC.2014.6742874](https://doi.org/10.1109/ASPDAC.2014.6742874).
- [15] Hao Chen, Ayse K. Coskun, and Michael C. Caramanis. Real-time power control of data centers for providing Regulation Service. In *52nd IEEE Conference on Decision and Control*. 2013, pp. 4314–4321. DOI: [10.1109/CDC.2013.6760553](https://doi.org/10.1109/CDC.2013.6760553).
- [16] Hao Chen, Yijia Zhang, Michael C. Caramanis, and Ayse K. Coskun. EnergyQARE: QoS-Aware Data Center Participation in Smart Grid Regulation Service Reserve Provision. In *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 4.1 (Jan. 2019). ISSN: 2376-3639. DOI: [10.1145/3243172](https://doi.org/10.1145/3243172).
- [17] Shuang Chen, Angela Jin, Christina Delimitrou, and José F. Martínez. ReTail: Opting for Learning Simplicity to Enable QoS-Aware Power Management in the Cloud. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Seoul, Korea: IEEE, 2022, pp. 155–168. DOI: [10.1109/HPCA53966.2022.00020](https://doi.org/10.1109/HPCA53966.2022.00020).

- [18] Andrew A. Chien, Richard Wolski, and Fan Yang. The Zero-Carbon Cloud: High-Value, Dispatchable Demand for Renewable Power Generators. In *The Electricity Journal* 28.8 (2015), pp. 110–118. ISSN: 1040-6190. DOI: <https://doi.org/10.1016/j.tej.2015.09.010>.
- [19] Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. A Roofline Model of Energy. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, May 2013, pp. 661–672. ISBN: 978-1-4673-6066-1. DOI: [10.1109/IPDPS.2013.77](https://doi.org/10.1109/IPDPS.2013.77).
- [20] Anders Clausen, Gregory Koenig, Sonja Klingert, Girish Ghatikar, Peter M. Schwartz, and Natalie Bates. An Analysis of Contracts and Relationships between Supercomputing Centers and Electricity Service Providers. In *Workshop Proceedings of the 48th International Conference on Parallel Processing*. ICPP Workshops '19. Kyoto, Japan: Association for Computing Machinery, 2019. ISBN: 9781450371964. DOI: [10.1145/3339186.3339209](https://doi.org/10.1145/3339186.3339209).
- [21] Julita Corbalan and Luigi Brochard. EAR: Energy management framework for supercomputers. Technical Report. Barcelona Supercomputing Center (BSC), 2019. URL: <https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear.pdf>.
- [22] Julita Corbalan, Oriol Vidal, Lluís Alonso, and Jordi Aneas. Explicit uncore frequency scaling for energy optimisation policies with EAR in Intel architectures. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. 2021, pp. 572–581. DOI: [10.1109/Cluster48925.2021.00089](https://doi.org/10.1109/Cluster48925.2021.00089).
- [23] Paul Stewart Crozier, Heidi K Thornquist, Robert W Numrich, Alan B Williams, Harold Carter Edwards, Eric Richard Keiter, Mahesh Rajan, James M Willenbring, Douglas W Doerfler, and Michael Allen Heroux. Improving performance via mini-applications. Sandia National Laboratories, United States, Sept. 2009. DOI: [10.2172/993908](https://doi.org/10.2172/993908).
- [24] Howard David, Eugene Gorbatoov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED '10. Austin, Texas, USA: Association for Computing Machinery, 2010, pp. 189–194. ISBN: 9781450301466. DOI: [10.1145/1840845.1840883](https://doi.org/10.1145/1840845.1840883).
- [25] Daniele De Sensi and Marco Danelutto. Application-Aware Power Capping Using Nornir. In *Parallel Processing and Applied Mathematics*. Ed. by Roman Wyrzykowski, Ewa Deelman, Jack Dongarra, and Konrad Karczewski. Cham: Springer International Publishing, 2020, pp. 191–202. ISBN: 978-3-030-43222-5.
