Taming and Controlling Performance and Energy Trade-offs Automatically in Network Applications

Abstract

In this paper, we demonstrate that a server running a single latency-sensitive application can be treated as a black box to reduce energy consumption while meeting an SLA target. We find that when the mean offered load is stable, one can find "sweet spot" settings in the batching of packets (via interrupt coalescing) and controlling the processing rate (DVFS) that represents optimal tradeoffs in the interactions of the software stack and hardware with the arrival rate and composition of requests currently being served. Trying a few combinations of settings on the live system, an example Bayesian optimizer can find settings that reduces the energy consumption to meet a desired tail latency for the current load.

This research demonstrates that: 1) with no software changes, dramatic energy savings (up to 60%) can be achieved across diverse hardware systems if one controls batching and processing rate, 2) specialized research OSes that have been developed for performance can achieve over 2x better energy efficiency than general-purpose OSes, and 3) a controller, agnostic to the application and system, can easily find energy-efficient settings for the offered load that meets SLA objectives.

1 Introduction

Today latency-sensitive cloud applications[1] play a critical role; in many cases, fleets of servers are dedicated to running a single instance of these applications [47, 88, 108, 123, 137]. Researchers have shown that it is worth exploiting techniques to bypass the kernel and design highly specialized software stacks that combine a purpose-built library OS with these applications to improve their performance [4, 13, 64, 71, 86, 89, 90, 100, 102, 103, 107, 110, 112, 113, 132]. As global data center energy use continues to rise [39,48,98,122], it is critical to find ways to meet the challenging requirements of these applications while reducing their energy use.

Studies of latency-sensitive applications have shown that they experience stable mean demand curves. These curves show gradual changes in offered loads over extended periods, ranging from multiple hours to days. Such stability arises from recurring diurnal patterns and use of load balancers [24, 25, 112, 137]. Generally, these studies suggest that for a particular service there exists a stable mean arrival rate and composition of requests over some time scale.

This load stability (i.e. request rates and composition of requests) offers opportunities to meet SLA objectives while reducing energy use. Specifically, queuing theory suggests that the slack between request arrival, service time, and the SLA can be leveraged to improve energy efficiency. For example, induced queuing can amortize per-packet overhead to improve coalescing and processing efficiency [38], and even introduce idle periods in which the system can enter low-energy sleep states [77].

However, for a specific offered load, application, operating system (OS), and hardware, the most energy-efficient way to meet the SLA objective is specific to how the exact combination of software and hardware interacts. For example, queuing and processing rate settings that mimic a "race-to-idle" policy, executing as fast as possible to create the greatest amount of idle time to spend in a deep sleep state (that may flush CPU caches), maybe the right choice. It is, however, also possible that for the combination of hardware and software being used, it is better to choose a setting that mimics a "pace-to-idle" policy, executing more slowly and either entering a light sleep state or not entering a sleep state altogether [72].

Our research adds to the body of work on energy management [22, 72, 77, 93] by demonstrating that one can exploit stability in system behavior to efficiently find queuing and CPU processing rate settings to meet a tail latency target while reducing energy consumption. We explore three basic conjectures:

[1] Examples of latency-sensitive cloud applications include kev-value stores, search, and image and speech recognition. The execution of such applications must often meet a specific performance target expressed as a Service Level Agreement (SLA). A common SLA is a 99% tail latency requirement – Eg. 99% of all requests must be completed within some time limit.

1. There is a combination of queuing and CPU frequencies for a particular offered load and system (application, OS, hardware) that yields "sweet spots" where one can achieve an acceptable latency distribution while significantly reducing energy consumption.

2. Despite complex interactions between software and hardware, the "sweet spot" setting for a system and load are stable and, once found, will continue to yield good behavior *if* queuing and CPU frequencies are fixed (i.e., not dynamically changed by the OS).

3. A system's response to changes in the queuing and CPU frequencies, at a fixed mean offered load, is well-behaved such that it is possible to use a generic black-box search strategy to quickly find a "sweet spot" setting on a running system.[2]

We explore the first two conjectures in an experimental study (§2) on two applications across two distinct OSes: Linux and an OS specialized for latency-sensitive applications (EbbRT [117]). Similar to Co-PI [68], we use existing hardware mechanisms: network interrupt coalescing (ITR-delay [66]) and dynamic voltage frequency scaling [34] (DVFS) to externally sweep queuing and CPU frequency on the server for a fixed set of offered load. Our study explored up to 340 combinations of ITR-delay, and DVFS and found "sweet spots" that both improved performance by 60% while also lowering energy use by 50% (§2.5.1) in closed-loop applications. For open-loop applications (§2.5.2, §2.5.3), these mechanisms can lower energy use by 76% in Linux (in contrast to its *ondemand* DVFS policy)[3] while meeting SLA objectives. We find that the specialized system has not only much better performance but also achieves a **further** 43% reduction in energy. Most importantly, we found that, while the settings differ, the general purpose and specialized system have similar responses to changes, suggesting one could formally capture this common structure.

