Summer 2017

Fast demand response with datacenter loads: a green dimension of big data

Josiah McClurg
University of Iowa

FAST DEMAND RESPONSE WITH DATACENTER LOADS: A GREEN DIMENSION OF BIG DATA

by

Josiah McClurg

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

August 2017

Thesis Supervisor: Associate Professor Raghuraman Mudumbai

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Josiah McClurg

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Electrical and Computer Engineering at the August 2017 graduation.

Thesis Committee: _____
Raghuraman Mudumbai, Thesis Supervisor

_____
Soura Dasgupta

_____
Jon G. Kuhl

_____
Guadalupe M. Canahuate

_____
Suely Oliveira

ACKNOWLEDGEMENTS

I'll always be grateful: To my kind and insatiably-inquisitive advisor Raghu, for believing in me and inspiring a deep love of learning that I hope I never lose. To my friends Gregg and Abhay, who haven't left me behind even when I can't often make time to catch up. To my adopted family Jeff, Kelly, Jemmie, Ben, and Kate, for welcoming me into their lives and for reminding me just how much joy there is in openness and generosity. To my sisters Moriah and Micah, for continuing to teach me that putting people first is always the right choice. To my Dad and brother, for showing me how to not give up when the going gets tough. To my Mom, for her tireless love and for giving me my first and second chances at life. Last and most of all, to my wife Martha, for extracting strength and hope I never knew I had; for sticking by me in failure and success; and for loving me wholeheartedly, without pretense, and beyond all reason.

ABSTRACT

Demand response is one of the critical technologies necessary for allowing large-scale penetration of intermittent renewable energy sources in the electric grid. Data centers are especially attractive candidates for providing flexible, real-time demand response services to the grid because they are capable of fast power ramp-rates, large dynamic range, and finely-controllable power consumption. This thesis makes a contribution toward implementing load shaping with server clusters through a detailed experimental investigation of three broadly-applicable datacenter workload scenarios. We experimentally demonstrate the eminent feasibility of datacenter demand response with a distributed video transcoding application and a simple distributed power controller. We also show that while some software power capping interfaces performed better than others, all the interfaces we investigated had the high dynamic range and low power variance required to achieve high quality power tracking. Our next investigation presents an empirical performance evaluation of algorithms that replace arithmetic operations with low-level bit operations for power-aware Big Data processing. Specifically, we compare two different data structures in terms of execution time and power efficiency: (a) a baseline design using arrays, and (b) a design using bit-slice indexing (BSI) and distributed BSI arithmetic. Across three different datasets and three popular queries, we show that the bit-slicing queries consistently outperform the array algorithm in both power efficiency and execution time. In the context of datacenter power shaping, this performance optimization enables additional power

flexibility – achieving the same or greater performance than the baseline approach, even under power constraints. The investigation of read-optimized index queries leads up to an experimental investigation of the tradeoffs among power constraint, query freshness, and update aggregation size in a dynamic big data environment. We compare several update strategies, presenting a bitmap update optimization that allows improved performance over both a baseline approach and an existing state-of-the-art update strategy. Performing this investigation in the context of load shaping, we show that read-only range queries can be served without performance impact under power cap, and index updates can be tuned to provide a flexible base load. This thesis concludes with a brief discussion of control implementation and summary of our findings.

PUBLIC ABSTRACT

When resources are limited, most people benefit from "playing nice." The most effective kind of sharing is about more than politeness and good manners. It's about relationships and trust. Those things take a lot of effort and understanding – especially between people with opposing goals – but they're worth it in the end. The idea of datacenter demand response is to design a mutually-beneficial relationship between an unpredictable electricity user (load) and a cautious electricity supplier (utility). The challenge is a difficult one, but the benefits to both participants are worth it in the end.

Demand response is a way for large electric loads to "play nice" both with utilities and with other loads that can't control their own power consumption. Sharing strategies are beneficial to both loads and utilities because electric power is a fundamentally limited resource. This isn't just because most power plants run off of limited energy sources like coal and natural gas. At any given instant, loads can use no more power than is being generated at that moment. And, to avoid power surges, utilities have to pay money to "dump" any excess power that they generate. It's impossible to predict exactly how much power will be needed at any given moment, and renewable energy suppliers like wind and solar make it difficult for utilities to precisely control the amount of power they generate. If some loads help utilities out by temporarily raising or lowering their power consumption, the delicate balance between generation and load is easier to maintain and everyone receives cheaper, more reliable power.

Datacenter operators have become so good at providing transparent access to Internet applications, scientific computing, business analytics and other services, that it's easy to forget that "the cloud" is a collection of physical machines that need regular maintenance and lots of electrical power. Datacenters' large, quickly-changing, and unpredictable electric load currently makes it more difficult for utilities to provide reliable power. However, some of the very features that make datacenters such difficult customers can actually let datacenters provide demand response services that other controllable loads can't. In particular, datacenters are uniquely well-suited to provide *realtime* demand response services because of their ability to quickly change their power consumption across a wide range of power levels. Other types of demand response (peak capping, emergency service) can help with long-term reliability planning. However, realtime demand response is especially valuable in the fast-changing power grid of today, because it can respond quickly and continuously to unpredictable loads and renewable generation.

The benefit both to utilities and datacenters is very much worth the effort of making datacenter demand response a reality, and this thesis answers several questions related to that goal: As a way of showing that power shaping is actually practical, it presents experimental video transcoding service that also provides a high quality fast power shaping service to the electric grid. Next, we explore how power shaping affects the performance of two common datacenter workloads: Read-optimized queries (used to quickly explore large datasets which do not change very often), and update-aware indexing (used to speed up access to large, frequently-changing datasets such

as Twitter trending tweets). In these contexts, we show that certain performance optimizations can also have a positive impact on power requirements. This reduction in *required* power consumption gives clusters serving these workloads the flexibility they need to participate in load shaping programs. We finish up by mentioning some preliminary ideas on designing power controllers that are robust to the complex and quickly-changing environments of real-world datacenters.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

LIST OF ALGORITHMS

CHAPTER 1
DATACENTER DEMAND RESPONSE

1.1 Roadmap

This thesis uses experimental results from a power-aware cluster to show how the fast response time, large dynamic range, and fine-grained controllability, of computer servers can allow datacenters to provide flexible, large-scale, real-time demand response services to the grid. The power characteristics of server clusters are highly-dependent on the workloads served by those clusters. So, as a step toward implementing load-shaping services on a larger scale, we analyze the power-performance tradeoffs of several important datacenter workloads including video transcoding and two classes of Big Data analytics workloads. We also present some preliminary results on a controller implementation.

1.1.1 Why Datacenter Demand Response?

Data centers are an important and ever-increasing portion of the electric load in the US electric power grid [1]. However, in their current configuration, datacenters are not particularly friendly to the grid. The largest of them can require hundreds of megawatts guaranteed capacity [2], but frequently experience large, unpredictable fluctuations in actual power consumption [3].

Demand response in datacenters has been proposed [4] as a way to can turn this challenge into an opportunity. Instead of paying a premium for colocated generation or other reliability services *from* utilities, datacenter operators may soon be able to

offer reliability services *to* the electric grid through realtime demand response.

At the same time, datacenters also have a significant amount of *excess capacity*. In other words, there is considerable evidence (summarized in Section 1.2) that shows that datacenters operate at substantially less than peak utilization most of the time. This suggests the possibility that the "spare capacity" of computer servers can be used to provide load-shaping services to the electric grid and indeed this idea has been proposed [5], and is being actively studied by researchers [4, 6–13].

This thesis represents a new contribution to the datacenter demand response literature by specifically considering *real-time* demand response at much shorter time-scales than previous work in this area. Compared to other types of electric loads, computer servers are especially well-suited to provide real-time services because their power consumption is controllable to (a) a fine granularity over (b) a large dynamic range of power levels with (c) fast ramp rates, and back up these arguments with detailed experimental evidence. A video transcoding application in Chapter 2 demonstrates that highly-parallelizable workloads with flexible performance constraints can be adapted to high quality power shaping with a simple controller.

1.1.2 Scope of Investigation

When workloads are more complex, the challenge of realtime datacenter power control must be considered jointly with the performance impact of this power shaping. To that end, we investigate common workloads within two important "Big Data" architectures: Bitmap Indexing and Update-aware Indexing. These two classes are

important because the past few years have seen an explosion in stored data volumes [14] and have produced a major change in how people and computing systems interact with data. Terabyte-sized datasets are now common in earth, space and life sciences [15], finance, and security and law enforcement [16]. At the same time, the decades-long trend of processors becoming faster (Moore's law) [17] appears to be coming to a close [18]. This fact and several other factors have led to an increasing dependence on cloud services employing distributed algorithms over server clusters in large-scale datacenters [19].

Optimizing the performance of Big Data algorithms is a problem of obvious importance and is being investigated on many fronts. However, the *computational* performance can no longer be considered in isolation from its effects on *power consumption* which accounts for a large fraction of the costs [20] of operating large datacenters. Much previous literature (summarized in Section 1.2) has considered the power impact of computational workloads [7,21–28]. The majority of previous work differs from our contribution in two ways. First, where other work focuses on ways to improve the energy performance and resource utilization of algorithms through top-down architectural optimizations and scheduling, our experimental investigations investigate low-level software power control features and direct algorithmic power/performance tradeoffs without power-aware scheduling. Second, our work does not only focus on reducing energy or power, but specifically investigates the potential for power shaping over time.

1.1.3 Thesis Outline

The rest of this thesis is organized as follows. The remainder of Chapter 1 provides a more detailed motivation for this work and provides a brief overview of the relevant previous literature. Specifically, we discuss: (1) why fast demand response is needed, why datacenters are attractive in this area and how this thesis makes a contribution to the demand response literature, (2) which kinds of datacenter workloads are amenable to power shaping, (3) how this thesis fills a gap in the power-aware computing research, and finally (4) the experimental testbed cluster which is common to the remaining chapters.

The next three chapters summarize three different experimental investigations of power shaping in a datacenter context. Each investigation studies a specific datacenter application with high power requirements, and these applications are chosen to be representative of a larger class of relevant workloads. The chapters are arranged such that the application space becomes more specific. In particular, we introduce the concept and general framework for datacenter power shaping in the context of general-purpose batch computing workloads and a sample video transcoding application. We follow this with an in-depth study of the tradeoffs that power capping brings out in latency-sensitive "big data" queries. We finish by investigating load shaping in the context of a latency-sensitive "streaming big data" application.

The purpose of Chapter 2 is to demonstrate the possibility of doing realtime power tracking in a realistic datacenter context. In this chapter, we (1) show that a simple controller around a standard video transcoding application can achieve high-

quality power tracking on a timescale of seconds; (2) experimentally demonstrate the fast ramp rate, large dynamic range, and finely-controllable power consumption of server clusters; and (3) present a detailed experimental investigation of the power characteristics associated with different software-based power capping interfaces. Included in this investigation is a practical discussion of these interfaces and their advantages and limitations for realtime demand response.

In the interest of simplicity, Chapter 2 makes the implicit assumptions of constant CPU load, few job dependencies between servers, and software quality of service metrics that apply only on timescales much longer than the timescales on which demand response is performed. Together, Chapters 3 and 4 investigate what happens when these three assumptions do not apply. Specifically, we demonstrate that with the appropriate choice of algorithms, time-sensitive service guarantees can be met under power cap, and investigate load shaping using more complex workloads (time-varying computational load and possible inter-server dependencies).

With the goal of better understanding the performance impact of load shaping, Chapter 3 investigates the power-performance tradeoffs of a big data query optimization called bit-sliced indexing, compared to a baseline approach. Even though this particular optimization was not designed with power in mind, we show that it achieves better or higher performance than the baseline approach, with lower peak power requirements.

In the context of load shaping, Chapter 3 is focused on meeting QoS guarantees under power cap. The assumption in this chapter is that load shaping will

be achieved in part due to some unrelated always-available, always-deferrable "base" workload that is executing in parallel with the queries. Chapter 4 investigates an application where deferring the "base" workload can impact performance. Specifically we investigate the power-performance tradeoffs of update-aware bitmap indexing. In this application, we show that high-priority read queries do not suffer latency penalty under power cap and the lower-priority updates provide a flexible base workload which can be adjusted for power shaping. We show that while power capping does impact query freshness, the impact is manageable. In addition to the power-performance tradeoffs of state-of-the-art update-aware indexing strategies, we present an index update optimization that achieves improved performance over the current state of the art, especially under multi-row updates.

Chapter 5 concludes with a brief discussion and some preliminary results related to the control aspects of real-time power shaping, along with open research questions for further study.

1.2 Motivation and Context of Realtime Demand Response

The idea of using load shaping to increase power grid reliability has been around since Edison's time [29]. However, grid operators have traditionally not emphasized load-side management, relying instead on the paradigm of providing energy on demand to completely *passive* consumers who have essentially unrestricted ability to vary their usage over time.

Indeed, active load-side management was arguably redundant in the tradi-

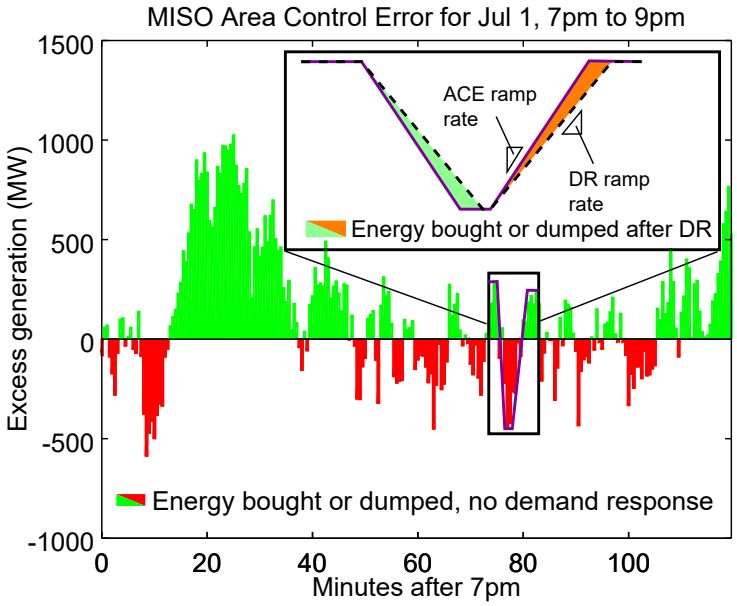


Figure 1.1: MISO Area Control Error

tional power grid. Grid operators have long been aware of the high degree of statistical regularity in electricity demand over time on daily, weekly and yearly time-scales and were traditionally able to take advantage of these patterns to predict loads and optimize generation schedules accordingly [30].

This situation has changed dramatically with the increasing penetration of intermittent renewables like wind and solar in the grid; renewable energy generation has proved to be far less predictable than load [31], and treating renewables as "negative loads" is not only expensive and wasteful [32] in many ways, it also strains the existing reliability and stability mechanisms of the grid [33]. This has led to a growing recognition that large-scale energy storage and/or "virtual storage" [5,34] in the form of demand response are essential to accommodate renewable energy sources in the grid.

A simple case study illustrates how realtime demand response can provide a benefit over slower demand response. Fig. 1.1, plots the area control error (ACE) of the Midcontinent Independent System Operator (MISO) over a two hour period [35]. During this period, 460 MWh of energy is dumped and 200 MWh of unscheduled energy purchases are made. Simulations show that using 200 MW demand response capacity with a ramp rate of 10 MW/min reduces the dumped energy by a mere 70 MWh. Increasing the ramp rate to 100 MW/min reduces the dumped energy by 190 MWh. And, similar results hold for the purchased energy reductions.

1.2.1 Real-Time Demand Response

The idea of using load shaping to increase power grid reliability has been around since Edison's time [29]. However, grid operators have traditionally not emphasized load-side management. Instead they have provisioned resources according to the paradigm of providing energy on demand to completely *passive* consumers who can vary their usage over time without restriction.

For many years, this paradigm was appropriate – in part because active load-side management was arguably redundant in the face of load forecasting. Grid operators have long been aware of the high degrees of statistical regularity in the variation of electricity demand over time on daily, weekly and yearly time-scales. Traditionally, electric power utilities have been able to take advantage of these patterns to predict loads and optimize generation schedules accordingly [30].

This situation changed dramatically with the increasing penetration of inter-

mittent renewables like wind and solar in the grid. First, renewable energy generation has proved to be far less predictable than load [31]. Second, treating renewables as "negative loads" is not only expensive and wasteful [32] in many ways, but it also strains the existing reliability and stability mechanisms of the grid [33]. These issues have led to a growing recognition that large-scale energy storage and/or "virtual storage" [5, 34] in the form of demand response are essential to accommodate renewable energy sources in the grid. Unfortunately, the electric grid cannot take full advantage of certain demand response services like frequency response or voltage regulation, due to a lack [5, 36, 37] of participating realtime loads.

Datacenters exhibit three features that make them especially attractive candidates for real-time demand response.

1. Fast power ramp rate
2. Large dynamic range
3. Finely-tunable power consumption (i.e. not on/off type loads)

Even so, the field of datacenter demand response is very young. As mentioned in a 2014 survey of datacenter demand response [4], the the majority of applicable research from previous years has been separated across disciplines and is not specifically targeted at datacenter demand response. Importantly, many of the existing power-aware scheduling algorithms operate on a timescale of hours or longer – which wastes the full flexibility of datacenters as demand response resources [12]. When the time scale of interest shrinks to seconds or less, as is the case with realtime demand response, it is clear that the challenge of realtime power shaping within a datacenter

is still very much an open problem [8]. The broad goal of our research is to help fill this gap.

1.2.2 Datacenter Demand Response

There has been significant recent interest in the idea of datacenters as a demand response resources [4, 6–13].

An empirical study by Lawrence Berkeley National Laboratory (LBNL) discussed the advantages and disadvantages of various datacenter demand response strategies [6]. The LBNL study was an important proof of concept that datacenter demand response was possible. Because we are interested in the fastest possible timescales, our work focuses more narrowly on realtime CPU throttling rather than on higher-order scheduling or cooling-system control.

The other existing work in datacenter demand response has been largely mathematical – focused on algorithms for optimizing price performance of different datacenter models in demand response markets. For example, the authors of [7] presented an economically-driven algorithm for distributed workload migration in a demand response context. The proposal [9] made an economic case for for datacenter demand response, providing a mathematical framework for optimizing datacenter workload against day-ahead dynamic pricing. At faster timescales, [11] presents simulation results and a control strategy for the joint optimization of datacenter and plug-in electric vehicles for frequency regulation. Similarly, [12] modeled a realtime demand response market with participating datacenter loads controlled via CPU throttling.

And, [8] studied joint fulfillment of probabilistic quality of service guarantees while performing regulation service to the grid and provided a partial implementation. Our work differs from existing research in that we are not focused on market participation, but on the details of actually implementing datacenter demand response. In particular, we provide measurements from a proof-of-concept cluster achieving realtime demand response with a simple distributed algorithm and we outline the software interfaces available to power-aware systems designers.

Our work is a contribution to the field in two important ways. First, we consider *realtime* demand response (i.e. controlling loads on short time-scales on the order of tens of seconds) rather than predictive methods (minute-ahead, etc). Second, we consider direct capping of server power consumptions, rather than indirectly managing power through the workload scheduler.