- [26] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware cluster management. In *ACM SIGPLAN Notices* 49.4 (2014), pp. 127–144.

- [27] Luiz DeRose, Bill Homer, and Dean Johnson. Detecting Application Load Imbalance on High End Massively Parallel Systems. In *Euro-Par 2007 Parallel Processing*. Ed. by Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 150–159. ISBN: 978-3-540-74466-5.
- [28] Pierre-François Dutot, Yiannis Georgiou, David Glessner, Laurent Lefevre, Millian Poquet, and Issam Rais. Towards Energy Budget Control in HPC. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. CCGrid '17. Madrid, Spain: IEEE Press, 2017, pp. 381–390. ISBN: 9781509066100. DOI: [10.1109/CCGRID.2017.16](https://doi.org/10.1109/CCGRID.2017.16).
- [29] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. In *High Performance Computing*. Ed. by Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes. Cham: Springer International Publishing, 2017, pp. 394–412. ISBN: 978-3-319-58667-0.
- [30] Efficiency - Data Centers - Google. <https://www.google.com/about/datacenters/efficiency/>. 2021.
- [31] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, and Martin Schulz. POW: System-wide dynamic reallocation of limited power in HPC. In *HPDC 2015 - Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (2015)*, pp. 145–148. DOI: [10.1145/2749246.2749277](https://doi.org/10.1145/2749246.2749277).
- [32] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L. Rountree, and Mark E. Femal. Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications. In *IEEE Transactions on Parallel and Distributed Systems* 18.6 (2007), pp. 835–848. DOI: [10.1109/TPDS.2007.1026](https://doi.org/10.1109/TPDS.2007.1026).
- [33] Yangyang Fu, Xu Han, Kyri Baker, and Wangda Zuo. Assessments of data centers for provision of frequency regulation. In *Applied Energy* 277 (2020), p. 115621. ISSN: 0306-2619. DOI: <https://doi.org/10.1016/j.apenergy.2020.115621>.
- [34] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. PShifter: Feedback-Based Dynamic Power Shifting within HPC Jobs for Performance. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '18. Tempe, Arizona: Association for Computing Machinery, 2018, pp. 106–117. ISBN: 9781450357852. DOI: [10.1145/3208040.3208047](https://doi.org/10.1145/3208040.3208047).
- [35] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*. ISCA '11. San Jose, California, USA: Association for Computing Machinery, 2011, pp. 341–352. ISBN: 9781450304726. DOI: [10.1145/2000064.2000105](https://doi.org/10.1145/2000064.2000105).

- [36] Pouya Haghi, Anqi Guo, Tong Geng, Anthony Skjellum, and Martin C. Herbordt. Workload Imbalance in HPC Applications: Effect on Performance of In-Network Processing. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 2021, pp. 1–8. DOI: [10.1109/HPEC49654.2021.9622847](https://doi.org/10.1109/HPEC49654.2021.9622847).
- [37] C. Hankendi, A. K. Coskun, and H. Hoffmann. Adapt&Cap: Coordinating System- and Application-Level Adaptation for Power-Constrained Systems. In *IEEE Design & Test* 33.1 (2016), pp. 68–76.
- [38] HPE. XC(tm) Series User Application Placement Guide - S-2496. URL: https://pubs.cray.com/bundle/XC_Series_User_Application_Placement_Guide_CLE60UP01_S-2496/page/Run_Applications_Using_the_aprun_Command.html.
- [39] Demand Response. <https://www.iea.org/reports/demand-response>. IEA, 2022.
- [40] World Energy Outlook 2022. <https://www.iea.org/reports/world-energy-outlook-2022>. IEA, 2022.
- [41] Intel Advisor. <https://software.intel.com/en-us/advisor>. [Online; accessed 2020-03-04].
- [42] Intel® 64 and IA-32 Architectures Optimization Reference Manual. <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>. Feb. 2022.
- [43] Intel® 64 and IA-32 Software Developer’s Manual, Volume 3. <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>. Mar. 2023.