This common structure led us to the third conjecture – it is plausible to use a generic search strategy to dynamically find energy-efficient ITR-delay, and DVFS settings for a given offered load. We successfully model (§3) our experimental data to capture latency and energy profiles across both OSes. The accuracy of our model fit suggests that a generic black-box-based controller can be used. We then built a prototype controller (§4) using an established machine learning technique, Bayesian optimization [45], and we illustrate its use in exploiting the stable mean demand curve of a publicly available key-value trace [65] to save up to 60% in server energy. Note that the goal of the prototype is to validate that a black box approach is possible; issues like how and when to trigger search are not studied. Finally (§4.3) we demonstrate the generality of our approach, finding savings up to 36% on applications different from our study (Tailbench [70]) and on radically different hardware platforms released almost a decade apart (i.e. Intel E5-2640-released Q1'12 and Ampere ARMv8 released Q4'21) with different interrupt coalescing mechanisms.

[2]Such an approach has the potential to be universal as it operates at runtime on the entire system and does not depend on tables of parameters, prior training, or profiling.

[3]We've studied the available Linux governors and found *ondemand* to the most appropriate for these workloads.

OS	App	Network Loads	Loop	Type
Linux, EbbRT	NetPIPE	64B, 8KB, 64KB, 512KB	Closed	OS
	NodeJS	N/A	Closed	App
	Memcached	200K, 400K, 600K QPS	Open	OS
	Silo	50K, 100K, 200K QPS	Open	App

Table 1: Operating system (OS), application, and network configurations. **Network Loads** reflect mean values: requests-per-second *(QPS)* or message sizes *(KB)*. **Type** indicates whether an application is more reliant on application processing or OS processing.

This work shows that in environments where: 1) dedicated servers are used for critical cloud applications and, 2) there is significant stability (on the order of minutes) in offered load:

1. There are dramatic energy savings possible if one controls queuing and CPU frequency outside the OS for an offered demand; controls that can be applied to a general purpose OS like Linux with no changes.

2. Today's specialized research systems that have been developed for performance achieve dramatically better energy use than general purpose system when run baremetal.

3. A black-box-based controller can be built to exploit the stable demand curves of latency-sensitive applications to find energy-efficient "sweet spots" that are apply across a range of applications, operating systems and hardware.

The rest of our paper is structured as follows: §2 details our study and some key experimental findings, §3 presents a subset of our modeling results as motivation towards the design and evaluation of our controller in §4. We then present related works in §5 and conclude in §6.

2 Energy Study

We designed this study to validate our conjectures that externally manipulating queuing and CPU frequency can yield a diverse space for exploring energy-efficient "sweet spots". To our knowledge, this study is the first to conduct a study of interrupt coalescing, CPU frequency combinations across two distinct OSes running baremetal, and with a variety of network applications shown in table 1.

2.1 Study Setup

Our infrastructure consists of seven nodes, featuring a mix of 16-core Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz with 126 GB RAM and 12-core Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40GHz processors with 256 GB RAM, all equipped with Intel 82599ES 10-Gigabit SFI/SFP+ NICs. The single node used for booting into both baremetal EbbRT and Linux includes a 16-core Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz, 126 GB RAM, and an Intel 82599ES 10-Gigabit SFI/SFP+ NIC. Ensuring hardware parity between Linux and EbbRT, we carefully configured IA-32 Architectural MSRs, processor-specific MSRs (refer to Tables 35-2 and 35-18 in [58]), and NIC features, including disabled direct-cache injection (DCA), enabled receive-side scaling (RSS) for multi-core packet processing distribution, and enabled hardware checksum offloading. We matched NIC transmit and receive descriptors and write-back thresholds for packet transmissions. Additionally, to minimize system noise, hyperthreads and TurboBoost are disabled on all processors. We excluded TurboBoost due to reported energy anomalies when used with different sleep states [77].

2.2 Hardware Mechanisms

We summarize the software and hardware techniques we use to conduct sweeps of static ITR-delay, and DVFS combinations.

2.2.1 Interrupt Coalescing (ITR-delay)

Most modern NICs have a hardware feature to control per-interrupt rates [59, 94] that induce interrupt coalescing. Typically in Linux, these rates are set dynamically by pre-built dynamic policies within their respective device drivers. However, it is possible to set them statically and we use *ethtool* [1] in the Linux study[4]. For EbbRT, we program the NIC directly via its Intel device driver. ITR-delay on Intel NIC's can be programmed in 2 μs increments.