1.2.3 Relation to Power-Aware Computing

Power considerations are being increasingly recognized as economically crucial to the operation of computing systems of all scales ranging from mobile devices to super-computers. For large data centers that are typically used to provide Big Data processing services, this recognition has motivated the development of sophisticated methods [21,22] for managing and controlling the two major categories of power consumption: (a) the building maintenance and cooling systems, and (b) the computer servers themselves.

Power-aware computing [23] can be broadly defined as a class of hardware

and software design techniques that specifically account for power consumption in some way. It is important to note that designing with power in mind can be done at many different architectural levels (all the way from chip layout to high level software abstractions) and is not necessarily the same as designing to minimize power consumption. That is why we frame our discussion in terms of power, energy, and performance tradeoffs rather than simply exploring energy reductions.

For example, at the cluster scheduling level, there exists a growing literature on power shaping of servers in order to match renewable generation. For instance, [38] and [24] dispatch respectively batch workloads and MapReduce jobs to shape datacenter power to match predicted generation over the course of a day. Likewise, [7] discussed the geographical migration of virtual machines in response to hour-ahead locational marginal pricing. In these methods, the goal of power-aware design is not to *minimize* the total energy consumption, but rather to *move* intensive computations to a time or location that is most efficient. In this context, datacenter-scale workloads which have some flexibility in their power consumption are useful and attractive for more than just their processing power per watt.

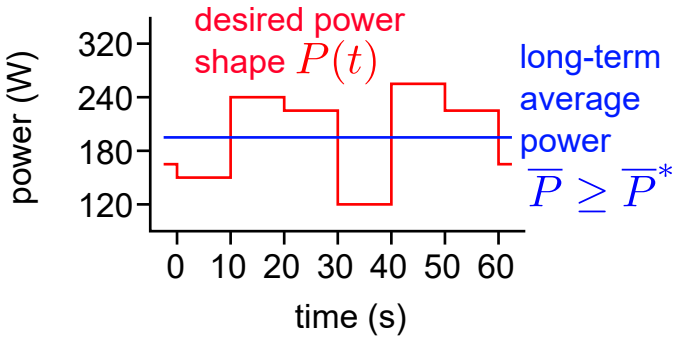
Similarly, power aware design can be done at the instruction scheduling level. For example, modern processors and operating systems are designed to dynamically scale voltage and frequency (DVFS) to achieve maximum performance-per-watt for a given workload. Other methods operate at the firmware level [26] to manage the temperature of realtime systems, or at the operating system [27] and hypervisor [28] levels.

Much of the previous work in power aware data management has focused on ways to improve the energy performance of existing systems through better query plans or other "top-down" optimizations [25] – leveraging better resource utilization to save energy without sacrificing performance, or providing ways to trade off among power, energy, and performance. One of the goals of this thesis is to take a "bottom-up" approach to power aware algorithm design. The basic idea is similar to the one proposed in [39], but applied at a cluster computing level: Bit slicing allows better resource utilization of the processor on a per-instruction basis. We investigate the cluster-level tradeoffs among power, energy, and performance of distributed bit sliced queries compared to baseline distributed queries.

1.3 Workload Applicability for Datacenter Demand Response

To understand which kinds of workloads are amenable to fast power shaping, it is important to have a basic working knowledge of how demand response agreements work in practice. There is some variation in how these contracts work, but operators that offer realtime demand response programs (PJM, for example) generally provide a power shaping signal (illustrated by $P(t)$ in Fig. 1.2) around an agreed-upon long term average power value $\overline{P}$, as illustrated in Fig. 1.2.

The demand response resource attempts to track this power shape as closely as possible, and there are typically penalties assessed for different measures of deviation from the setpoint.



minimum feasible long-term average power:

$$\overline{P}^* = \begin{cases} \min\limits_{c_0,c_1,\ldots,c_{n-1}} \left[\sum_{i=0}^{n-1} g_i c_i\right] + \sum_{i=0}^{n-1} b_i \\ \text{subject to } \hat{s}(w, c_0, c_1, \ldots, c_{n-1}) \geq s(w) \end{cases}$$

Figure 1.2: Example Demand Response Setpoint

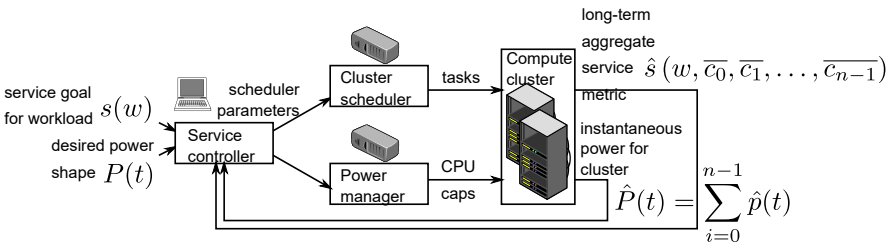


Figure 1.3: Example Datacenter Demand Response Application

1.3.1 Properties of Applicable Workloads

Now, consider the example scenario of Fig. 1.3, which depicts a cluster that a datacenter has allocated to participate in a demand response program. Because of the high development/operations cost associated with re-tooling the existing cloud management infrastructure, the cluster must work with an existing power-unaware scheduling framework.

Furthermore, since the power metric is calculated on a similar or faster timescale than the service metric, scheduling parameters alone are insufficient to meet the required power shaping guarantees. Rather than purchasing specialized hardware, the team must use existing realtime power management features – specifically, the servers' CPU thermal management infrastructure – to control the power shaping.

In short, the engineers in charge of the demand response compute team have the ability to

- select which workloads to admit to their cluster,
- set limited scheduler parameters such as job priorities, and
- set a "CPU utilization" cap – (such as maximum clock frequency, number of cores allocated, periodically-enforced processor time limits, etc).

As is shown in the simplified server power and performance model shown in Fig. 1.4, CPU load can always be throttled to effect a reduction in server power. However, an increase in power consumption requires pre-existing CPU demand. From this model, it is also clear that the workload assigned to each power-shaping server must include some performance flexibility.

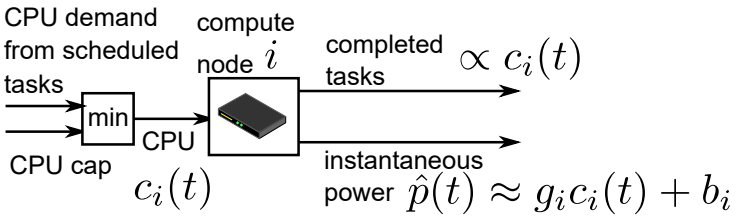


Figure 1.4: Compute Node Power and Performance Model

Demand-response-applicable workloads have the following properties:

- They are CPU-intensive workloads (CPU throttling reduces server power).
- They always have useful work to do (reduction in CPU throttle increases performance).
- Part of the workload is insensitive to small latency increases.

1.3.2 Workloads in this Thesis

In this thesis, we investigate three different workloads at different stages along the applicability spectrum.

The video transcoding workload represents a "low-hanging fruit," for participation in load shaping. Since many commercial transcoding services only offer best-effort QoS, the short timescale performance need not be considered. Cluster operators simply need only set the agreed-upon average power setpoint ($\overline{P}$, in Fig. 1.2) high enough to service their average demand. Chapter 2 shows that a very simple distributed controller can achieve industry-grade power tracking performance in this scenario.

In Chapter 3, we investigate are read-optimized big data queries. This work-

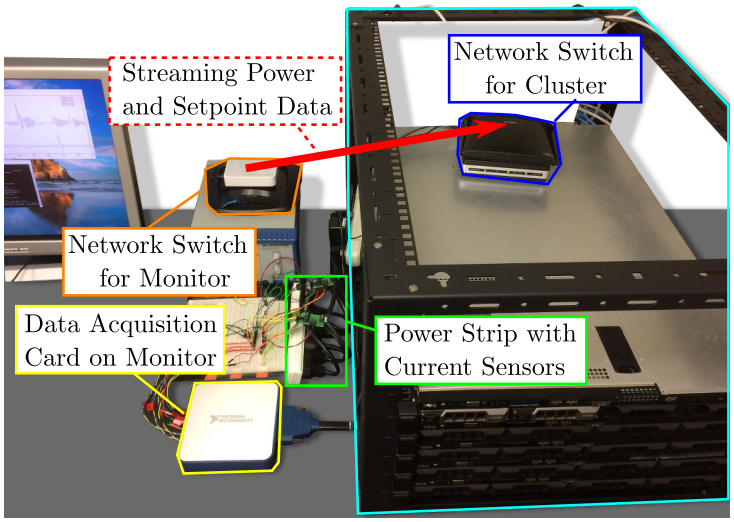


Figure 1.5: Experimental Setup

load has inter-server job dependencies and quality of service constraints on response time. An experimental evaluation of the performance tradeoffs of power capping shows that bit sliced index queries always outperformed the baseline approach, and also required less power to do so.

Chapter 4 investigates update-aware indexes, as a widely-applicable workload that also hits all the main aspects required for high-quality demand response: It is CPU-intensive, there is always benefit to having a fresher index, and in dynamic big data environments, there is not much penalty for having a slightly stale index.

1.4 Experimental Setup

In this section, we describe the experimental hardware and software setup of the server cluster which is common to many experiments throughout this thesis. The cluster is pictured in Fig. 1.5 and was chosen as a small-scale model that approximates

as closely as possible commonly-used server hardware configurations in datacenters [1, 40].

The cluster consists of four standard Dell PowerEdge R320 rack servers each powered by an Intel Xeon E5-2400 series processor with 6 cores and hyperthreading enabled. The servers all run 64-bit Ubuntu Server operating system version 15.04 with the 3.19.0-43-generic Linux kernel, and power management options are set to their default settings.

To obtain faster power measurements than are available from a commercial power distribution unit (PDU), we powered the servers through a specially instrumented power strip. A National Instruments PCIe-6323 data acquisition card measures the wall voltage and the output of amplified current shunts (Linear Technology LT1999 amplifier and a 0.02 Ω resistor, with combined nominal current measurement tolerance of $\pm 1.6\%$). A standard desktop PC serves as a monitor computer that reads five channels (one for rack voltage, and four for individual server currents) at 10 kS/s from the data acquisition card.

On the monitor computer, a software application polls rack-level power measurements from the data acquisition card, block averages these measurements at a configurable rate, and either saves the results locally to a file, or streams aggregated (total cluster power) measurements back to the cluster as power control feedback.

CHAPTER 2
POWER CHARACTERISTICS OF SIMPLE WORKLOADS

One of the primary goals of chapter is to introduce a general framework for datacenter power shaping and to explore the primary power metrics of interest in a realistic context. With this in mind, we begin by studying the power behavior of a testbed cluster running architecturally simple workloads. In particular, this chapter intentionally select a class of workloads that have relatively constant CPU load, few job dependencies between servers, and quality of service metrics that apply only on timescales longer than the power shaping timescale.

In order to make the point that datacenter demand response is eminently feasibly, we identify video transcoding as an important real-world application with minimal QoS constraints that also is highly CPU-intensive. The best-effort QoS constraints mitigate business concerns about load-shaping participation and the increased power costs from compute-heavy clusters further incentivizes load shaping. After showing that industry-grade power tracking can be achieved with minimal infrastructure changes and a simple distributed controller, we discuss the ubiquity of software interfaces for directly controlling server power consumption. Complementing the results of [41], we explore the tradeoffs of these software interfaces, and show that while some interfaces perform better than others, all interfaces investigated have low enough variability and high enough dynamic range do perform high quality fast load shaping.

2.1 A Simple Power-Tracking Experiment

Our first experiment (discussed in more detail in [42]) is intended to illustrate that very simple software methods can be effective in accurately controlling the real-time power consumption of servers under realistic conditions using an example video transcoding workload.

Using the experimental testbed cluster outlined in Section 1.4, we chose a 100 ms averaging window[1]. We streamed the power measurements at 10 S/s, and stream the power target only when it changes, and calculated the tracking error separately on each server. The result is functionally equivalent to streaming $e(t)$ to each server, as shown in Fig. 2.1 at 10 S/s. P_{set} in this experiment represents a desired total power setpoint for all the servers in the cluster combined and can be thought of as the signal that the electric grid sends to request demand response services from the cluster. The servers independently perform power shaping using only the total power tracking error signal $e(t)$ that is common to all servers.

In this experiment, the cluster is programmed to use the Linux avconv program to transcode video chunks from an NFS-mounted data store in a manner similar to the dynamic GOP transcoding scheme described in [43]. Each server has a worker which grabs the next available 10-minute block from the shared video source file, transcodes that block, and writes the result back to the shared drive, for later concatenation.

[1]We did not perform zero-crossing detection when choosing the borders for the block averaging operation on the instantaneous power. So, the measurements presented throughout this thesis have a higher variance than would have been obtained from averaging real power over an integer number of cycles. In other words, the tracking error measurements are somewhat conservative values.

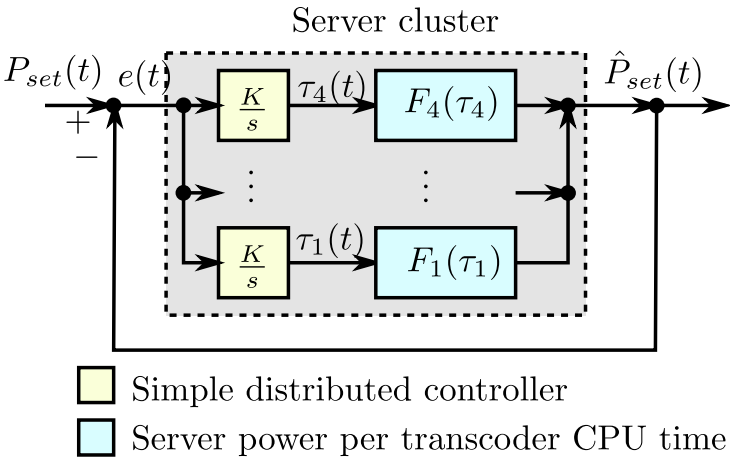


Figure 2.1: Block Diagram of Simple Control Scheme

The source files are chosen to be large enough that the transcoding takes longer than the duration of the power tracking test. We chose the highly parallelizable transcoding application because it is especially well-suited for power tracking using simple controllers running independently on each server. It is worth noting that some major commercial transcoding services (such as Amazon Elastic Transcoder [44]) offer pay-per-video pricing at a best-effort conversion speed that is similar to the application we use in this experiment. Furthermore, by replacing the simple controller in Fig. 2.1 with the one described in [45], we can achieve realtime power tracking with soft service level agreement (SLA) enforcement. But, that is outside of the scope of this thesis.

We assume that our realtime controllable load cluster has a demand response regulation service agreement in the PJM market. According to [46, 47], the cluster must respond to a signal $s(t) \in [0, 1]$ from the utility, so that its measured power consumption (measured in 10-second intervals) is close to P_{set} in Eq. (2.1) with D

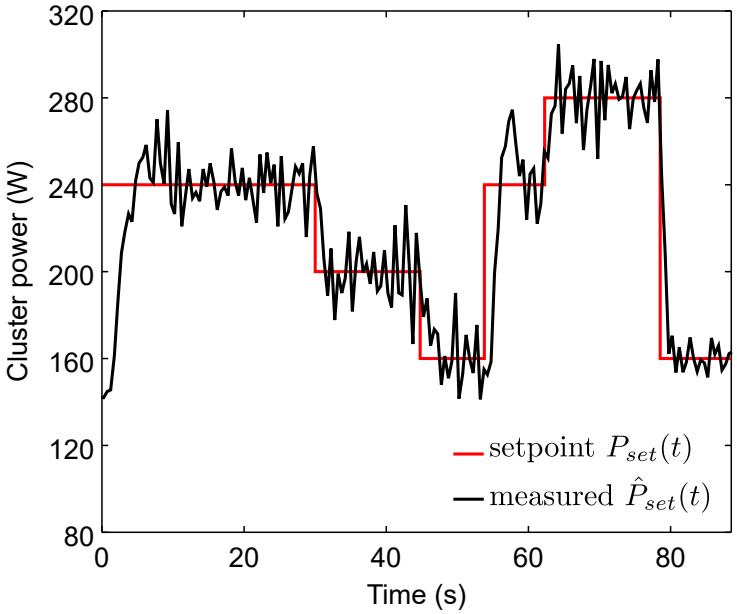


Figure 2.2: Tracking Target with Four Servers

and B being the dynamic range and base load agreed upon in the contract.

$$P_{set}(t) = Ds(t) + B \tag{2.1}$$

In this experiment, we choose that B to be cluster's power consumption with all the servers at idle, and D to be the maximum possible dynamic range. We choose $s(t)$ to vary in a piecewise-constant manner over time to show the cluster's closed-loop response to a step input.

In order to track the $P_{set}(t)$ calculated from Eq. (2.1), the controller on server i simply integrates the received tracking error $e(t)$, and uses a standard anti-windup procedure to produce a signal $\tau_i(t) \in [0, 1]$, which represents the fraction of the time that the CPU on server i is active.

We use the "userspace idle injection" interface described in Section 2.3 (essentially, using Linux signals to duty cycle the workload) to adjust τ_i whenever the servers receive an updated error signal measurement $e(t)$ (which is updated every 100 ms in our setup, as noted earlier). Adjustments to τ_i change server i's power consumption in a software and hardware-dependent manner, indicated by $F_i(\tau)$ in Fig. 2.1.

The power tracking experiment shown in Fig. 2.2 shows a maximum settling time of around 3 s and a root mean square tracking error (see Eq. (2.3)) of around 40 W for the whole cluster. Calculating the PJM-defined [46] "precision score," we get a value of 0.86 for this interval, which is well above the minimum value of 0.75. It should be noted that this tracking error was achieved with a very basic distributed controller without any communication between nodes – and might be expected to improve with more advanced control.

2.2 Exploring Server Power Characteristics

In the power tracking experiment of Section 2.1, our controller made no assumptions on the control plant. Datacenter power management literature often assumes that the power consumption of a computer server s follows the power model of Eq. (2.2) with $\tau \in [0, 1]$ being percent CPU time [48], and K_i, I_i constants.

$$F_i(\tau) = K_i\tau + I_i \tag{2.2}$$

2.2.1 Experiment to Test Accuracy of Power Model

We designed an experiment to test the accuracy of the model of Eq. (2.2) for our servers, under the "userspace idle injection" CPU-throttling interface we used in Section 2.1. For this purpose, we first measured the idle power I_i, and the dynamic range K_i for each of our four servers. After this, we ran the Linux stress program for 90 s for 100 evenly-spaced increments of τ ranging from 0 to 1, recording power measurements in 100 ms block averages. For each value of τ, we compared $F_i(\tau)$ (the server power predicted by Eq. (2.2)) with $\hat{F}_i(t,\tau)$, the measured server power consumption at time t. Using this information, we found the root mean square error (RMSE) according to Eq. (2.3) with $T = 100$ ms and $N = 900$.

$$\mathrm{RMSE}(\tau) = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} |\hat{F}_i(nT,\tau) - F_i(\tau)|^2} \tag{2.3}$$

Since RMSE compares the measured power with the predicted power at every sample, it gives a more conservative measure of the accuracy of Eq. (2.2), than simply comparing $F_i(\tau)$ with the average measured power $\hat{F}_i(\tau)$. Fig. 2.3 plots RMSE versus τ and shows that the maximum RMSE is around 3 W, meaning that Eq. (2.2) is a reasonably accurate model for the power consumption of a server using the user-space idle injection method to control the CPU duty cycle τ.