- [44] International Energy Outlook 2021. https://www.eia.gov/outlooks/ieo/pdf/IEO2021_ChartLibrary_Electricity.pdf. Oct. 2021.
- [45] Ali Jahanshahi, Nanpeng Yu, and Daniel Wong. PowerMorph: QoS-aware server power reshaping for data center regulation service. In *ACM Transactions on Architecture and Code Optimization (TACO)* 19.3 (2022), pp. 1–27.
- [46] Bran Knowles. ACM TechBrief: Computing and Climate Change. In *ACM Technology Policy Council* (Nov. 2021).
- [47] Yuetsu Kodama, Tetsuya Odajima, Eishi Arima, and Mitsuhsa Sato. Evaluation of Power Management Control on the Supercomputer Fugaku. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 2020, pp. 484–493. DOI: [10.1109/CLUSTER49012.2020.00069](https://doi.org/10.1109/CLUSTER49012.2020.00069).
- [48] Sunil Kumar, Akshat Gupta, Vivek Kumar, and Sridutt Bhalachandra. Cuttlefish: Library for Achieving Energy Efficiency in Multicore Parallel Programs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’21. St. Louis, Missouri: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: [10.1145/3458817.3476163](https://doi.org/10.1145/3458817.3476163).

- [49] Jacklin Kwan. Climate change threatens supercomputers. In *Science (New York, NY)* 378.6616 (2022), pp. 124–124. URL: <https://www.science.org/doi/10.1126/science.adf2882>.
- [50] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*. Vol. 40. New York, NY, USA: Association for Computing Machinery, June 2012, pp. 175–186. DOI: [10.1145/2318857.2254779](https://doi.org/10.1145/2318857.2254779).
- [51] Zhenhua Liu, Adam Wierman, Yuan Chen, Benjamin Razon, and Niangjun Chen. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*. New York, NY, USA: Elsevier, 2013, pp. 341–342.
- [52] LLNL: msr-safe. <https://github.com/LLNL/msr-safe>. [Online; accessed 2020-04-02].
- [53] Matthias Maiterth, Gregory Koenig, Kevin Pedretti, Siddhartha Jana, Natalie Bates, Andrea Borghesi, Dave Montoya, Andrea Bartolini, and Milos Puzovic. Energy and Power Aware Job Scheduling and Resource Management: Global Survey — Initial Analysis. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 685–693. DOI: [10.1109/IPDPSW.2018.00111](https://doi.org/10.1109/IPDPSW.2018.00111).
- [54] Olli Mämmelä, Mikko Majanen, Robert Basmadjian, Hermann De Meer, André Giesler, and Willi Homberg. Energy-aware job scheduler for high-performance computing. In *Computer Science Research and Development 27* (2012), pp. 265–275. DOI: [10.1007/s00450-011-0189-6](https://doi.org/10.1007/s00450-011-0189-6).
- [55] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. A Run-Time System for Power-Constrained HPC Applications. In *High Performance Computing*. Ed. by Julian M. Kunkel and Thomas Ludwig. Springer International Publishing, 2015, pp. 394–408. ISBN: 978-3-319-20119-1.
- [56] Aniruddha Marathe, Yijia Zhang, Grayson Blanks, Nirmal Kumbhare, Ghaleb Abdulla, and Barry Rountree. An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*. E2SC’17. Denver, CO, USA: Association for Computing Machinery, 2017. ISBN: 9781450351324. DOI: [10.1145/3149412.3149421](https://doi.org/10.1145/3149412.3149421).

- [57] Aniruddha Marathe, Yijia Zhang, Grayson Blanks, Nirmal Kumbhare, Ghaleb Abdulla, and Barry Rountree. An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*. E2SC'17. Denver, CO, USA: Association for Computing Machinery, 2017. ISBN: 9781450351324. DOI: [10.1145/3149412.3149421](https://doi.org/10.1145/3149412.3149421).