To put these measurements into context, we can calculate the worst-case PJM "precision score," using Eq. (2.2) as the setpoint, to see whether a datacenter could feasibly use an open-loop controller based on Eq. (2.2). In our measurements, the calculated minimum value of 0.95 was actually higher than the score we got for our

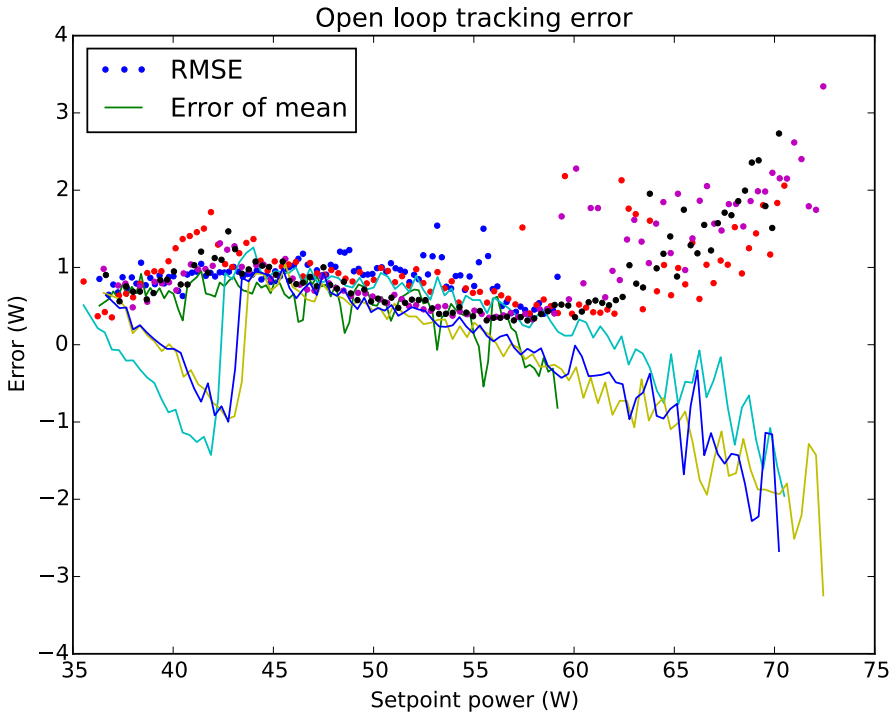


Figure 2.3: Minimum Tracking Error in Cluster

closed-loop controller in part because of the setpoint fluctuations in the previous section, and in part because the simple controller is not tuned to avoid overshoot. An open loop controller has obvious drawbacks, but it is encouraging to note that it is at least in theory feasible.

2.2.2 Ramp Rate and Dynamic Range Experiment

Our next experiment is designed to measure the maximum possible ramp rate of the cluster. Our strategy in this experiment was to signal each server in the cluster *at exactly the same instant* to transition from an idle CPU state to full load, thereby driving the entire cluster from the lowest to the highest power consumption state in as short a time interval as possible.

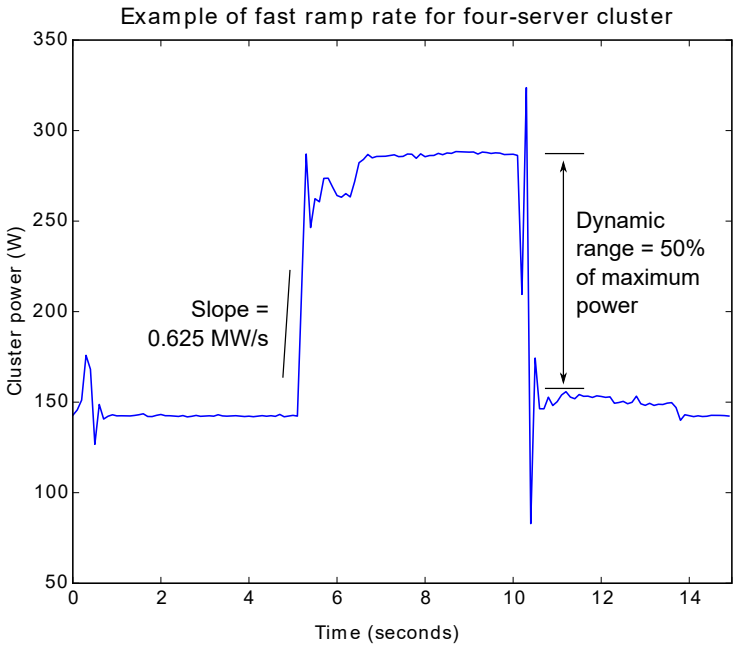


Figure 2.4: Fast Ramp Rate of Cluster

In order to achieve this synchronized transition, we wrote a simple remote procedure call (RPC) software to run on each server. This software responded to IP multicast requests to stop and start a computationally-intensive workload. Specifically, we chose the Linux stress [49] workload generator (performs repeated square root operations) for this experiment, but similar results were obtained with a variety of other computationally-intensive workloads (k-means clustering, financial markets simulation, and video transcoding are a few examples). As Fig. 2.4 shows, it is possible to obtain a fast power ramp rate of about 625 W/s, over a dynamic range of around 145 W, or 50% of the maximum power. To put this into context, consider scaling these results up to a 10 MW, 1.5 PUE datacenter with a fourth of its servers designated to participate in demand response. At this scale, the datacenter could

absorb almost a megawatt, at a rate of up to 4 MW/s.

2.3 Exploring Software Methods for Direct Power Control

In the experiments described in Sections 2.1 and 2.2, we used the userspace idle injection interface for throttling the CPU time. However, there are a number of different software methods for throttling CPU time, each with unique features and availability. Furthermore, each of these methods produces different statistics for the measured power $\hat{F}_i(\tau)$. As it turns out, some methods are more suited to power control than others. For example, the userspace idle injection interface we used earlier is more deterministic and more closely matches Eq. (2.2) than the other methods we investigated. Some previous work such as [50] have produced good experimental measurements in the context of power-aware metering, and previous studies like [51] have made contributions to the theoretical side of modeling server power. We complement these studies with a detailed experimental investigation that focuses on the specific software interfaces used to modulate the server power consumption.

We used measured power samples $\hat{F}_i(\tau)$, across the full range of $\tau \in [0, 1]$ to estimate some power statistics for each interface. Furthermore, to gain insight into *why* the power statistics look as they do, we also measured the percentage of time spent by the processor in each of its active and idle states (in the following, we refer to this information as "state residency") using hardware counters in a modified version of Intel's turbostat [52] utility.

2.3.1 Experiment to Compare Power Modulating Interfaces

In the experiments described in Sections 2.1 and 2.2, we used a "userspace idle injection" interface for controlling the CPU time. We now justify this choice of interface by presenting a detailed experimental comparison with a number of alternative software methods for controlling CPU utilization. Each of these methods have a different power characteristic $F_i(\tau)$. It turns out that the userspace idle injection interface we used earlier is more deterministic and more closely matches Eq. (2.2) compared to the other methods which makes it more suitable for our power tracking application.

Throughout the experiment, we recorded both the server power consumption (averaged in 500 ms blocks) and processor state residency information (again, collected in 500 ms blocks), and grouped those samples by τ. We removed outliers from each group using the median absolute deviation method [53]. And, for each sample group, we recorded the mean and a conservative estimate[2] of the sample range. We performed this experiment for each of the four servers in our cluster, and present results from one server for space considerations and because the power and state residency statistics were nearly identical across the four servers.

If the model of Eq. (2.2) were correct, plotting the mean of measured power $\hat{F}_i(\tau)$ against τ should produce a straight line with slope K_i (the dynamic range of the server), and y-intercept I_i (the idle power). And for many interfaces, this is close to what we observe, as is shown by the near-linear slope of Fig. 2.5.

[2]The observed range after outlier rejection, plus 3 standard deviations.

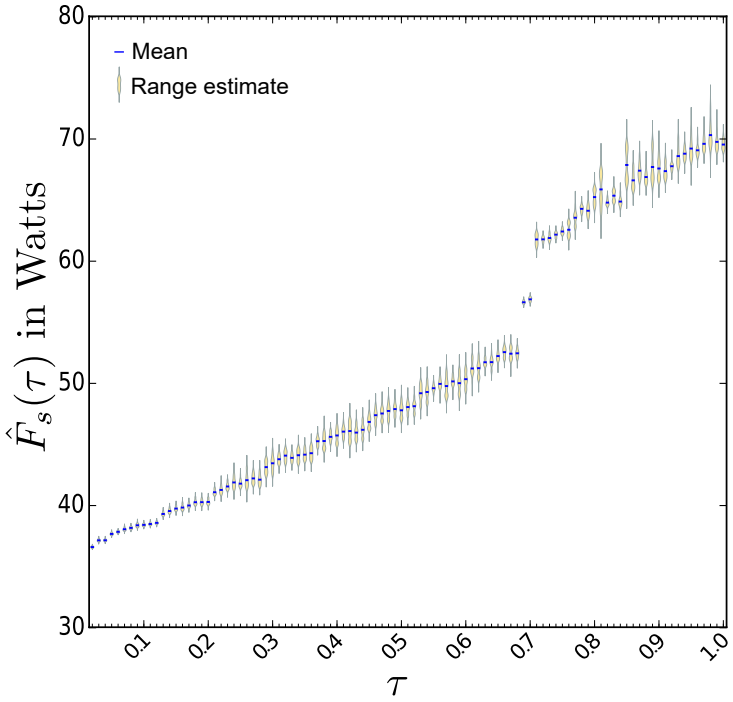


Figure 2.5: Power Profile of Server 4 under cgroups Interface

2.3.2 Linux cgroups Interface

The "user-space idle injection" interface that we used in our previous experiments is a custom implementation of a standard and more widely used CPU-capping framework called cgroups. We thus begin our discussion of the different power modulating interfaces by looking in detail at the Linux cgroup interface. Like many other interfaces, cgroups uses a general technique called idle cycle injection (ICI) to rapidly pause and resume processes in order to limit the average workload "seen" by the processor. Unlike direct dynamic voltage and frequency scaling (DVFS) methods which control processor state without modifying the computational workload, ICI can be used to access the full dynamic range of server power. Because ICI does not directly control processor state, the power consumption at a particular τ may have a larger

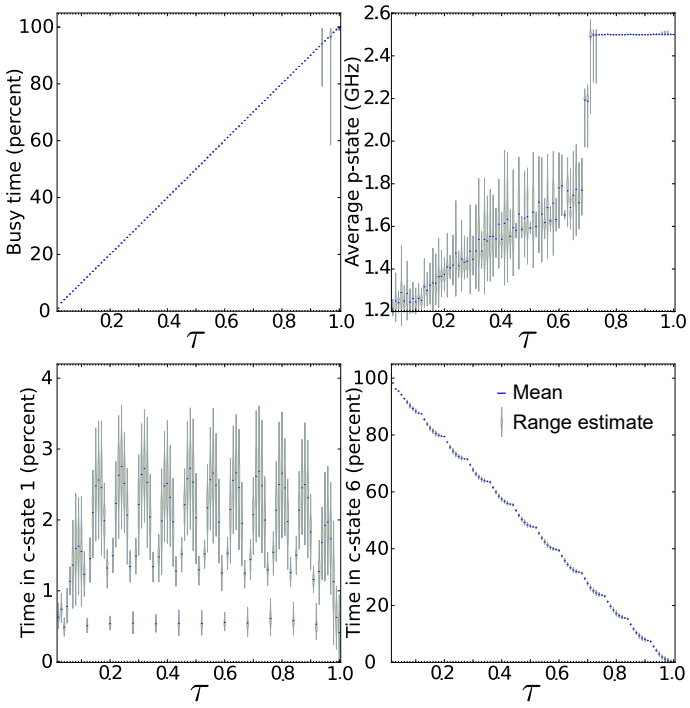


Figure 2.6: Residency Distribution of Server 4 under cgroups Interface

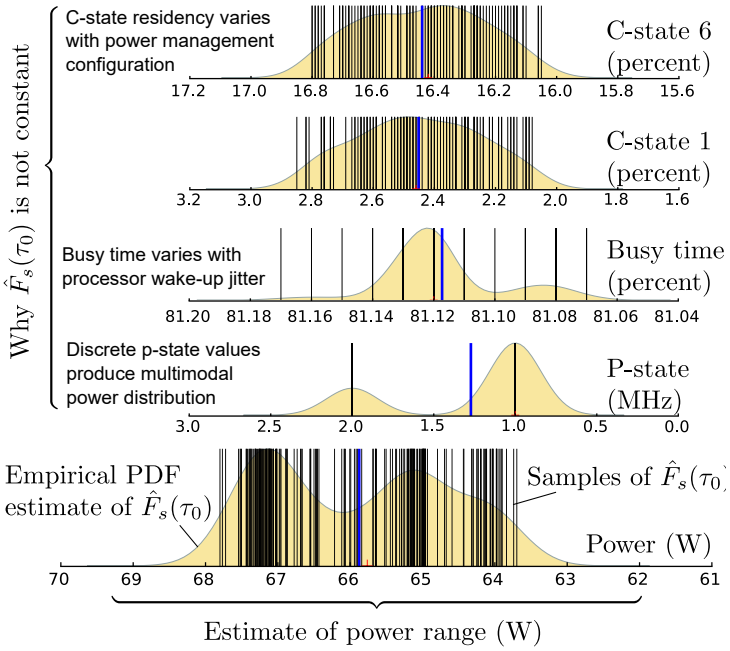


Figure 2.7: Detailed Look at Power and Processor State Samples for $\tau_0 = 0.81$

variance than a DVFS interface would produce. Because of this variance, ICI has been traditionally used in long-term average power control applications like thermal management [26,27] rather than direct power control. However, our results show that ICI can be quite effective at controlling server power consumption, even on short time scales.

To understand why $F_i(\tau)$ shows nondeterministic behavior, consider Fig. 2.7, which gives a detailed look at the observed processor behavior and measured power consumption for the cgroups interface at $\tau = 0.81$. Cgroup's default scheduler allows setting a CPU time cap within a periodic interval (set to 100 ms in our case). Once the workload exceeds its allotted time cap, the operating system pauses the workload processes, and power management features automatically transitions the server to one of its idle states. Not only is there some variation to which idle state gets entered, but there is some jitter in the processor's wake-up cycle – both of which work to producing a non-constant power consumption, even at a constant τ.

Fig. 2.6 gives a look at the processor behavior under the cgroups interface, across the full range of τ. At the top left, the "Busy time" graph has a unit slope, and a small estimated range for each τ – showing that the cgroups interface is effective at enforcing its CPU time cap. At the top right, the "Average p-state" graph gives a reason for the nonlinearity around $\tau = 0.7$ in the power graph of Fig. 2.5. At this τ, the power management system stops using the lower-power active modes of the processor, and starts running the processor at maximum speed whenever it is active. At the lower left and right, the "Time in c-state" graphs show that the power

management system under the cgroups interface prefers to let the processor idle in its lowest-power sleep mode (c-state 6).

Compared to the userspace idle injection interface we used in our previous experiments, one of the main advantages of the cgroups interface is that it has been built in to the Linux kernel since 2008, and is already in use for managing resources in several cluster computing environments (for example, YARN's LinuxContainerExecutor and MESOS's default containerizer). Furthermore, cgroups is designed to work on arbitrary groups of processes and therefore can throttle some groups of processes independently from others. The primary disadvantage is that this method is tied to Linux, while other methods may apply across many different operating system kernels.

2.3.3 Other Idle Cycle Injection Interfaces

Userspace Idle Injection

Inspired by the Linux cpulimit program, we designed a custom userspace tool to perform ICI using the Linux SIGSTOP and SIGCONT signals. We ran the same experiment as for the cgroups interface. This tool does the same thing as the cgroups interface, except that our tool operates in user space and throttles processes in a more synchronized manner. As shown in Fig. 2.10, this causes the power management system to essentially "duty cycle" the processor – toggling between sleep mode and the highest active state. Because the power management system decides to stop using low power active processor modes around $\tau = 0.2$ rather than cgroup's $\tau = 0.7$, the

$\hat{F}_i(\tau)$ for the userspace interface more closely matches $F_i(\tau)$ predicted by Eq. (2.2).

The power profile of this tool is the least variable and most linear of all ICI interfaces we tested. The advantage of such a tool is that it can be run in the background of any POSIX operating system, without accessing operating system or hypervisor power management policies. The disadvantage is that it is a custom tool, which only offers marginal improvements over the well-maintained and more full-featured cgroups interface.

Hypervisor CPU Capping

Various datacenter power management thesiss such as [28] have made use of hypervisor-based CPU throttling techniques. The Xen Project [54] provides a widely-deployed open source hypervisor, whose sched-credit framework uses ICI to arbitrate processor access among virtual machines (VMs). The current implementation uses a priority queue and an accounting thread to ensure that the virtual CPUs assigned to the VM do not exceed their CPU use cap for each 30-ms accounting interval.

In order to run a workload comparable with the other interfaces, we instantiated a paravirtualized linux guest VM allocated twelve virtual CPUs (two for each hyperthreaded physical core), and ran the same repeated-square-root workload as for the cgroups interface. Compared to the cgroups interface, the power variance at each interface setpoint is higher and Fig. 2.8 shows that the power profile achieves the full dynamic range but is highly nonlinear. Similar to the cgroups interface, the hypervisor-based interface has the advantage of being integrated into existing virtu-

alized environments and the disadvantage of being tied to those environments. Xen's sched-credit interface has the further inconvenience of a highly nonlinear setpoint-to-power characteristic.

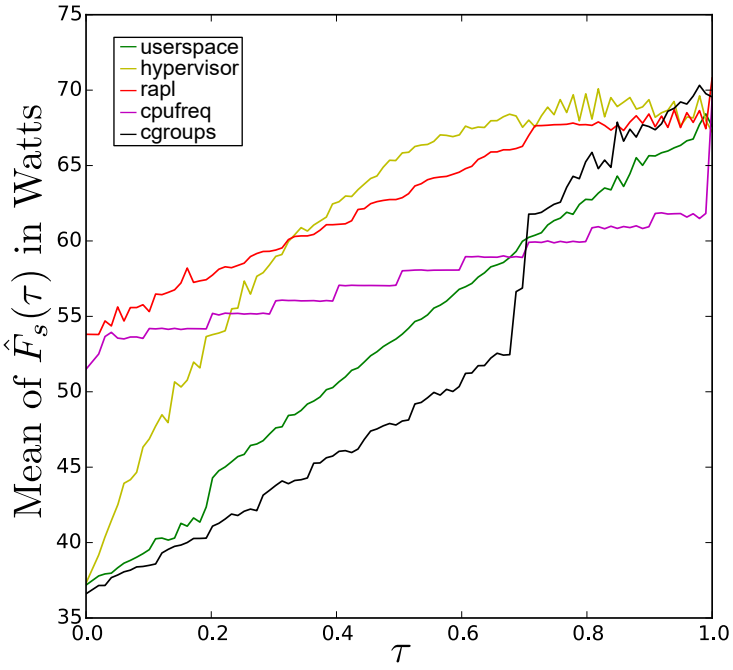


Figure 2.8: Comparison of Mean of $\hat{F}(\tau)$ for Server 4 under Different Power Modulating Interfaces

2.3.4 Direct Voltage and Frequency Scaling Interfaces

Rather than throttling workload and letting power management automatically transition processor state, dynamic voltage and frequency scaling (DVFS) interfaces communicate with the processor package directly in order control its operating state. As such, these interfaces offer much tighter control over the server power consumption. However, because they do not cause the processor to enter sleep states, DVFS

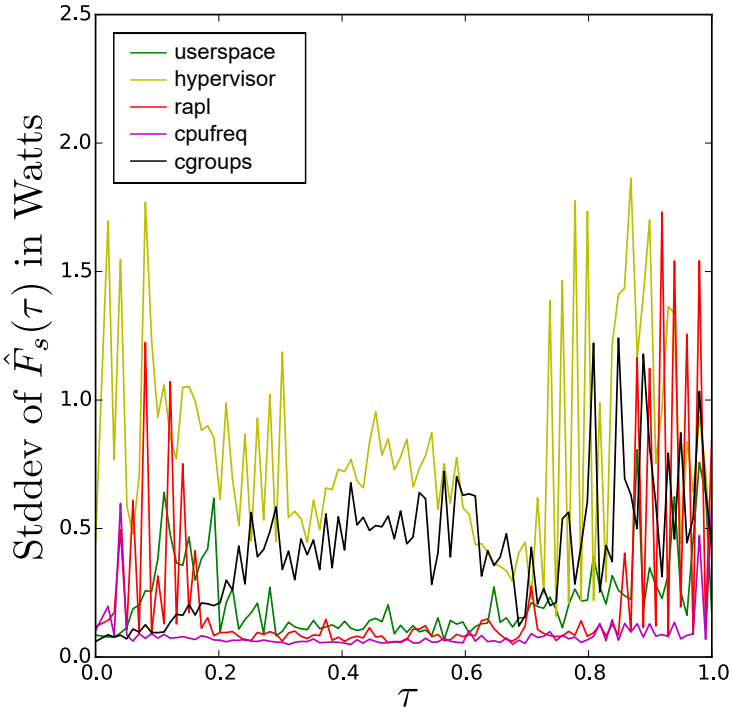


Figure 2.9: Comparison of Standard Deviation of $\hat{F}(\tau)$ for Server 4 under Different Power Modulating Interfaces

interfaces do not allow power control over the server's full dynamic range.

Cpufreq Driver

For example, several thesiss like [55,56] have successfully applied the userspace governor of the Linux acpi-cpufreq legacy driver to implement energy-aware DVFS on older processors. However, it is worth noting that most modern Intel processors only allow DVFS "hinting" rather than direct control. The more modern intel_pstate driver offloads power consumption to hardware and does not include a userspace governor. This technique has the advantage of a very low power variance at each setpoint, but only about half of the dynamic range is controllable. Further, as Fig. 2.10 shows, the cpufreq driver only allows twelve distinct voltage-frequency pairs, meaning that the

power characteristic of Fig. 2.8 has only twelve unique levels.

Proprietary On-chip Power-Limiting

The next DVFS technique is an improvement to the quantized cpufreq interface. RAPL precisely limits the processor package and the memory power consumption using a proprietary hardware mechanism, but the observed state residency (see Fig. 2.8) indicates that a DVFS technique is used. Certain Intel processors support a model-specific register (MSR) interface called Running Average Power Limit (RAPL) [57]. thesiss like [58] use the RAPL API to shape the power consumption of servers running transactional workloads to achieve high energy efficiency with minimal service-level objective violations. This technique again has very low power variance at each setpoint and covers almost the same dynamic range as the cpufreq interface. But, it is not limited to discrete levels of power consumption, as in the cpufreq interface.

2.4 Chapter Summary

The measurements in Section 2.3 showed that server power can be controlled with fast ramp rate, across a wide dynamic range, and with a high level of precision using several different power modulating software interfaces.

Out of all the power-modulating interfaces discussed in Section 2.3, the Linux cgroups interface stands out because of its low power variance, ubiquitous availability (almost all Linux servers have this feature), and highly flexible nature (allows different process groups to be managed separately). However, it is important to consider the

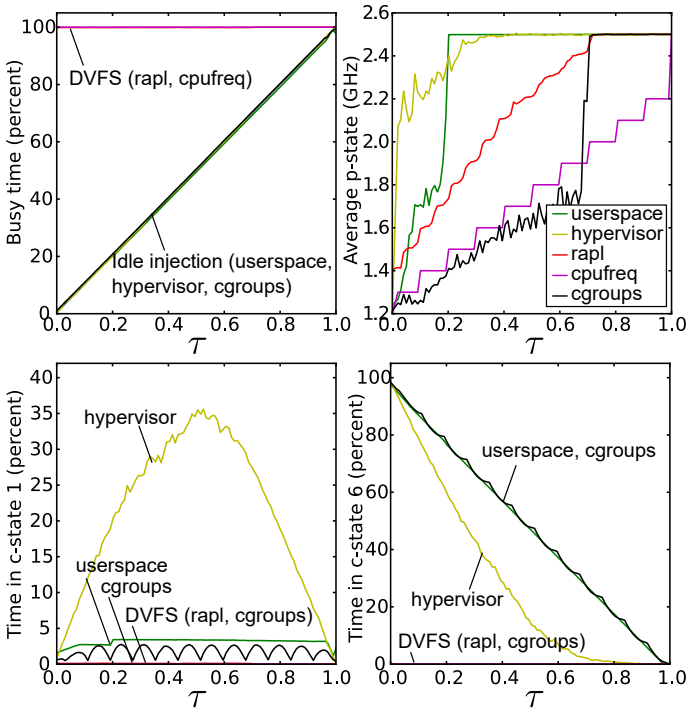


Figure 2.10: Comparison of Mean State Residency Profile for Server 4 under Different Power Modulating Interfaces

specific needs of that datacenter when selecting a power modulating interface for servers in a realtime datacenter demand response system. For example, Xen's sched-credit interface may be more applicable in a highly virtualized environment, or the RAPL interface may be more appropriate if precise controllability is valued over dynamic range. Even direct DVFS or custom software solutions may offer the best tradeoff for certain applications.

Furthermore, our experiments have shown that the linear power model of Eq. (2.2) commonly used in datacenter power management literature is not always valid, depending on the interface used. Server power $F_i(\tau)$, with CPU time τ controlled by Xen's sched-credit interface, for example, exhibits highly nonlinear depen-

dence on τ. The results of our comparison of the software power control interfaces can be summarized as follows.

- The linear model of Eq. (2.2) relating server power to CPU time holds reasonably well for the Linux cgroups, userspace idle injection, and RAPL interfaces, but is not a good model for the Xen sched-credit interface or the cpufreq driver interface.
- Idle Cycle Injection is very effective in providing a wide dynamic range but suffers from increased power variance.
- Direct Dynamic Voltage and Frequency Scaling is effective in accurate power control with low variance, but offers a limited dynamic range.

In this chapter, we have provided experimental evidence that datacenters are attractive controllable loads. And, as a first step toward understanding the details of implementing a controller for datacenter power shaping we have compared and contrasted several different classes of cpu-utilization capping software available to rack servers. Custom software used in this thesis is open source and can be downloaded from [59]. Our power tracking experiment showed that a very simple controller can exceed industry power tracking specifications. Further measurements showed a dynamic range of around 50% of the maximum power, and practically no upper limit on the power ramp rate.

CHAPTER 3
POWER TRADEOFFS OF READ-OPTIMIZED QUERIES

Where Chapter 2 introduced a general framework for load shaping and investigated power characteristics of the cluster under architecturally simple workloads, this chapter undertakes to experimentally investigate a scenario where (1) there may be job dependencies between servers and (2) quality of service metrics are calculated on timescales similar to the power shaping timescale. Rather than repeating the full "closed loop" power tracking experiment of Section 2.1, we undertake a detailed investigation of the power-performance tradeoffs of a latency-sensitive workload under power cap – implicitly assuming that the "base" CPU load required for load shaping can be supplied by some parallel workload whose deferral does not impact QoS. Chapter 4 takes this notion a step further by exploring a workload where the performance of high priority jobs depends indirectly on the low priority workload performance.

This chapter and Chapter 4 together demonstrate that several important latency-sensitive "big data" workloads are amenable to load shaping, given an appropriately-chosen query implementation. The application we have in mind in both of these chapters is a big data analytics engine which serves latency-sensitive queries on a distributed dataset. In this chapter, we discuss the performance impact of power capping on three different types of read-optimized query algorithms using a bit-sliced datastructure. We find that the power requirements for the optimized algorithms are significantly lower than a baseline approach, and have improved performance. Since the read latency of clusters using the bit-slicing optimization is scarcely impacted by

power cap, load shaping can be achieved in these clusters simply by the addition of a parallel deferrable "base" CPU load. Put differently, clusters which migrate from the baseline query approach to the bit-slicing optimization could gain enough flexibility in their power requirements to support load shaping *and* an additional low-priority workload.

In Chapter 4, we extend the discussion to an application with differentiated QoS requirements. In this case, we also assume a big data query service, but in an environment where the base data is frequently updated. We also make the assumption that the cluster is already using read-optimized bitmap index algorithms to serve their customers so there is no latency flexibility. Instead, as is common with many big-data applications, we assume there is some flexibility in query freshness and show that read latency is not compromised under power cap.

3.1 Performance Impact of Power Capping

As stated previously, our purpose in this chapter is to perform a *power-aware performance evaluation* of two different classes of Big Data indexing algorithms: (a) traditional distributed index requiring arithmetic operations such as multiplications and additions and (b) a distributed version of Bit-Sliced Indexing (BSI). The latter algorithms use only bit-wise operations and were designed with the expectation that they will execute faster on modern processors than algorithms that use traditional arithmetic operations. However, the effect of using bit-wise operations on power consumption is not a-priori obvious. On the one hand, one might expect that faster

algorithms would require less energy simply because the processor is used for a shorter period of time. On the other hand, modern hardware has power configurations that control the different states of the hardware e.g. frequency and voltage scaling, and many of these configurations deliver higher throughput at the expense of higher power consumption.

In this chapter, we explore the energy, power, and performance tradeoffs experimentally. Specifically, we consider three well-known queries that are recognized as fundamental to a large variety of Big Data applications: (a) aggregation [60], (b) top-k [61], and (c) nearest neighbor [62] queries. For each of these three queries, we conducted a series of experiments comparing traditional indexing algorithms that use arithmetic operations against alternative implementations that use only bit-wise operations. For these experiments, we use a Spark cluster that was instrumented to measure in real-time the consumption of individual servers as well as to impose configurable power limits on the processors of each server in the cluster.

Bitmap indices, first introduced in [63], are a way to represent arbitrary groupings of a set of records. Each bit position in the bitmap corresponds to a particular record, and the value of that bit marks the record's membership in the group. In the context of Big Data, bitmap indices are used to speed up queries on records with a particular property or attribute value-range. For categorical attributes, one bitmap vector is created for each attribute value. Continuous attributes are discretized into a set of ranges (or bins) and bitmaps are generated for each bin.

Figure 3.1 shows an example of a dataset whose two attributes can each take

on 3 distinct values (cardinality=3). One way to index this dataset is to generate one bitmap for each attribute value and set the bit is set if the tuple has that value. In this way, a query asking for tuples with Attributes 1 amd 2 equal 1 can be answered by simply ANDing the corresponding bitmap vectors together. The resulting bitmap satisfies this selection query. This method (called equality encoding [63]) is only one of several bitmap encoding schemes. For example, range [64], interval [64], and workload and attribute distribution oriented [65], and even several commercial database management encodings all leverage bitmaps to enhance indexing and query performance.

Raw	Equality			BSI	
	=1	=2	=3	$B_1[1]$	$B_1[0]$
1	1	0	0	0	1
2	0	1	0	1	0
1	1	0	0	0	1
3	0	0	1	1	1
2	0	1	0	0	1
3	0	0	1	1	1

Figure 3.1: Simple Example of Equality Encoded Bitmaps and Bit-Sliced Indexing for a Table with Two Attributes and Three Values per Attribute.

3.1.1 Review of Bit-Sliced Indexing

BSI (Bit-Sliced Index) [63,66] can be considered a special type of bitmap index encoding [67] designed to support complex queries over a large number of attributes. In this method, one bitmap (a BSI) is created for each attribute A, and "slice" k of this bitmap points to all the records whose A attribute has bit k set.

BSI arithmetic for a number of operations is defined in [66], and often requires only simple logic operations.

Previous work in bit slicing has focused almost exclusively on the memory savings, algorithmic efficiency, and query speed up of BSI. However, given the fact that the most common BSI operations only require logic gates and shift registers rather than full adders, it is reasonable to expect that this particular encoding might have power and energy performance features that other indexing schemes do not.

Compression of Bit-Vectors

Even though BSI is the most compact representation of an attribute (only $\lceil \log_2 values \rceil$ vectors are needed to represent all values), the high density of bit vectors makes them hard to compress any further – making it very challenging to fit a complete set of indices into memory.

There have been several different approaches proposed for dealing with this challenge. For example, a Bitmap Index for Database Service (BIDS) [68] indexes data selectively, storing compressed BSIs only for point and range queries.

Unfortunately, many of these proposed solutions to overcoming the index size challenge have difficulties. For example, BIDS actually requires an additional index such as Trojan [69] to retrieve complex queries. Moreover, BIDS compresses BSIs indiscriminately where it has been shown that this can have significant overhead in many high-dimensional queries [70]. Similarly, BBC also suffers from CPU intensive query retrieval which can be costly both in terms of query duration and in energy

consumption.

For very large datasets, all attempts to store a complete set of indices in a single machine's memory will eventually encounter limitations and in many cases, the indices must be distributed across multiple servers. It is in this space of massive distributed indices that we are interested in exploring the energy tradeoffs of BSI queries, compared to distributed arrays-based queries.

To maximize the size of index set that we can store and for better query performance, our implementation uses a modification of the Enhanced Word Aligned Hybrid (EWAH) [71] index compression method, that is described in [70]. This technique makes access to the bitmaps more CPU-friendly by using words instead of bytes to match the computer architecture, and only compresses the sparse bit-vectors.

Distributed Query Execution using Bit-Vector Arithmetic

Since our goal is to compare the power, energy, and performance tradeoffs of distributed BSI based queries with a more traditional distributed query approach, we will now discuss how big data analytics queries are executed using bitmap-based indices. Following this, we will describe the power measurement infrastructure set up over a Spark cluster.

Consider a relational big-dataset D with m attributes and n tuples. Bitmap indices are built over each attribute value or range of values and stored column-wise. The data can be partitioned vertically by grouping the bitmaps into multiple parti-

tions and/or horizontally, by splitting the bitmaps into segments. The query processing over bitmaps is done by executing bit-wise operations over one or more bitmap columns, then if horizontal partitioning was applied, the results are concatenated.

Traditional Distributed Queries

Querying an arrays-based distributed index is straightforward. The data is partitioned horizontally by rows across all servers in the cluster. When a query is made, a local agent sequentially scans through each row to see which ones match the query. Data must be examined on a field-by-field basis, and traditional arithmetic operations are used to perform the required matching tests. Once the local agent has collected all the matching records from its machine, it forwards these results on to an aggregator. Finally, the aggregators cooperate with each other to return the records of interest.

A Note on Hybrid Bitmap Compression

Before digging into the details of our bit-sliced index (BSI) implementation, a few notes on the compression method we used are in order. BSI encodes the binary representation of attribute values using one bit-vector to represent each binary digit. Since BSI bitmaps exhibit a high bit-density, i.e. a large number of bits is set in each bitmap, they do not compress well and are often stored verbatim, i.e. not compressed. However, queries execute a cascade of bit-wise operations. Often, even when the original bit-vectors are dense, the intermediate results become sparse and could benefit from compression. The hybrid compression [70] allows compressed and

verbatim bitmaps to coexist and be queried together.

Hybrid uses EWAH to compress the bit-vectors using groups of w-bits to maintain the alignment with verbatim bitmaps. EWAH uses two types of words to compress the bitmaps: marker words and literal words. Half of a marker word is used to encode the fills. The upper half (most significant bits) of the fill word encode the fill value, and the run length. The remaining bits contain the number of literal words following the run.

Verbatim Bitmap (in hex)		400003C0	00000000 00000000 00000000	001FFFF0	000001FF
EWAH Bitmap (in hex)	0000 0001	400003C0	0003 0002	001FFFF0	000001FF

Figure 3.2: A Verbatim Bitmap and its EWAH Encoding.

Figure 3.2 shows the EWAH encoding for a bitmap representing 128 bits, assuming 32-bit words. The EWAH bit-vector always starts with a marker-word. The first half of a marker-word represents the header specifying the type and number of fill words. The second half of the marker-word tells the number of literals that follow a marker-word (1 in the example). After the literal word comes another marker-word. The first bit (0) indicates a run of zeros and the value 3 is the number of words in the run. The second half of the marker word indicates that there are two literals following the fill.

3.1.2 Description of Bit-Sliced Queries

Selection and Range Queries over Compressed Bit-Vectors

A selection query is a set of conditions of the form A *op* v, where A is an attribute, $op \in \{=; <; \leq; >; \geq\}$ is the operator, and v is the value queried. We refer to point queries as the queries that use the equal operator ($A = v$) for all conditions and range queries to the queries using a between condition (e.g. $v_1 \leq A \leq v_2$).

The values queried, v_i, are mapped to bitmap bins, b_i, for each attribute. If the bitmaps correspond to the same attribute then the resulting bitmaps are ORed together, otherwise they are ANDed together. In the case of selection queries, the resulting bitmap has set bits for the tuples that satisfy the query constraints.

Algorithm 1: Pseudo-Code for SUM Aggregation over BSI

```
   Input: BSI B
   Output: sum
1  sum = 0;
2  for Each slice b_i ∈ B do
3  |  sum+= COUNT(b_i) × 2^i;
4  end
5  return sum
```

In parallel, these operations can be done independently over the different partitions as long as the bit positions are maintained consistently across partitions. For vertical partitions, bitmaps can be operated together. For horizontal partitions, bitmaps can be concatenated together.

Aggregation Queries

Bitmap indices are well known for their improved performance on aggregation queries. A COUNT aggregation can be performed just by counting the number of set bits in an equality encoded bitmap. BSI indices allow to compute efficient SUM aggregations as well. Algorithm 1 shows the pseudo-code to perform SUM aggregation for a BSI B that encodes the values to be aggregated. The algorithm returns the SUM aggregate value corresponding to the aggregation of the values by counting the number of ones in each bit-vector $b_i \in B$, and multiplying it by the corresponding power of 2 (2^i) (Line 3). This operation is done by calling the specialized CPU instruction POPCNT. Also, note that this multiplication is effectively a shift operation of i positions.

This aggregation can be combined with selection queries efficiently by ANDing the selection bitmap together with each slice in the BSI before performing the SUM aggregation.

Note that the result for the SUM aggregation is just a number, not a bitmap or BSI. In parallel, SUM aggregation is computed over each partition and these partial sums are then reduced into a global aggregate.

Top-k Preference Queries

In this work we define as Top-k query the process of summing all the attribute values for each data object and then perform ranking over the resulting aggregation.