- [58] Sandro M. Marques, Thiarles S. Medeiros, Fábio D. Rossi, Marcelo C. Luizelli, Antonio Carlos S. Beck, and Arthur F. Lorenzon. Synergically Rebalancing Parallel Execution via DCT and Turbo Boosting. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 277–282. DOI: [10.1109/DAC18074.2021.9586201](https://doi.org/10.1109/DAC18074.2021.9586201).
- [59] MGHPCC. <https://www.mghpcc.org/>. 2021.
- [60] nasa.gov. NAS Parallel Benchmarks. <https://www.nas.nasa.gov/software/npb.html>. Jan. 2022.
- [61] NVIDIA. NVML API Reference Manual, vR450. June 2020. URL: <https://docs.nvidia.com/develop/nvml-api/index.html>.
- [62] Optimizing Performance with Intel® Advanced Vector Extensions. <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf>. Sept. 2014.
- [63] Michael Ott, Woong Shin, Norman Bourassa, Torsten Wilde, Stefan Ceballos, Melissa Romanus, and Natalie Bates. Global Experiences with HPC Operational Data Measurement, Collection and Analysis. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 2020, pp. 499–508. DOI: [10.1109/CLUSTER49012.2020.00071](https://doi.org/10.1109/CLUSTER49012.2020.00071).
- [64] Ali Pahlevan, Marina Zapater, Ayse K. Coskun, and David Atienza. ECOGreen: Electricity Cost Optimization for Green Datacenters in Emerging Power Markets. In *IEEE Transactions on Sustainable Computing* 6.2 (2021), pp. 289–305. DOI: [10.1109/TSUSC.2020.2983571](https://doi.org/10.1109/TSUSC.2020.2983571).
- [65] Srinivas Pandravadu. Intel(R) Speed Select Technology User Guide. <https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html>. 2020.
- [66] Tirthak Patel, Adam Wagenhäuser, Christopher Eibel, Timo Hönig, Thomas Zeiser, and Devesh Tiwari. What does power consumption behavior of hpc jobs reveal?: Demystifying, quantifying, and predicting power consumption characteristics. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. New Orleans, LA, USA, 2020, pp. 799–809.

- [67] Tapasya Patki, Natalie Bates, Girish Ghatikar, Anders Clausen, Sonja Klingert, Ghaleb Abdulla, and Mehdi Sheikhalishahi. Supercomputing Centers and Electricity Service Providers: A Geographically Distributed Perspective on Demand Management in Europe and the United States. In *High Performance Computing*. Ed. by Julian M. Kunkel, Pavan Balaji, and Jack Dongarra. Cham: Springer International Publishing, 2016, pp. 243–260.
- [68] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS '13. Eugene, Oregon, USA: Association for Computing Machinery, 2013, pp. 173–182. ISBN: 9781450321303. DOI: [10.1145/2464996.2465009](https://doi.org/10.1145/2464996.2465009).
- [69] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '15. Portland, Oregon, USA: Association for Computing Machinery, 2015, pp. 121–132. ISBN: 9781450335508. DOI: [10.1145/2749246.2749262](https://doi.org/10.1145/2749246.2749262).
- [70] Oliver Peckham. At SC22, Carbon Emissions and Energy Costs Eclipsed Hardware Efficiency. In *HPCwire* (Dec. 2, 2022). URL: <https://www.hpcwire.com/2022/12/02/at-sc22-carbon-emissions-and-energy-costs-eclipsed-hardware-efficiency/> (visited on 07/11/2023).
- [71] Kevin Pedretti, Ryan E. Grant, James H. Laros III, Michael Levenhagen, Stephen L. Olivier, Lee Ward, and Andrew J. Younge. A Comparison of Power Management Mechanisms: P-States vs. Node-Level Power Cap Control. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 725–729. DOI: [10.1109/IPDPSW.2018.00117](https://doi.org/10.1109/IPDPSW.2018.00117).
- [72] PJM Manual 12: Balancing Operations. <https://pjm.com/~media/documents/manuals/m12.ashx>. June 2022.
- [73] Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. Carbon-aware computing for datacenters. In *IEEE Transactions on Power Systems* 38.2 (2022), pp. 1270–1280.