The steps of performing the top-k preference query over a BSI attribute are

depicted in Figure 3.3:

- First, the weights are applied to their respective dimensions for each data object. Using BSI arithmetic, this multiplication is done by shifting bit-slices and performing additions (SUM_BSI). This multiplication and addition procedures are described in Algorithm 2 and 3 respectively by Guzun et al. in [72].
- Aggregate all the dimensions into one dimension by summing the values across attributes.
- Extract the top-k data objects by finding k rows with the largest values in the aggregated attribute. This procedure over a BSI attribute is described in Algorithm 4.2 by Rinfret et al. in [66].

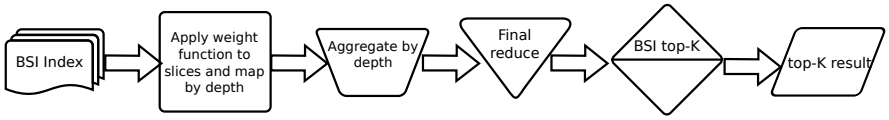


Figure 3.3: Top-k (preference) Query Stages using Distributed BSI Arithmetic

In the second step described above, for performing SUM_BSI in parallel we use the slice-depth mapping as described in [73]. This aggregation algorithm promotes the bit-slices as the processing data units and applies the lessons-learned in computer arithmetic optimization to further improve the performance of the parallel aggregation. The basic idea of this approach lies in use the bit-slice depth as the mapped key and implement a two-phase algorithm.In the first phase, the slices are added by

bit-depth, producing a weighted partial sum BSI. In the second phase, all the partial sums are added together in a method similar to a carry-save adder.

3.2 Experimental Results

3.2.1 Experimental Setup

In this section we evaluate the execution times and energy usage of the three types of queries described above: $SumAggregation$, $Top-k$, and kNN. We compare the algorithms that use the BSI index and distributed bit-vector arithmetic (referenced as BSI) against distributed map-reduce methods (referenced as Arrays). We did not evaluate individual logical bitwise operations such as AND or XOR, since the $Top-k$, and kNN consist of a string of these types of operations.

In our experiments we used two synthetically generated datasets with uniform distribution, and two representative scientific datasets from recent publications in particle physics and face detection. The two synthetically generated datasets have 1 Billion (designated as `Synth`) and 200 Million rows respectively (designated as `SynthSmall`), with a cardinality of 10^9 (30 slices per BSI). The two nonsynthetic datasets are referred to in the remainder of this paper as `HIGGS` and `Images`. The `HIGGS` dataset was provided by the authors of [74], who used Monte Carlo simulations to produce 21 kinematic features representative of measured particle detector events, and then generated 7 high-level classification features from the "raw" event data. This dataset has a high cardinality, as each attribute is represented by real numbers with with numeric precision of 16. In a non-compressed form, this dataset has a size

of 7.4 GB.

The `Images` [75] data are from the Edith Cowan University face detection database. The database consists of over 3200 color images of various skin types and their corresponding images of segmented skin regions. In our experiments we used 2000 of these images. Each pixel is sampled together with a 6x6 pixel block surrounding it. Thus each pixel has 6x6x3 = 108 features. The size of this dataset is approximately 20 GB.

For the $SumAggregation$, $Top-k$, and kNN queries, the raw data is horizontally partitioned and stored as HDFS object files, to avoid incurring parsing costs during query execution. We did not consider vertical partitioning as the queries under evaluation involve all the dimensions, and vertical partitioning would require more data shuffling. For all queries, the data is arranged such that sub-tasks can be executed in parallel over each partition and then aggregated. The `HIGGS` data were stored as Doubles, the `Images` data as Integers, and the synthetically-generated data, as Longs.

In choosing the parameters for our 4-server Spark cluster, we set the HDFS replication factor to 3, the number of Spark executors to 8 with 6 cores per executor, and allocated 8GB of memory to each executor.

3.2.2 Measurement Procedure

Some power-aware algorithm designs emphasize the common wisdom that "faster is better" when it comes to energy savings. And, BSI is indeed faster than the

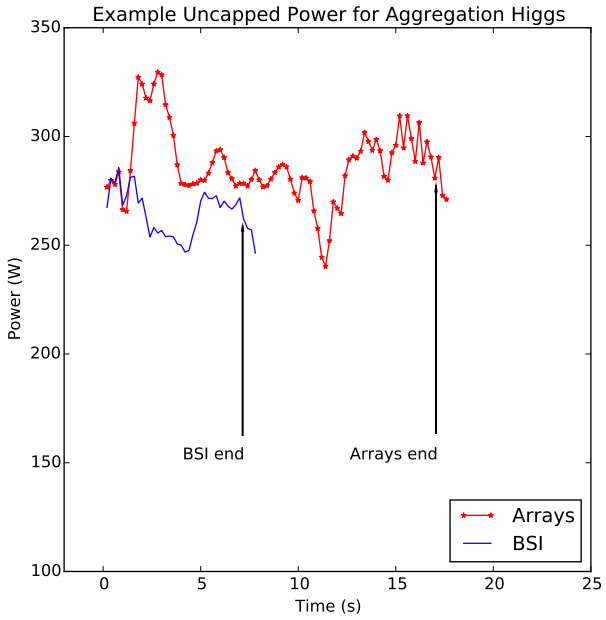


Figure 3.4: Queries without Power Constraints - Higgs Dataset - Aggregation Query

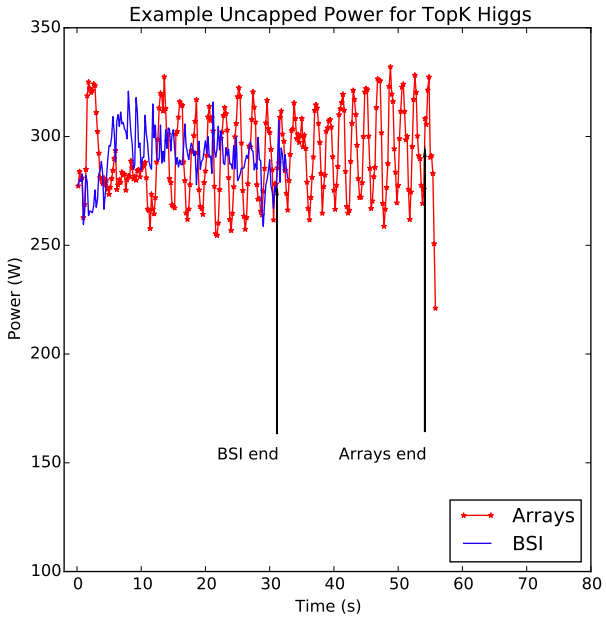


Figure 3.5: Queries without Power Constraints - Higgs Dataset - TopK Query

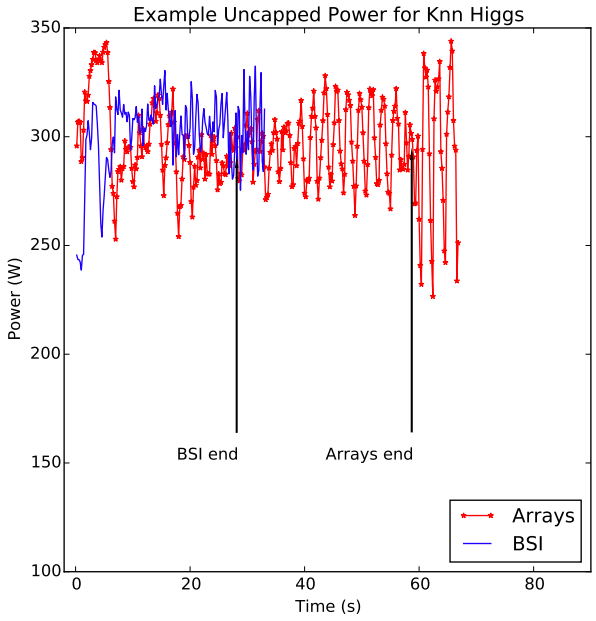


Figure 3.6: Queries without Power Constraints - Higgs Dataset - Knn Query

distributed query over arrays. However, BSI has design advantages which go beyond accessing hardware acceleration features (like GPU) or simply "pushing the processor harder" (caching and other ways to reduce non-CPU latency). These "extra" benefits are most clearly seen when the processor is constrained to a certain maximum power – as might be the case in datacenters seeking to save power by allowing an increased inlet temperature for the servers or in datacenters with many smaller servers rather than a few high-powered servers.

To enforce a maximum processor power consumption for each server, we used the Running Average Power Limit (RAPL) hardware feature accessible via a model-specific register (MSR) interface on many Intel servers. Extensive initial measure-

ments showed that RAPL accurately limited the processors' power consumption to within less than a watt of the desired value, so we assumed that the limit was accurately enforced throughout the rest of the experiments.

For each combination of query type, algorithm, dataset, and per-processor power cap, we ran twenty sequential queries over the whole dataset (we will refer to one of these query sets as an "application") while logging the cluster's total power consumption over the entire interval. We extracted the duration of each application, defined to be the difference between the last job's completion time and the first job's start time, from the Spark logs.

Because the total run time for each application was often small compared to the application's resource allocation time, it was useful to separate the "working time" from the "container setup time." To do this, we configured the Spark scheduler to defer until all resources were allocated before starting any of the application's jobs.

Since there is some variation in the duration of an application run and because unrelated background processes on the cluster cause some uncontrollable power variation, we ran each application ten times.

For convenience of discussion, let us define $d_{i,j}$ to be the j'th measured duration of application i. Further, let $\langle d_{i,j} \rangle_j = \frac{1}{j_{max} - j_{min}} \sum_{j_{min}}^{j_{max}} d_{i,j}$ be the sample average of the duration. In the discussion that follows, when we speak about application i's "duration," we refer specifically to $\langle d_{i,j} \rangle_j$.

The notion of application i's "energy" requires a bit more explanation since the quantity we are interested in is actually the *extra* energy required to run the

application rather than to just sit idle. To avoid "double counting" the effect of idle power consumption in our energy comparison, we measured the average idle power of the whole cluster over a 30 min interval, and subtracted this value from the measured power consumption before using that power to calculate energy. Specifically, let $p_{i,j,k}$ to be the k'th time sample of measured cluster power when running application i for the j'th time, b_l to be the l'th time sample of the measured cluster power at idle. Application i's "energy" is then the average extra energy it takes to run the application instead of sitting idle for that time, or $\langle d_{i,j} \times (p_{i,j,k} - \langle b_l \rangle_l) \rangle_j$.

Table 3.1: Energy, Time, and Power Tradeoffs among Arrays and BSI - Aggregation Queries

	Higgs arr	bsi	Img arr	bsi	Syn arr	bsi	SmSyn arr	bsi
Peak(W)	125.9	81.9	145.8	99.1	145.9	86	159.6	111.1
	enrg	time	enrg	time	enrg	time	enrg	time
22Wratio	2.9	2.2	9.9	8.6	323.1	194.4	18.3	13.4
35Wratio	3.2	2.4	9.5	9.2	326.6	169.5	18.6	12.5

Table 3.2: Energy, Time, and Power Tradeoffs among Arrays and BSI - Top-K Queries

	Higgs arr	bsi	Img arr	bsi
Peak(W)	140.2	128.7	159.7	143.1
	enrg	time	enrg	time
22Wratio	2.4	2.5	3.4	3.7
35Wratio	2	2.1	2.9	3.6

Table 3.3: Energy, Time, and Power Tradeoffs among Arrays and BSI - KNN Queries

	Higgs arr	bsi	Img arr	bsi
Peak(W)	128.3	117.1	154.8	141.8
	enrg	time	enrg	time
22Wratio	2	2	5.6	5.9
35Wratio	1.8	1.7	4.8	5.8

3.2.3 Peak Power Experiment

Our first experiment was to compare the peak cluster power consumption under BSI queries with the peak cluster power consumption under distributed queries over arrays. To limit the influence of outliers on each application's peak power, we performed an ensemble average across the ten application runs in order to obtain a typical load shape for that application $l_i(k) = \langle p_{i,j,k} \rangle_j$. The typical load shapes for the Higgs dataset, without power constraints are shown in Figs. 3.4 to 3.6 and suggest that the BSI queries consistently exhibit a lower peak power than the queries over arrays. The other datasets tabulated in Tables 3.1 to 3.3 tell a similar story.

The hardware counters available to us did not allow for detailed measurement of the processor state. However, one reasonable explanation for BSI's lower peak power is that the computational load of BSI-based queries is composed of simple logic operations, rather than more complicated arithmetic. At a hardware level, this might be expected to reduce the number of logic transitions, resulting in a reduced dynamic processor power. Furthermore, depending on the operating system and processor

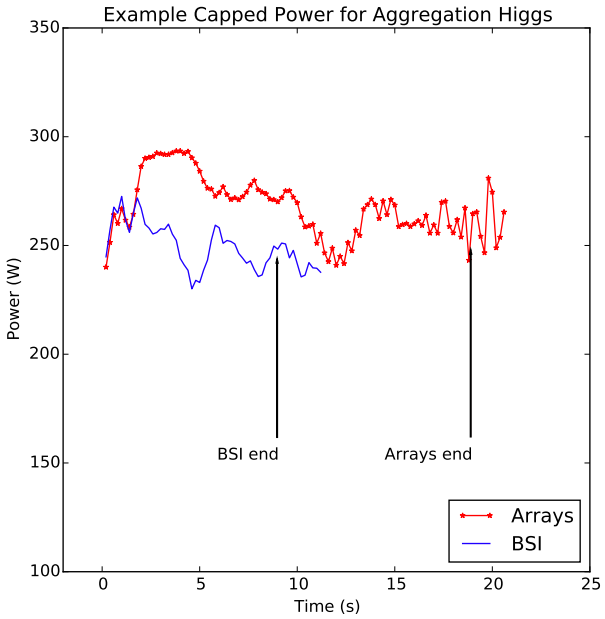


Figure 3.7: Queries with 25 W Power Constraint - Higgs Dataset - Aggregation Query

power management behavior, unused adders could be switched off, resulting in a reduced static power draw.

3.2.4 Duration Experiment

Because the BSI and the traditional arrays-based queries have different peak power, we were curious about the effects of processor power capping both on the duration of the queries and the total energy that the queries consumed. Put a different way, the significantly-shorter runtime of the BSI algorithms suggests some flexibility in their power constraints. In fact, our results showed that even the power-constrained BSI queries were faster than the corresponding unconstrained queries over arrays.

Figs. 3.7 to 3.9 show the effect of a 25 W per-processor power cap (since the

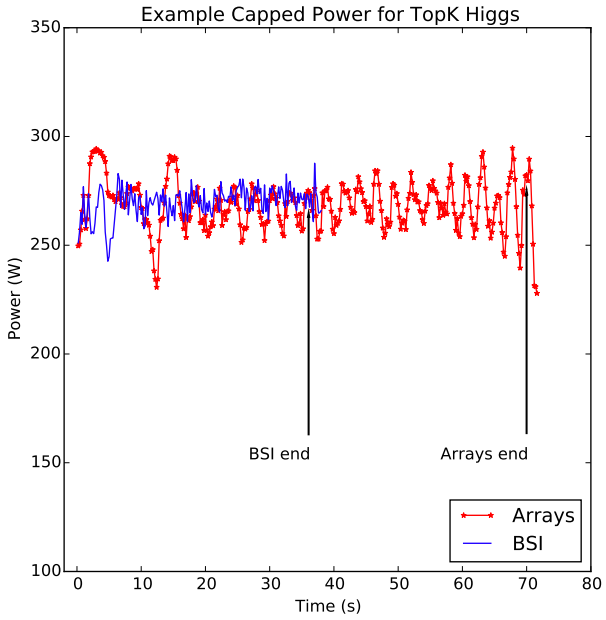


Figure 3.8: Queries with 25 W Power constraint - Higgs Dataset - TopK Query

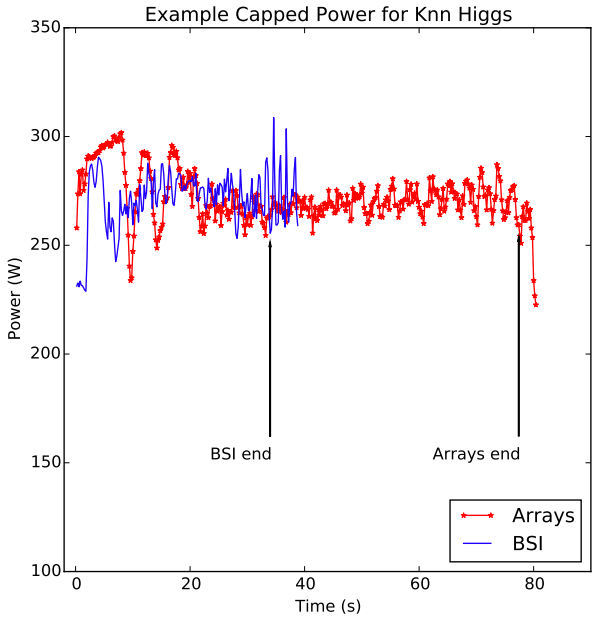


Figure 3.9: Queries with 25 W Power Constraint - Higgs Dataset - Knn Query

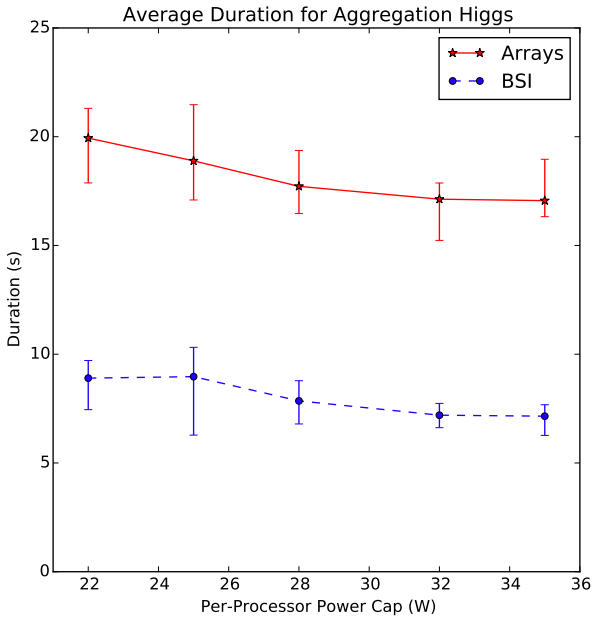


Figure 3.10: Query Set Duration under Power Constraints - Aggregation Query

power cap applied only to the processor, the whole server power was not strictly constrained) on the cluster. Comparing Fig. 3.9 with Fig. 3.6, the power cap had a negligible impact on the duration of the BSI query, while it had a large impact on the duration of the distributed array query.

We systematically evaluated the effect of power capping on the duration of the query sets. We chose five different power caps, ranging from the minimum processor power (22 W) to the maximum processor power (35 W). We recorded the minimum, maximum, and mean duration over the ten application runs for each power cap. This data is shown for the Higgs dataset in Figs. 3.10, 3.12 and 3.12. As is especially evident in Fig. 3.12, power capping often has a stronger effect on the distributed

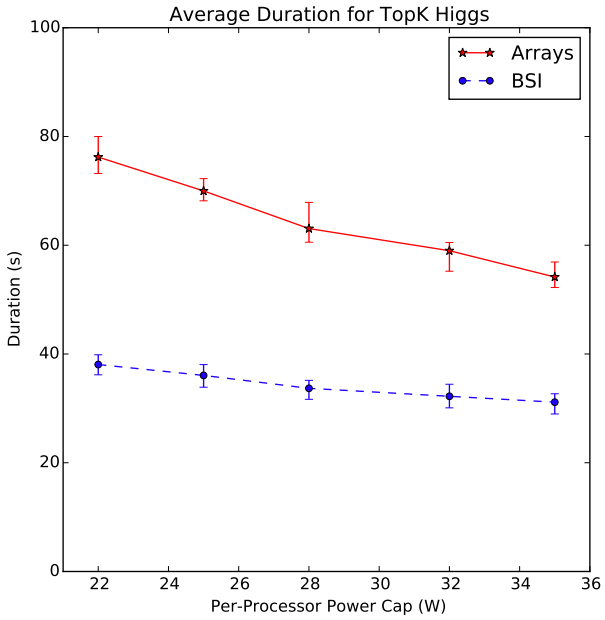


Figure 3.11: Query Set Duration under Power Constraints - TopK Query

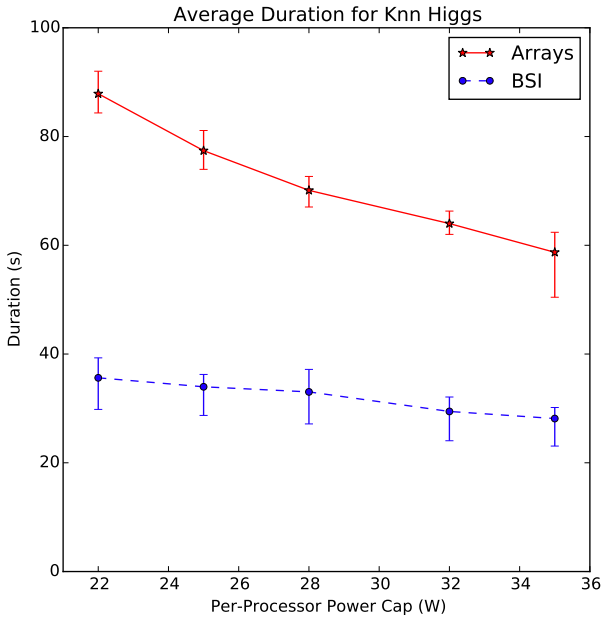


Figure 3.12: Query Set Duration under Power Constraints - Knn Query

array scan query duration than it does on the BSI query duration.

3.2.5 Energy Ratio Compared to Duration Ratio

To get a better understanding of the tradeoff between the energy required to run a query and the time a query takes to run, we calculated the ratio of distributed arrays application energy to BSI energy and the ratio of distributed arrays-based application duration to BSI duration for a wide variety of query types and datasets. As is shown in Tables 3.1 to 3.3, the worst-performing BSI queries were around twice as fast as the corresponding arrays-based queries. Furthermore, with the exception of the Aggregation queries on the Higgs and Images dataset, every combination we ran showed that the query execution over arrays' duration was more sensitive to power capping – with the BSI queries showing improved relative performance as the power cap approached its minimum.

We also observed that every aggregation query showed energy gains beyond what might be expected from the faster runtime alone. The BSI shows very extreme improvement over the arrays in the aggregation query with synthetic dataset because the index size for BSI fit into memory, but index size for the arrays was too large to fit into memory.

3.3 Chapter Summary

In this chapter, we set out to experimentally evaluate the power, energy, and performance tradeoffs of distributed BSI, compared to a more traditional array based query indexing. In the first few sections, we included enough background and imple-

mentation details to reproduce our experiment. Our results over three popular types of Big Data queries (aggregation, top-k, and k-nearest neighbor) and several real-world and synthetic datasets showed that BSI consistently outperforms non-indexed data in execution time. It was Furthermore, the peak power of BSI was always lower than the peak power of the arrays-based approach. Because of this, the relative performance (in terms of duration) of all but two of the BSI queries got even better as the power cap became smaller.

The power and energy characteristics of BSI suggest more effective resource allocation strategies within a datacenter. Servers with power constraints (due to temperature, electricity prices, etc) can be allocated to query processing, while unconstrained servers can be put to other use. For future work we plan to explore other types of indexing and analytic queries, as well as emerging hardware technologies such as co-processors.

Because the read latency of clusters using the bit-slicing optimization is somewhat insensitive to power cap, it is possible to achieve load shaping simply by the addition of a parallel deferrable "base" CPU load. Put another way, clusters which migrate from the baseline query approach to the bit-slicing optimization could gain enough flexibility in their power requirements to support load shaping *and* an additional low-priority workload.

CHAPTER 4
POWER SHAPING WITH UPDATE-AWARE BITMAP QUERIES

As with Chapter 3, the larger context of this chapter is investigating the power/performance tradeoffs of demand response-applicable workloads which have QoS metrics calculated on timescales similar to the load shaping timescale. Where Chapter 2 focused on architecturally simple workloads in order to introduce a general framework for datacenter demand response and Chapter 3 investigated a more complex workload where QoS could be made insensitive to power capping, this chapter explores a situation where deferring the low-priority "base" workload can impact the performance of the high-priority workload. The specific application of interest is a "streaming big data" service, where queries are latency-sensitive, but have some small flexibility in freshness.

In this chapter, we begin with with an experiment demonstrating that even in a cluster environment, simple job priorities can allow query latency to be unaffected by power cap, while index refreshing supplies sufficient "reserve load" to do bidirectional load shaping. Following this experiment, we provide some brief context and motivation for update-aware bitmap indices in dynamic big data environments, compared to other strategies. We then present a new bitmap update strategy which outperforms the existing state-of-the-art method, at least in the single server case.

4.1 Priorities and Power Shaping

While any workload with differentiated QoS requirements can benefit from job priorities, low-power algorithms like the ones used for bit-sliced-index queries in Chapter 3 may have the added benefit of not requiring separate power-aware priority management. For example, consider a situation where the cluster's power setpoint can vary between 280W to 400W. If the high-priority workload on that cluster only uses a maximum of 240W, there will always be at least 40W available – suggesting extra compute capacity to ensure low-priority jobs do not starve. Beyond setting the high priority workload's share to maximum, no further adjustments of job priorities are needed – regardless of how the power setpoint varies within that range.

The ability to avoid adjusting job priorities in realtime is especially helpful when operating in a cluster environment. For example, cluster management frameworks like Apache Spark [76, 77], and YARN [78] allow for setting static job queue priorities using configuration files but provide no simple way for adjusting queue priorities during runtime.

One important difference between traditional operating-system priority management and cluster-level priorities is the granularity of the scheduling units. For scalability reasons, cluster schedulers generally do not support the ability to suspend and resume low-priority tasks in order to share CPU time at a fine-grained level. Instead, workload priorities generally take the form of queuing order, reserved CPU bins, and preemption (ability to cancel lower-priority tasks).

Since cancelling and restarting tasks incurs both computational and power

overhead, the coarse granularity of cluster-level priorities raises the question of whether or not the simple priority policy as described above actually applies in a cluster environment. In the context of latency-sensitive query reads and lower-priority index updates, we are curious if the simple policy can (1.) ensure that queries do not suffer latency impact under power cap, and (2.) does not cause updates to starve or query freshness to degrade excessively. Furthermore, cancelling and restarting tasks might cause temporary jumps above or dips below the current power setpoint – impacting the quality of the power tracking. So, we are also interested to evaluate the dynamic power behavior of this particular workload.

4.1.1 Query Latency and Freshness

Architecture and Setup

The hardware setup is the same as described in Chapter 1, and the cluster architecture is similar to the setup described in Chapter 3. Specifically, the cluster runs an up-to-date Spark-on-YARN stack (Spark 2.1.1 on YARN 2.3) and consists architecturally of four "executor" nodes, one "master" node, and a separate non-cluster node for monitoring power consumption. We use the RAPL interface discussed in Chapter 2 to enforce the power caps.

The diagram in Fig. 4.1 gives a functional view of the software setup. The idea behind this architecture is to optimize for a scenario where the cluster must provide low-latency reads to a constantly-changing database, while providing a best-effort query freshness guarantee. In this experiment, the database consists of 100 columns,

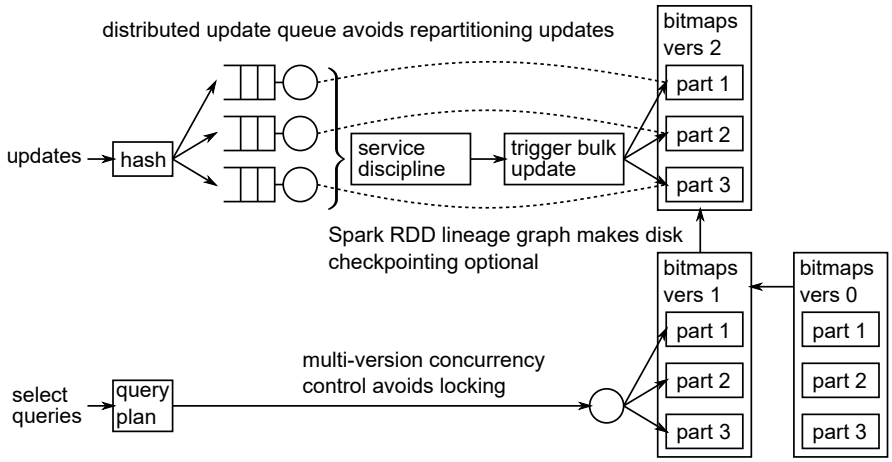


Figure 4.1: System Diagram of Update-Aware Query Processing Framework

1 million rows, and uniformly-distributed values of cardinality 100. We investigate the power tradeoffs of four different update strategies on a run-length encoded bitmap index. Each update strategy is discussed and compared in more detail in the following sections. In this section, we focus on showing that read-only query latency does not suffer under power cap,

To avoid an update queuing bottleneck, incoming updates are hashed by row or column (depending on which indexing strategy is used) to a queue on the executor node containing the data to be updated. In this experiment, updates arrive as Poisson process and are uniformly distributed across the rows and the columns of the database. The rate of updates is constant across all experiment runs, and is set at 1000 rows per second.

The master node uses the default Spark scheduler to repeatedly trigger "snapshot" updates, which merge all updates which have been accumulated since the last

snapshot into a new database version.

Versioning is done to avoid contention with incoming queries. In this particular experiment, we checkpoint after every update in order to ensure maximum possible query speed. However, it is worth noting that our update architecture uses Spark's in-memory Resilient Distributed Datasets [77] paradigm and therefore does not require disk checkpointing with every update.

In this experiment, wide range queries (with value selectivity of 99% and column selectivity of 50%) arrive at the master node according to a Poisson process. The arrival rate is selected to be the same for all test runs and is set at 4 queries per second (corresponding to around 50 million rows per second, given the choice of column selectivity). The query priorities are set somewhat conservatively in that while queries are always first in line at the cluster scheduling queue, each update aggregation task can be preempted (cancelled and restarted) only once, and queries have no "reserved" cpu slots. This is done to avoid wasting compute resources and to ensure that the full dynamic range of power can be used.

Experimental Results

By measuring the instantaneous power consumption of the cluster at each power capping level, we obtained the results of Figs. 4.2 and 4.3, which show that this update-aware indexing architecture provides high quality dynamic power characteristics. Fig. 4.2 shows that this particular workload has sufficient CPU load to

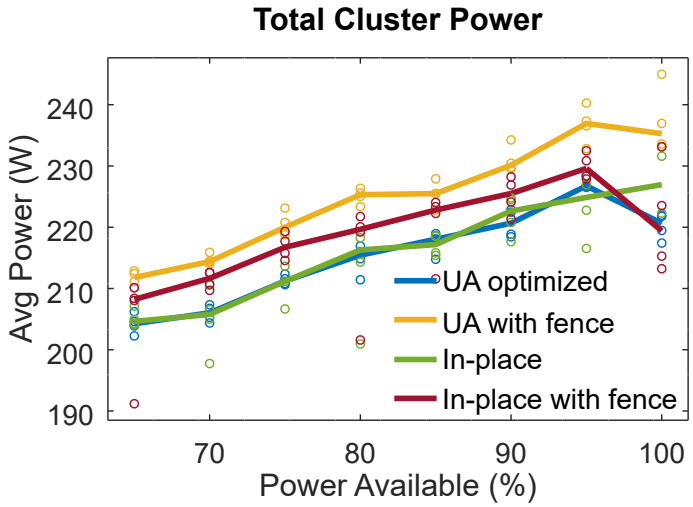


Figure 4.2: Average Power Consumption of Whole Cluster

ensure the power consumption stays within the controllable range[1] for the power capping interface we selected. Fig. 4.3 plots the PJM "power tracking score" measure [46] discussed in Chapter 2, which indicates how "cleanly" the measured instantaneous power consumption matches a desired setpoint value. A value of 100% indicates that there is no variability around the setpoint. While certain update disciplines responded better to power capping than others, all workloads manage to stay well above the minimum score (75%) required to participate in PJM's regulation reserve market.

Having answered our question the dynamic power characteristics of this workload, we now verify our intuition that the low power requirements of bitmap index queries translates to a query latency guarantee across a range of power caps. This experimental validation is especially important, considering the coarse-grained nature

[1]Please refer to Chapter 2 for a discussion of how server power dynamic range relates to the choice of power capping interface.

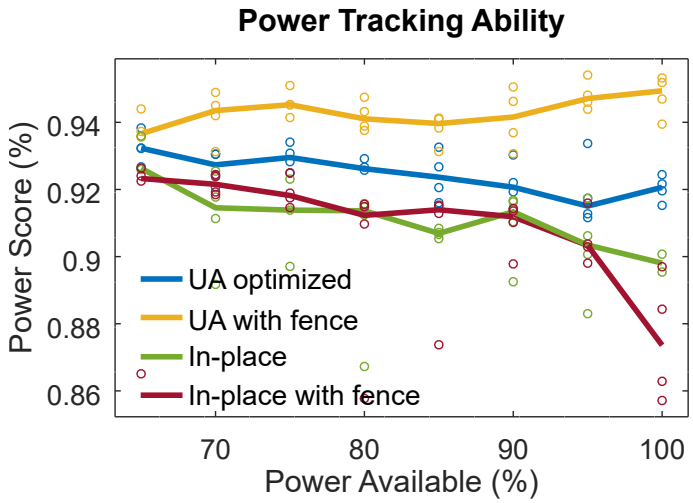


Figure 4.3: Power Tracking Score of Whole Cluster

of the cluster priority mechanisms.

As is shown in Fig. 4.4, there are some differences in query latency across update discipline that we will discuss later. However, the current takeaway is that query latency does not pay a penalty for power capping. Furthermore, the results of Fig. 4.5 show that the query freshness degrades smoothly under power capping – indicating that the updates do not starve even when allocating high priority to the queries.

4.2 Read-Optimized Queries on Changing Datasets

In the previous section, we showed that a mixed workload of high-priority queries and best-effort updates to a distributed bitmap-indexed database can provide good power tracking characteristics while ensuring high quality of service for queries. In the course of investigating this particular workload with respect to power shaping, we discovered a new bitmap index updating technique that outperforms the existing

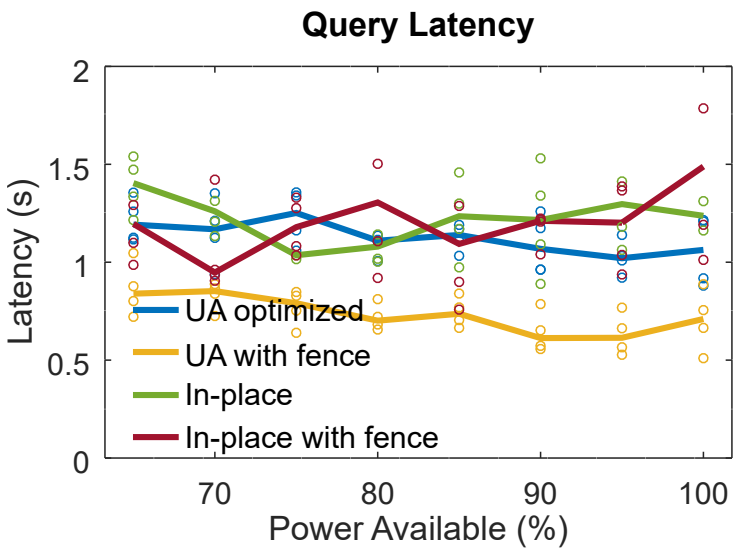


Figure 4.4: Query Latency under Power Cap

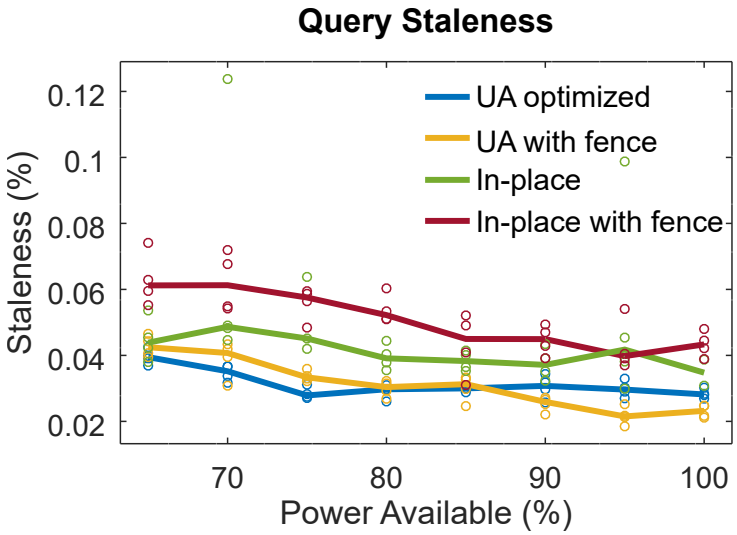


Figure 4.5: Query Staleness under Power Cap

state-of-the-art solution. Similarly to the bit-sliced index query optimizations of Chapter 3, this update optimization was not designed with power in mind, but it did have the fortunate side-effect of reducing the power required for updates.

We begin by briefly outlining the broader context of update-aware indexing and discuss existing techniques for update-aware bitmap indices. Following this, we discuss the

4.2.1 Dynamic Datasets and Update-Aware Indexing

Architectural frameworks like Apache Storm (Twitter) and Spark Streaming provide convenient development platforms for performing queries on large, constantly-changing datasets. In fact, over half of the 900 companies surveyed in Databricks' 2016 report indicated that they used Spark Streaming in their production environments [79]. The use cases of these dynamic big data frameworks range from fraud detection and network traffic analysis, recommendation engines, data warehousing, business analytics, and scientific computing [80].