- [74] Power Shifting Ratio. IBM Power9 Overview. URL: <https://variorum.readthedocs.io/en/latest/IBM.html>.
- [75] Renewable Energy Explained. <https://www.eia.gov/energyexplained/renewable-sources/portfolio-standards.php>. June 2021.

- [76] Barry Rountree, David K. Lowenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing*. ICS '09. Yorktown Heights, NY, USA: Association for Computing Machinery, 2009, pp. 460–469. ISBN: 9781605584980. DOI: [10.1145/1542275.1542340](https://doi.org/10.1145/1542275.1542340).
- [77] Issa Saba, Eishi Arima, Dai Liu, and Martin Schulz. Orchestrated Co-scheduling, Resource Partitioning, and Power Capping on CPU-GPU Heterogeneous Systems via Machine Learning. In *Architecture of Computing Systems*. Ed. by Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck. Cham: Springer International Publishing, 2022, pp. 51–67. ISBN: 978-3-031-21867-5.
- [78] Théo Saillant, Jean-Christophe Weill, and Mathilde Mougeot. Predicting job power consumption based on rjms submission data in hpc systems. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35*. Springer. Frankfurt/Main, Germany: Springer, Cham, 2020, pp. 63–82.
- [79] Varun Sakalkar, Vasileios Kontorinis, David Landhuis, Shaohong Li, Darren De Ronde, Thomas Blooming, Anand Ramesh, James Kennedy, Christopher Malone, Jimmy Clidas, et al. Data center power oversubscription with a medium voltage power plane and priority-aware capping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 497–511.
- [80] Ryuichi Sakamoto, Thang Cao, Masaaki Kondo, Koji Inoue, Masatsugu Ueda, Tapasya Patki, Daniel Ellsworth, Barry Rountree, and Martin Schulz. Production Hardware Overprovisioning: Real-World Performance Optimization Using an Extensible Power-Aware Resource Management Framework. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2017, pp. 957–966. DOI: [10.1109/IPDPS.2017.107](https://doi.org/10.1109/IPDPS.2017.107).
- [81] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J. Austin Ellis, and Feiyi Wang. Revealing Power, Energy and Thermal Dynamics of a 200PF Pre-Exascale Supercomputer. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–18. DOI: [10.1145/3458817.3476188](https://doi.org/10.1145/3458817.3476188).
- [82] Slurm Power Management Guide. https://slurm.schedmd.com/power_mgmt.html. [Online; accessed 2022-06-13]. Mar. 2018.
- [83] Michael B. Taylor. Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse. In *DAC Design Automation Conference 2012*. 2012, pp. 1131–1136.

- [84] Aidan P. Thompson, H. Metin Aktulga, Richard Berger, Dan S. Bolintineanu, W. Michael Brown, Paul S. Crozier, Pieter J. in 't Veld, Axel Kohlmeyer, Stan G. Moore, Trung Dac Nguyen, Ray Shan, Mark J. Stevens, Julien Tranchida, Christian Trott, and Steven J. Plimpton. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. In *Computer Physics Communications* 271 (2022), p. 108171. DOI: [10.1016/j.cpc.2021.108171](https://doi.org/10.1016/j.cpc.2021.108171).
- [85] TOP500 List - June 2023. <https://www.top500.org/lists/top500/list/2023/06/>. June 2023.
- [86] Paul Veitch, John J Browne, and Chris MacNamara. Resource Tuning for Energy Efficient Slicing. In *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2021, pp. 100–104. DOI: [10.1109/ICIN51074.2021.9385531](https://doi.org/10.1109/ICIN51074.2021.9385531).
- [87] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation Cores: Reducing the Energy of Mature Computations. In *SIGARCH Computer Architecture News* 38.1 (Mar. 2010), pp. 205–218. ISSN: 0163-5964. DOI: [10.1145/1735970.1736044](https://doi.org/10.1145/1735970.1736044).