Indexing has long been employed to reduce query latency in traditional database solutions. And, many of the distributed query processing frameworks that work atop streaming solutions (SparkSQL and Apache Cassandra, for example) also employ caching and indexes as a way to speed up data access. However, as with traditional databases, a common assumption is that the indexed data changes infrequently. Even so, there is a small but growing body of indexing solutions that do efficiently support updates. For example, two application-specific updateable indexes include fat

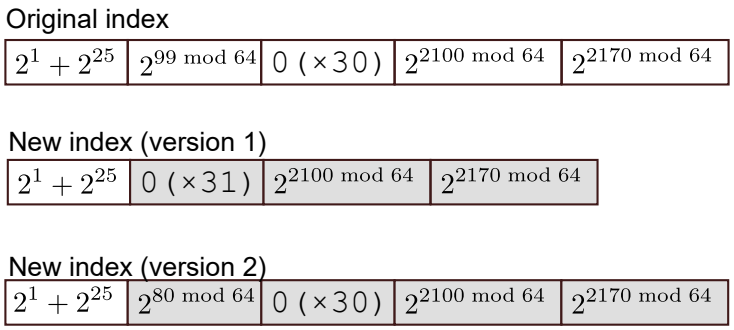


Figure 4.6: In-Place Updates with Word-Aligned Bitmaps

B-trees [81], which support parallel filesystem access, and velocity-constrained indexes [82], which are designed for moving object databases. However, these two application-specific approaches rely on prior knowledge of the class of queries and the structure of the data represented by the index.

4.2.2 Update-Aware Bitmap Indices

In the space of "big data analytics," unstructured data and ad-hoc queries are the norm rather than the exception. As discussed in Chapter 3, run-length encoded (RLE) bitmap indices are one type of compressed index that is particularly useful in unstructured data, unknown query environments because they can directly support a wide range of queries and column operations without the need to decompress or access the main dataset.

Two previous approaches have been proposed to add support for updates updates to RLE bitmap indices. An approach called update-conscious bitmaps [83] made use of a per-column "existence mask." This approach enabled an efficient delete-then-append operation without re-encoding the index. A technique called UpBit [84]

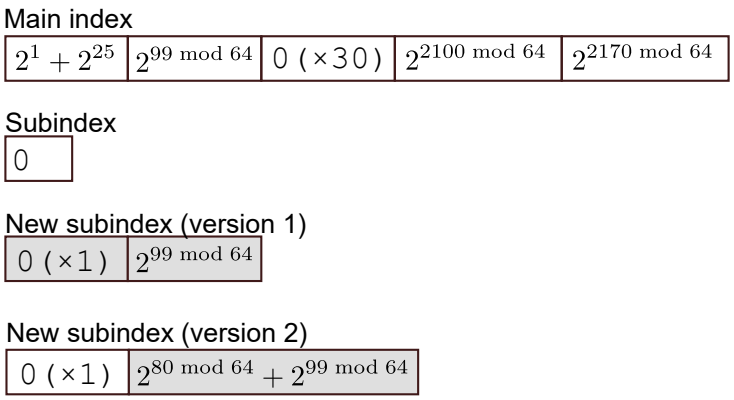


Figure 4.7: Improved Updates with Word-Aligned Bitmaps

improved on the update-conscious bitmaps by employing a distributed per-value ancillary index called an "update vector."

This work differs from the previous proposals for update aware bitmap indexing in two ways. First, while previous efforts have targeted on single-server multiprocessors, this implementation is designed expressly for "the cloud" – making use of the widely-deployed Apache Spark framework. Second, our approach allows improved support for multi-row updates. This is especially useful in shared-computing cloud platforms, where there may be benefit to micro-batching the update tasks[2]. We further show that this approach has a reduced memory imprint, compared to the previous technique.

[2]Put another way, there may be excessive network bandwidth or overhead cost to distributing updates individually.

4.3 Optimizing for Multi-Row Updates

4.3.1 Review of Single-Row Updates

The basic premise behind update-aware indices is to avoid re-encoding the entire with every update. In the case of bitmap indices, the following example illustrates how the UpBit design is able to achieve improved update performance over single-row in-place updates. We will extend this example in the next section to explain how our aggregation-optimized strategy allows continued performance improvements, even under muti-row updates.

For ease of explanation, our example considers a simple word-aligned bitmap encoding for the index, but it should be noted that a similar logic applies across a range of different bitmap encodings. The bitmap index of Fig. 4.6 uses a word length of 64 and indicates that rows $\{1, 25, 99, 2100\}$ are initially set. In this example, there are two updates which must be applied to the index:

1. First, row 99 is to be unset.
2. In the next update, row 80 is to be set.

An in-place update proceeds as is shown in Fig. 4.6, where shaded words highlight words that have been modified by the update. In order to determine which encoded word contains bit 99, the first two words must be read. Since the word previously encoding bit 99 is now empty, it must be merged into the following run of zeros. This operation changes the length of the encoded bitmap, and thereby triggers a read-and-copy operation on the remainder of the bitmap. When bit 80 is set in the second update, the entire bitmap must again be re-encoded.

To see how maintaining a separate sub-index for updates can help improve update speed, consider Fig. 4.7. To determine whether bit 99 needs unsetting, the first two words of the main index, and the first word of the (empty) sub-index are read. Since bit 99 is set in the main index and has not already been marked as unset in the sub-index, two words are written to the sub-index to mark bit 99 as needing flipped. In this case, it is important to note that the remainder of the main index need not be read, nor does the main index need to be copied. Similarly, when bit 80 is set, only the sub-index is re-encoded.

4.3.2 Need to Support Multi-Row Updates

This ability to gradually absorb updates comes at a slight cost to query read performance since the sub-index and the main index must be XOR'ed together to determine which rows are actually set or unset. One idea to mitigate this performance hit is to periodically merge the sub-index back into the main index. At low update rates, this is mergeback alone is sufficient to ensure that the sub-index remains sparse enough to make the read overhead of the sub-index negligible.

However, the case of aggregated updates can quickly limit these performance gains. To see this, consider a slight modification to the previous example:

1. First, rows 99 and 2100 are to be unset.
2. In the next update, rows 80 and 256 are to be set.

As is shown in Fig. 4.8, the size of the sub-index can grow much more rapidly than intuition might suggest. If the sub-index reaches its mergeback density after only

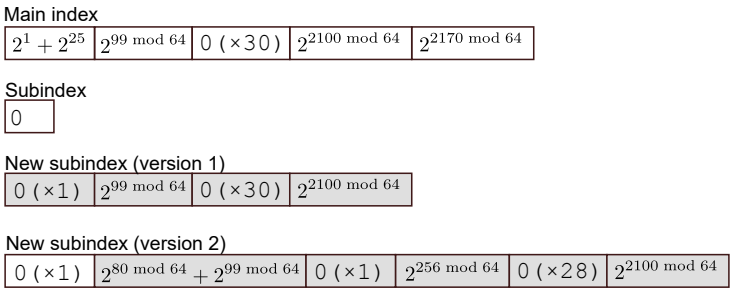


Figure 4.8: The Problem of Aggregation

a small number of multi-row updates, the performance of this approach can actually be worse than in-place updates, due to the overhead of maintaining two indices rather than one.

This can happen even at relatively small batch sizes. For instance, consider a database attribute with uniformly-distributed data of cardinality m. If a word size of w bits is used, the expected density of non-fill words in the m main index bitmaps is $1 - \left(1 - \frac{1}{m}\right)^w$. With $w = 64$ and $m = 100$, a little under half of the words are non-empty.

If updates are uniformly distributed across the rows, and are aggregated at some fraction p of the total number of rows, a given word has a $1-\left(1 - \left(\frac{1}{m} + p - \frac{2p}{m}\right)\right)^w$ probability of needing changed. With $w = 64$, $m = 100$, and $p = 0.05$, that comes to around a 3% density of non-empty words after just one multi-row update. In this case, the size of the secondary index would be expected to reach the size of the main index after only around 20 multi-row updates.

In order to keep the density of the secondary index from growing too rapidly in the presence of aggregated updates, it is possible to exploit the fact that when

the main index is relatively dense, there is a relatively high likelihood of any given update intersecting a literal word in the original index. As long as updating the word does not result in changing the size of the encoded bitmap, the word can be modified in-place without triggering a full re-encode of the bitmap. Only words which would trigger a full re-encode of the main index are merged into the secondary index. The algorithm is shown in Algorithm 2.

Algorithm 2: Pseudo-Code for Aggregation-Optimized Update-Aware Index

```
   Input: rows to set S, rows to unset U
   Output: new main index bv and new secondary index uv
   /* W_{s,u} is a list of (set, unset) word pairs                    */
 1 tmp → {};
 2 W_{s,u} ← rowToWords(S,U);
 3 for Each s_i, u_i ∈ W_{s,u} do
 4 |   w ← findWordIn(i,bv) ⊕ findWordIn(i,uv);
 5 |   if isRun(i,bv) then
 6 |   |   if isRun(i,uv) then
 7 |   |   |   appendWordTo(w,i,tmp)
 8 |   |   end
 9 |   |   else
10 |   |   |   updateInplace(s_i ∨ (w¬u_i),i,bv)
11 |   |   end
12 |   end
13 |   else
14 |   |   updateInplace(s_i ∨ (w¬u_i),i,bv)
15 |   end
16 end
17 newUV ← uv ⊕ tmp;
18 return (bv, newUV)
```

4.4 Performance Comparison of Aggregation-Optimized Bitmaps

Having now have explained the context, motivation, and operation of aggregation-optimized bitmaps, we can revisit our previously-described query latency and freshness experiment to discuss the tradeoffs of this particular update optimization. Following this discussion, we present a more traditional performance comparison using a single server and non-parallelized queries.

In both experiments, we compare the performance of our aggregation-optimized bitmaps (labeled "UA optimized" in the figures) with three other update strategies. The primary comparison of interest is, an implementation of UpBit [84] (labeled "UA with fence" in the figures, because it makes use of fence pointers to improve the speed of random row access on the main index). And, as a common baseline for comparison, we also included in-place updates (re-encode the whole index, with every update) and in-place updates making use of fence pointers to improve the speed of random row access.

4.4.1 Revisiting Query Latency, Freshness, and Power Characteristics

The average power consumption shown in Fig. 4.2 indicates that using a row-by-row update strategy (as is done when employing fence pointers) incurs a noticeable power overhead. UpBit has the best power tracking score (see Fig. 4.3) of the interfaces investigated, due to the fact that its extra power overhead was consistent over time rather than being the result of unpredictible "bursts" of power. A surprising aspect of this particular experiment, considering the single-server results in the next

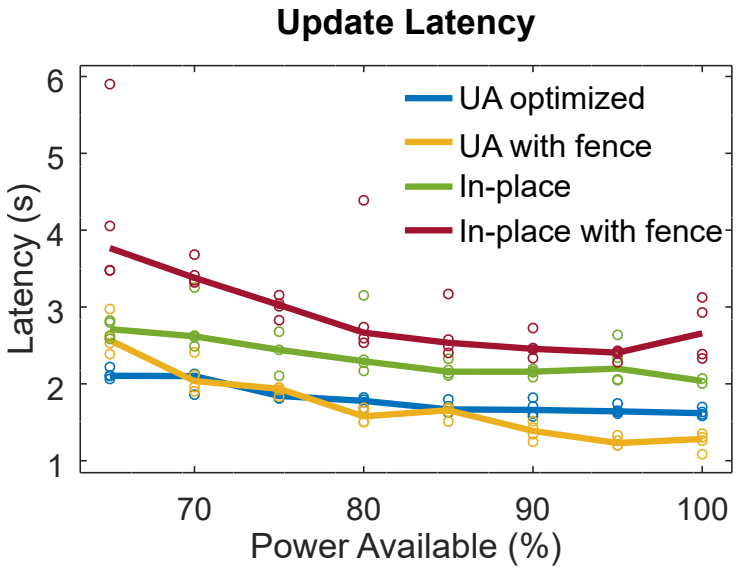


Figure 4.9: Update Latency of Different Bitmap Update Strategies under Power Cap

section, is that the UpBit strategy seems to have noticeably-improved query performance over both our aggregation-optimized approach and even the in-place updates under the specific choice of update rate. Given that query read time is necessarily slower than the in-place query reads (due to the necessity of reads needing to merge the secondary and main indices), we must conclude that the query latency advantage is the result of some other aspect of the system. Further investigation is needed to better understand this behavior. And, it is worth noting that this particular experiment only investigates updates at a single, relatively low rate. Had we chosen a higher update rate, it is possible that the results would show a different scenario.

Finally, we can observe that Figs. 4.5 and 4.9 demonstrating how update-aware bitmaps outperform in-place updates. In this case, as well, the UpBit implementation and our aggregation-optimized approach perform similarly.

4.4.2 Performance Comparison Experiment

To better understand the performance of the aggregation-optimized bitmaps, we designed a simple single-server experiment involving one 100000-row column initialized to uniformly distributed data with cardinality 100. Rather than allowing updates and queries to proceed in parallel as in Section 4.1, 200 updates are interleaved sequentially with range range queries (value selectivity of 99% and attribute selectivity of 50%). We vary the update aggregation size between 0.0001% and 10% of the total number of rows. After each update, we measured the total size of the index in words. For this experiment, we chose a mergeback size of 100 words for the update-aware indices. To ensure the validity of our results, each test was repeated four times.

From Fig. 4.10, it is clear that the aggregation-optimized bitmaps outperform or match the other strategies for all but the highest aggregation sizes – at which point the in-place updates start to outperform the other methods. The results of Fig. 4.4 show similar query latency for both aggregation-optimized bitmaps and the UpBit implementation. As discussed previously, the query latency of in-place updates is necessarily lower than can be achieved by update-aware bitmaps, due to the extra step of merging the main and secondary bitmap indices with every read. Last, the results of Fig. 4.12 show improved memory performance over the UpBit implementation.

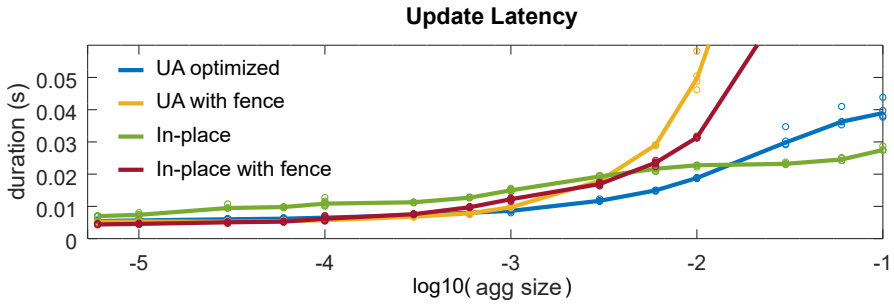


Figure 4.10: Update Latency of Different Bitmap Update Strategies

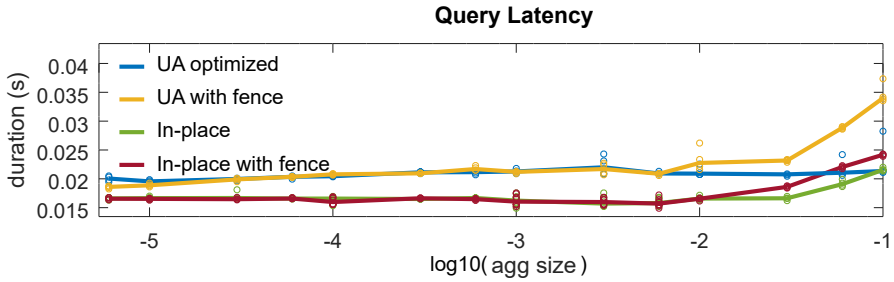


Figure 4.11: Query Latency of Different Bitmap Update Strategies

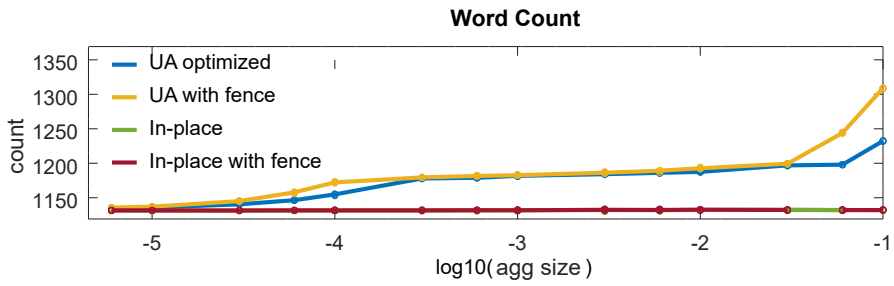


Figure 4.12: Memory Requirement (Number of 64-bit Words) for Different Bitmap Update Strategies

4.5 Chapter Summary

The goal of this chapter has been to investigate the power/performance trade-offs of a particular streaming big data application: Update-aware bitmap indexing with simultaneous read-only queries. The first experiment in this chapter was intended to show that the low power requirements of bitmap index queries mean that the latency of this portion of the workload need not suffer under power cap. We demonstrated this point with a full-featured implementation of a streaming query service with an updateable bitmap index. In the second part of this chapter, we discussed the details of a new kind of update-aware bitmap which is well-suited to handle aggregated updates, and showed that it outperforms the existing state-of-the art indexing strategy.

CHAPTER 5
CONCLUSION

In this thesis, we have investigated the power and performance tradeoffs of three different workloads along dimensions that are relevant to fast power shaping with datacenter loads. Chapters 2 to 4 each present example software applications representing datacenter workloads with progressively more specific QoS requirements. One aspect which the preceding chapters do not cover in detail (though, it was mentioned briefly in Chapter 2) is the question of systematic realtime controller design. We conclude this thesis with a few thoughts and preliminary results on the control aspects of power shaping.

5.1 Ideas on a Controller

Contrasting to relevant control literature such as [85], which employs a complex model predictive control to the problem of datacenter power shaping, our own experience with the wide variability of server power behavior across different workloads makes us curious whether it is possible and in fact advisable to avoid making many assumptions on the power model used to for server power control.

In particular, given that the relationship between the power f and CPU time x is unknown (except that it is increasing), and also given feedback on the workload progress y that a server is making, we are interested to know about how well the simple feedback controller described by Eq. (5.1) can allocate available power (in the sense of job completion time), with α being adjusted by an external integral controller

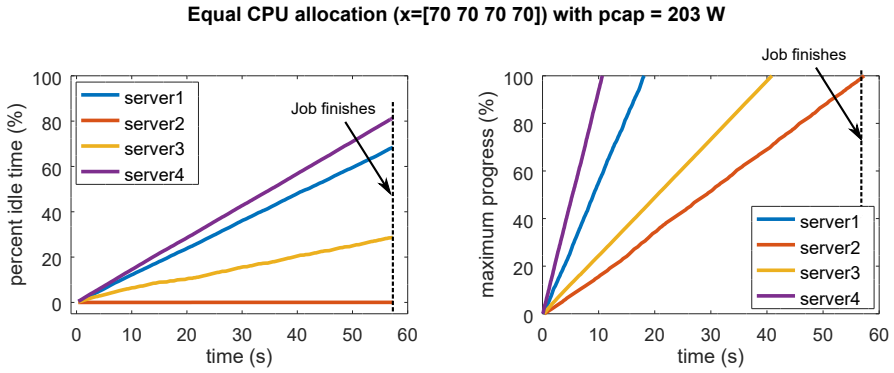


Figure 5.1: Differentiated Workload with Equal Power Assignment

to ensure zero steady-state error.

$$x_i = y_i \alpha \tag{5.1}$$

5.1.1 Preliminary Experimental Results

The situation of interest is one like that experienced in many cluster computing environments, where the tasks assigned to individual servers can proceed in parallel until they finish – at which point they must wait until the slowest server completes before moving to the next stage of computation. As such, we designed a simple experiment to see if such a control strategy we could achieve better performance than equal CPU allocation (as we used in our simple controller of Chapter 1).