- [88] Jons-Tobias Wamhoff, Stephan Diestelhorst, Christof Fetzer, Patrick Marlier, Pascal Felber, and Dave Dice. The TURBO Diaries: Application-controlled Frequency Scaling Explained. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 193–204. ISBN: 978-1-931971-10-2.
- [89] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. In *Communications of the ACM* 52.4 (Apr. 2009), pp. 65–76. ISSN: 0001-0782. DOI: [10.1145/1498765.1498785](https://doi.org/10.1145/1498765.1498785).
- [90] Daniel C. Wilson, Siddhartha Jana, Aniruddha Marathe, Stephanie Brink, Christopher M. Cantalupo, Diana R. Guttman, Brad Geltz, Lowren H. Lawson, Asma H. Al-rawi, Ali Mohammad, Fuat Keceli, Federico Ardanaz, Jonathan M. Eastep, and Ayse K. Coskun. Introducing Application Awareness Into a Unified Power Management Stack. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021, pp. 320–329. DOI: [10.1109/IPDPS49936.2021.00040](https://doi.org/10.1109/IPDPS49936.2021.00040).
- [91] Daniel C. Wilson, Ioannis Ch. Paschalidis, and Ayse K. Coskun. Site-Wide HPC Data Center Demand Response. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 2022, pp. 1–7. DOI: [10.1109/HPEC55821.2022.9926322](https://doi.org/10.1109/HPEC55821.2022.9926322).
- [92] Daniel C. Wilson, Asma H. Al-rawi, Lowren H. Lawson, Siddhartha Jana, Federico Ardanaz, Jonathan M. Eastep, and Ayse K. Coskun. Guiding Hardware-Driven Turbo with Application Performance Awareness. In *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*. 2022, pp. 1–8. DOI: [10.1109/IGSC55832.2022.9969356](https://doi.org/10.1109/IGSC55832.2022.9969356).

- [93] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook’s Data Center-Wide Power Management System. In *SIGARCH Computer Architecture News* 44.3 (June 2016), pp. 469–480. ISSN: 0163-5964. DOI: [10.1145/3007787.3001187](https://doi.org/10.1145/3007787.3001187).
- [94] Xingfu Wu, Aniruddha Marathe, Siddhartha Jana, Ondrej Vysocky, Jophin John, Andrea Bartolini, Lubomir Riha, Michael Gerndt, Valerie Taylor, and Sridutt Bhattachandra. Toward an End-to-End Auto-tuning Framework in HPC PowerStack. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 2020, pp. 473–483. DOI: [10.1109/CLUSTER49012.2020.00068](https://doi.org/10.1109/CLUSTER49012.2020.00068).
- [95] Huazhe Zhang and Henry Hoffmann. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’16. Atlanta, Georgia, USA: Association for Computing Machinery, 2016, pp. 545–559. ISBN: 9781450340915. DOI: [10.1145/2872362.2872375](https://doi.org/10.1145/2872362.2872375).
- [96] Yijia Zhang, Daniel C. Wilson, Ioannis Ch. Paschalidis, and Ayse K. Coskun. A Data Center Demand Response Policy for Real-World Workload Scenarios in HPC. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, France: IEEE, 2021, pp. 282–287. DOI: [10.23919/DATE51398.2021.9474075](https://doi.org/10.23919/DATE51398.2021.9474075).
- [97] Yijia Zhang, Daniel C. Wilson, Ioannis Ch. Paschalidis, and Ayse K. Coskun. HPC Data Center Participation in Demand Response: An Adaptive Policy With QoS Assurance. In *IEEE Transactions on Sustainable Computing* 7.1 (2022), pp. 157–171. DOI: [10.1109/TSUSC.2021.3077254](https://doi.org/10.1109/TSUSC.2021.3077254).
- [98] Jiajia Zheng, Andrew A Chien, and Sangwon Suh. Mitigating curtailment and carbon emissions through load migration between data centers. In *Joule* 4.10 (2020), pp. 2208–2222.

CURRICULUM VITAE