We achieved an unequal loading among our four servers by repeatedly running range queries that targeted some nodes more than others. Given that many partitioning schemes intentionally place similar data on the same node, there is a strong possibility of unequal loading among servers – at least on short time scales.

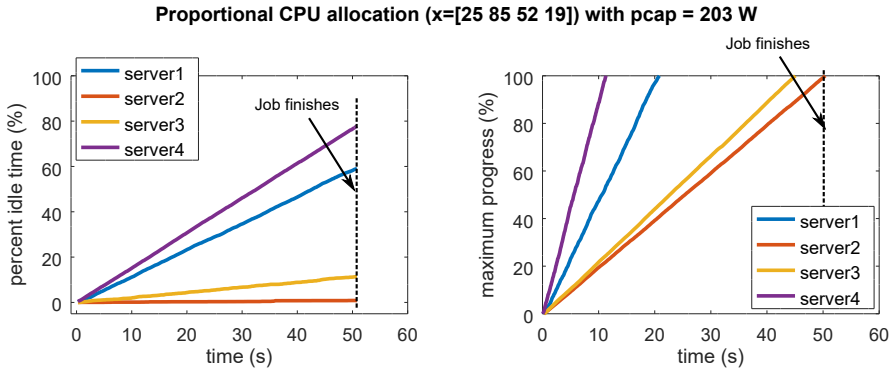


Figure 5.2: Differentiated Workload with Power Allocated Proportionally to Expected Completion Time

Because the workload we ran consisted of many short stages (each finishing in less than a second), the notion of "progress" had to be emulated by taking the inverse of non-idle time spent on each server. In both cases, the power setpoint was 203W. The basline approach using equal CPU allocation among all servers is shown in Fig. 5.1, and completes around 58 seconds. When we set the CPU allocation in proportion to each server's expected completion time once at the very beginning of the job, we did indeed achieve a shorter total completion time, around 50 seconds – even without using feedback.

5.1.2 Inspiration from an Optimization Problem

While thinking about the applicability of this particular feedback strategy, it is interesting to note that it is the solution of the following fairly general optimization problem.

In Eq. (5.2), $f_i(x_i)$ represents the power that server i uses, given x_i percent

CPU utilization.

$$f_i(x) = \begin{cases} g_i(x) & 0 \leq x \leq 1 \\ g_i(1) & x > 1 \end{cases} \tag{5.2}$$

where

$g_i(x)$ is strictly increasing and continuous

Each job is divided into several independent subtasks, one sent to each server. All subtasks must complete in order for the job to finish.

If w_i is the amount of work remaining on server i, and that server is working at x_i percent of its maximum speed s_i, then the estimated time remaining is $\frac{w_i}{s_i x_i}$. Defining constants $y_i = \frac{w_i}{s_i}$, the estimated time of completion of the job is $\max_i \frac{y_i}{x_i}$.

The goal is to minimize this estimated completion time, under a cluster power setpoint constraint P (assuming that constraint is feasible in the sense that $\sum_i f_i(0) \leq P$ and $\sum_i f_i(1) \geq P$). The problem is specified in Eq. (5.3).

$$J = \min_{x_1, x_2, \ldots, x_n} \max_i \frac{y_i}{x_i}$$

subject to

$$\sum_i f_i(x_i) = P \tag{5.3}$$

assuming

$$\sum_i f_i(0) \leq P \leq \sum_i f_i(1)$$

We hypothesize that the optimal value is given in Eq. (5.4).

$$J^* = \frac{1}{\alpha}$$
$$\text{with } \alpha \text{ such that} \tag{5.4}$$
$$\sum_i f_i(\alpha y_i) = P$$

The proof takes two parts. First, we show that there exists exactly one α such that $\sum_i f_i(\alpha y_i) = P$. Next, we show that the solution $x_i = \alpha y_i$ is at least as good as the optimal choice of $x_i = x_i^*$ within the power setpoint constraint space $\sum_i f_i(x_i) = P$.

Existence of Equality Constraint Solution

Show that there exists exactly one α such that $\sum_i f_i(\alpha y_i) = P$.

Let $h_i(\alpha) = f_i(\alpha y_i)$ and let $i^* = \text{argmin}_i y_i$, with $h^* = h_{i^*}(\alpha)$ and $y^* = y_{i^*}$. By definition (5.2), $h^*(\alpha)$ is strictly increasing (SI) for $0 \leq \alpha \leq \frac{1}{y^*}$ and also by definition, all other $h_i(\alpha)$'s are nondecreasing in that range. The sum of an SI function with a nondecreasing function is SI, so $F(\alpha) = \sum_i h_i(\alpha)$ is SI in the range $0 \leq \alpha \leq \frac{1}{y^*}$.

By definition, $F(\frac{1}{y^*}) = \sum_i f_i(\frac{y_i}{y^*})$. Furthermore, by definition of y^* as the minimum y_i, we have $y^* \leq y_i \ \forall i$. This directly implies that $\frac{y_i}{y^*} \geq 1$, and because $f_i(x) = f_i(1)$ for $x > 1$, also means that $f(y_i y^*) = f_i(1)$.

So, $F(\frac{1}{y^*}) = \sum_i f_i(1)$. Since $F(0) = \sum_i f_i(0)$, and $F(\alpha)$ SI and continuous for $0 \leq \alpha < \frac{1}{y^*}$, there must exist exactly one $\alpha \in [0, \frac{1}{y^*}]$ such that $F(\alpha) = P$ for any $\sum_i f_i(0) \leq P \leq \sum_i f_i(1)$.

Minimization under Constraint

With α such that $\sum_i f_i(\alpha y_i) = P$, choosing $x_i = \alpha y_i$ gives $\max_i \frac{y_i}{\alpha y_i} = \frac{1}{\alpha}$. Assume there exists some optimal choice of $x_i = x_i^* \neq \alpha y_i$ with $\sum_i f_i(x_i^*) = P$ that gives a better completion time $\max_i \frac{y_i}{x_i^*} = \frac{1}{\beta} < \frac{1}{\alpha}$.

Since $\frac{1}{\beta}$ is the maximum, $\frac{1}{\beta} \geq \frac{y_i}{x_i^*}\ \forall i$. Further, since $\frac{1}{\alpha} > \frac{1}{\beta}$, we have $\frac{1}{\alpha} > \frac{y_i}{x_i^*} \implies x_i^* \geq \alpha y_i$. Since $f_i(x_i)$ is strictly increasing in the range $0 \leq x_i \leq 1$, then $f_i(x_i^*) > f_i(\alpha y_i) \implies \sum_i f_i(x_i^*) > \sum_i f_i(\alpha y_i)$. Since we have already shown that there must be some α such that $\sum_i f_i(\alpha y_i) = P$, then $\sum_i f_i(x_i^*) > P$, which is a contradiction of x_i^* being in the power setpoint constraint space.

5.2 Summary of Thesis Contributions

In Chapter 1, we motivated the problem of demand response by discussing how datacenters' large, quickly-changing electric loads can be adapted to power shaping. Not all software workloads within datacenters are well-suited for demand response, and we discussed the software features that make strong candidates for load shaping. One workload that has ubiquitous application in today's datacenters is video transcoding. As discussed in Chapter 2, its heavy CPU use and inherent QoS flexibility make video transcoding and workloads with similar properties "low-hanging fruit" to add value to datacenters by participating in demand response. As a way to emphasize this point, we presented a very simple power shaping controller that achieved industry-grade power tracking while performing best-effort transcoding. In Chapters 3 and 4, we discussed the power and performance tradeoffs of differnet big

data query optimizations using bitmap indices – beginning with read-only queries, and finishing with an investigation of queries in a dynamic data environment. Our findings showed that there are several important latency-sensitive "big data" applications which can participate in load shaping without compromising QoS, with an appropriate choice of query implementation.

REFERENCES

[1] J. Koomey, "Growth in data center electricity use 2005 to 2010," Analytics Press, for The New York Times, Report, 2011.

[2] Rich Miller, "The world's largest data centers." [Online]. Available: http://www.datacenterknowledge.com/special-report-the-worlds-largest-data-centers/

[3] Patrick Donovan from Schneider Electric. An overlooked problem: Dynamic power variations. [Online]. Available: http://www.datacenterknowledge.com/archives/2013/06/21/a-commonly-overlooked-problem-dynamic-power-variations-in-data-centers-and-network-rooms/

[4] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad, "Opportunities and challenges for data center demand response," in *Green Computing Conference (IGCC), 2014 International*, Nov 2014, pp. 1–10.

[5] P. Siano, "Demand response and smart grids: A survey," *Renewable and Sustainable Energy Reviews*, vol. 30, pp. 461 – 478, 2014.

[6] G. Ghatikar, V. Ganti, N. Matson, and M. A. Piette, "Demand response opportunities and enabling technologies for data centers: Findings from field studies," 2012.

[7] R. Wang, N. Kandasamy, C. Nwankpa, and D. R. Kaeli, "Datacenters as controllable load resources in the electricity market," in *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ser. ICDCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 176–185.

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

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

[10] H. Chen, M. C. Caramanis, and A. K. Coskun, "The data center as a grid load stabilizer," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2014, pp. 105–112.

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

[12] Z. Liu, I. Liu, S. Low, and A. Wierman, "Pricing data center demand response," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '14. New York, NY, USA: ACM, 2014, pp. 111–123.

[13] H. Chen, B. Zhang, M. C. Caramanis, and A. K. Coskun, "Data center optimal regulation service reserve provision with explicit modeling of quality of service dynamics," in *2015 54th IEEE Conference on Decision and Control*, Dec 2015, pp. 7207–7213.

[14] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data," *The management revolution. Harvard Bus Rev*, vol. 90, no. 10, pp. 61–67, 2012.

[15] V. Marx, "Biology: The big challenges of big data," *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[16] C. A. Mattmann, "Computing: A vision for data science," *Nature*, vol. 493, no. 7433, pp. 473–475, 2013.

[17] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan 1998.

[18] M. M. Waldrop, "The chips are down for moores law," *Nature*, vol. 530, no. 7589, pp. 144–147, 2016. [Online]. Available: http://www.nature.com/doifinder/10.1038/530144a

[19] J. Larus, "Spending moore's dividend," *Commun. ACM*, vol. 52, no. 5, pp. 62–69, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1506409.1506425

[20] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496103

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

[22] I. n. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: ACM, 2013, pp. 51–64.

[23] P. Ranganathan, "Recipe for efficiency: Principles of power-aware computing," *Commun. ACM*, vol. 53, no. 4, pp. 60–67, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721673

[24] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: Leveraging green energy in data-processing frameworks," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 57–70.

[25] J. Wang, L. Feng, W. Xue, and Z. Song, "A survey on energy-efficient data management," *SIGMOD Rec.*, vol. 40, no. 2, pp. 17–23, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2034863.2034867

[26] P. Kumar and L. Thiele, "Cool shapers: Shaping real-time tasks for improved thermal guarantees," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, June 2011, pp. 468–473.

[27] F. Sironi, M. Maggio, R. Cattaneo, G. Del Nero, D. Sciuto, and M. Santambrogio, "Thermos: System support for dynamic thermal management of chip multiprocessors," in *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, Sept 2013, pp. 41–50.

[28] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-aware workload placement and migration in distributed sustainable datacenters," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, May 2014, pp. 307–316.

[29] O. B. Goldman, "The multiplex cost and rate system," *American Institute of Electrical Engineers, Proceedings of the*, vol. 34, no. 5, pp. 941–957, May 1915.

[30] G. Gross and F. D. Galiana, "Short-term load forecasting," *Proceedings of the IEEE*, vol. 75, no. 12, pp. 1558–1573, Dec 1987.

[31] K. Porter and J. Rogers, "Survey of variable generation forecasting in the west," National Renewable Energy Laboratory, Subcontract Report NREL/SR-5500-54457, NREL Technical Monitor: Dr. Kirsten Orwig.

[32] A. Hooshmand, J. Mohammadpour, H. Malki, and H. Daneshi, "Power system dynamic scheduling with high penetration of renewable sources," in *2013 American Control Conference*, June 2013, pp. 5827–5832.

[33] A. D. Lamont, "Assessing the long-term system value of intermittent electric generation technologies," *Energy Economics*, vol. 30, no. 3, pp. 1208 – 1231, 2008.

[34] P. Palensky and D. Dietrich, "Demand side management: Demand response, intelligent energy systems, and smart loads," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, pp. 381–388, Aug 2011.

[35] Midcontinent Independent System Operator, "ACE chart." [Online]. Available: https://www.misoenergy.org/MARKETSOPERATIONS/REALTIMEMARKETDATA/Pages/ACEChart.aspx

[36] R. Walawalkar, S. Fernands, N. Thakur, and K. R. Chevva, "Evolution and current status of demand response (dr) in electricity markets: Insights from PJM and NYISO," *Energy*, vol. 35, no. 4, pp. 1553 – 1560, 2010.

[37] P. Cappers, C. Goldman, and D. Kathan, "Demand response in u.s. electricity markets: Empirical evidence," *Energy*, vol. 35, no. 4, pp. 1526 – 1535, 2010.

[38] I. Goiri, K. Le, M. Haque, R. Beauchea, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: Scheduling energy consumption in green datacenters," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–11.

[39] S. Kumar, P. Pujara, A. Aggarwal, B. Falsafi, and T. N. VijayKumar, *Bit-Sliced Datapath for Energy-Efficient High Performance Microprocessors*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 30–45. [Online]. Available: http://dx.doi.org/10.1007/11574859_3

[40] J. Koomey, "Estimating total power consumption by servers in the U.S. and the world," Analytics Press, Report, 2011.

[41] G. Guzun, J. C. McClurg, G. Caahuate, and R. Mudumbai, "Power efficient big data analytics algorithms through low-level operations," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 355–361.

[42] J. McClurg, J. Hall, and R. Mudumbai, "Fast demand response with datacenter loads," in *Innovative Smart Grid Technologies Conference (ISGT), 2016 IEEE Power Energy Society*, Sept 2016, (accepted).

[43] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Performance evaluation of parallel mpeg-4 video coding algorithms on clusters of workstations," in *Parallel Computing in Electrical Engineering, 2004. PARELEC 2004. International Conference on*, Sept 2004, pp. 354–357.

[44] Amazon, "Amazon AWS elastic transcoder FAQ." [Online]. Available: https://aws.amazon.com/elastictranscoder/faqs/

[45] J. E. Hall, "Distributed control system for demand response by servers," Master's thesis, University of Iowa, 2015.

[46] C. Pilong, "Pjm manual 12: Balancing operations," PJM Interconnection, Manual Revision: 34.

[47] M. S. D. Department, "Pjm manual 28: Operating agreement accounting," PJM Interconnection, Manual Revision: 73.

[48] F. Kong and X. Liu, "A survey on green-energy-aware power management for datacenters," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 30:1–30:38, Nov. 2014.

[49] Amos Waterland, "stress POSIX workload generator." [Online]. Available: http://people.seas.harvard.edu/~apw/stress/

[50] A. Narayan and S. Rao, "Power-aware cloud metering," *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 440–451, July 2014.

[51] K. Li, "Improving multicore server performance and reducing energy consumption by workload dependent dynamic power management," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 122–137, April 2016.

[52] Len Brown, "Ubuntu 14.04 manpages: turbostat." [Online]. Available: http://manpages.ubuntu.com/manpages/trusty/man8/turbostat.8.html

[53] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *Journal of Experimental Social Psychology*, vol. 49, no. 4, pp. 764 – 766, 2013.

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

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

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

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

[58] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 301–312.

[59] Josiah McClurg, "Serverpower repository." [Online]. Available: https://github.com/jcmcclurg/serverpower

[60] A. Cuzzocrea, L. Bellatreche, and I.-Y. Song, "Data warehousing and olap over big data: current challenges and future research directions," in *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*. ACM, 2013, pp. 67–70.

[61] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in mapreduce," *The VLDB Journal*, vol. 23, no. 3, pp. 355–380, Jun. 2014. [Online]. Available: http://dx.doi.org/10.1007/s00778-013-0319-9

[62] G. Chatzimilioudis, A. Konstantinidis, D. Zeinalipour-Yazti, W. Pedrycz, and S.-M. Chen, *Nearest Neighbor Queries on Big Data*. Cham: Springer International Publishing, 2015, pp. 3–22. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08254-7_1

[63] P. O'Neil and D. Quass, "Improved query performance with variant indexes," in *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. ACM Press, 1997, pp. 38–49.

[64] C.-Y. Chan and Y. E. Ioannidis, "An efficient bitmap encoding scheme for selection queries," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 215–226.

[65] N. Koudas, "Space efficient bitmap indexing," in *Proceedings of the Ninth International Conference on Information and Knowledge Management*, ser. CIKM '00. New York, NY, USA: ACM, 2000, pp. 194–201.

[66] D. Rinfret, P. O'Neil, and E. O'Neil, "Bit-sliced index arithmetic," in *ACM SIGMOD Record*, vol. 30. ACM, 2001, pp. 47–57.

[67] M.-C. Wu and A. P. Buchmann, "Encoded bitmap indexing for data warehouses," in *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 220–230.

[68] P. Lu, S. Wu, L. Shou, and K.-L. Tan, "An efficient and compact indexing scheme for large-scale data store," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 326–337.

[69] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: making a yellow elephant run like a cheetah (without it even noticing)," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 515–529, 2010.

[70] G. Guzun and G. Canahuate, "Hybrid query optimization for hard-to-compress bit-vectors," *The VLDB Journal*, pp. 1–16, 2015.

[71] D. Lemire, O. Kaser, and K. Aouiche, "Sorting improves word-aligned bitmap indexes," *Data and Knowledge Engineering*, vol. 69, pp. 3–28, 2010.

[72] G. Guzun, J. Tosado, and G. Canahuate, "Slicing the dimensionality: Top-k query processing for high-dimensional spaces," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XIV*. Springer, 2014, pp. 26–50.

[73] G. Guzun, G. Canahuate, and D. Chiu, "A two-phase mapreduce algorithm for scalable preference queries over high-dimensional data," in *Proceedings of the 20th International Database Engineering & Applications Symposium*, ser. IDEAS, 2016.

[74] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Commun*, vol. 5, 2014.

[75] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: analysis and comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 1, pp. 148–154, Jan 2005.

[76] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[77] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org.proxy.lib.uiowa.edu/citation.cfm?id=2228298.2228301

[78] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, ACM. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[79] Databricks, "Apache spark survey 2016, report," Databricks, Inc, Report, 2016.

[80] R. Ranjan, "Streaming big data processing in datacenter clouds," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78–83, May 2014.

[81] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-btree: an update-conscious parallel directory structure," in *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, Mar 1999, pp. 448–457.

[82] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers*, vol. 51, no. 10, pp. 1124–1140, Oct 2002.

[83] G. Canahuate, M. Gibas, and H. Ferhatosmanoglu, "Update conscious bitmap indices," in *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, July 2007, pp. 15–15.

[84] M. Athanassoulis, Z. Yan, and S. Idreos, "Upbit: Scalable in-memory updatable bitmap indexing," in *ACM SIGMOD International Conference on Management of Data*, 2016.

[85] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, "Ship: Scalable hierarchical power control for large-scale data centers," in *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, Sept 2009, pp. 91–100.