2.2.2 CPU Frequency (DVFS)

DVFS power states (p-states) are features on modern processors that trade-off instruction execution speed for a reduction in energy use [5, 54, 56]. Normally, p-states are set dynamically by a policy governor in Linux [34]. In this study, we disable dynamic DVFS through Linux's `userspace` governor and write directly to the IA32_PERF_CTL MSR register [58] instead. We replicate this in EbbRT by writing to the same register.

[4] *ethtool* is a user tool that maps interrupt coalescing values to appropriate NIC settings

2.3 OS Softwares

We explored two OSes with fundamentally different system designs. This gave us the ability to deepen our understanding of how ITR-delay and DVFS mechanisms impact performance and energy consumption under different system implementations.

2.3.1 Linux

We build a set of application-specific Linux appliances [49, 118] for each of the applications shown in table 1. These appliances are specially constructed to run a RAM-based filesystem and contain only a small set of system libraries and kernel modules required to run their constituent applications. We construct these appliances from a custom 5.5.17 kernel which we built using a modified configuration file for high performance, following suggestions from previous work that studied Linux core operation costs [114]. To reduce scheduling overheads and noise, we pin all applications to physical cores, disable Linux *irqbalance*, and affinitize packet receive interrupts to their respective cores.

2.3.2 EbbRT

Specialized systems aimed at accelerating network applications have seen significant research [4, 13, 64, 71, 86, 89, 90, 100, 102, 103, 107, 110, 112, 113, 117, 132]. However, these systems often overlook importance of their energy efficiency [53, 98, 135]. To explore the performance and energy implications of such a specialized system, we chose EbbRT [117], an open-source platform for building per-application library OSes (around 20K LOC). EbbRT shares properties with these prior systems and employs a run-to-completion, event-driven model in a single execution domain. We developed a network device driver for EbbRT for the network applications to run baremetal on servers with Intel 82599 10 GbE NICs [60] (around 3K LOC).

2.4 Per-Interrupt Log Collection

For the study, we built a per-interrupt logging framework, `intlog` (Acesss to our data and logging scripts can be found at https://anonymous.4open.science/r/intlog-9925), in Linux and EbbRT's network device driver. We collect the following data in the NIC's interrupt handler code: received and transmitted bytes, descriptors, sleep state statistics, and current timestamp via `rdtsc` instruction. Additionally, per-core Intel performance monitoring counters (PMCs) capture hardware statistics every millisecond, including instructions, cycles, and last-level cache misses. We use standard Running Average Power Limit (RAPL) hardware registers to read per-package energy values [57] [5]. Using rack-level energy measurement

[5] The 1 ms rate is due to sampling granularity of RAPL

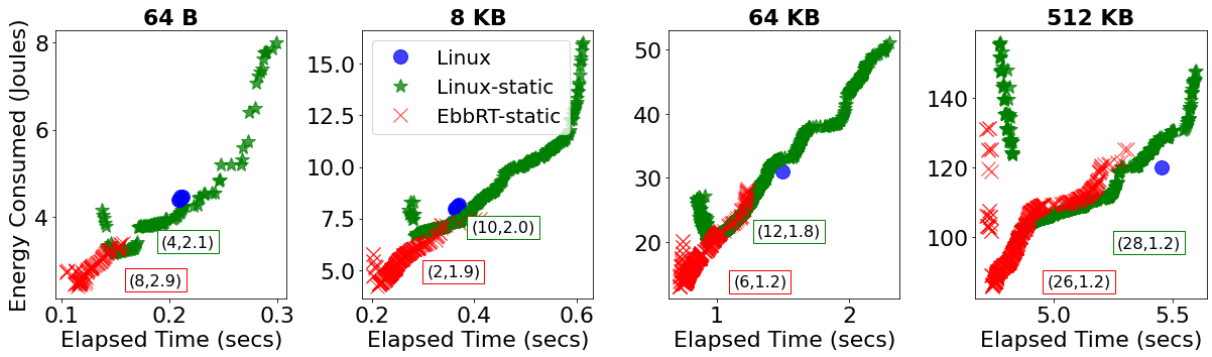

Figure 1: NetPIPE performance and energy results for different message sizes. Every **-static* datapoint is the result of a single experimental run with a unique ITR, DVFS combination while *Linux* has dynamic ITR-delay, DVFS algorithms enabled instead. The X-axis is a measure of performance (lower is better) and Y-axis shows the total energy consumed. For *Linux-static* and *EbbRT-static*, the labeled (ITR-delay, DVFS) pair are experimental values that resulted in the lowest energy use. *Note: The X and Y scales are different to show the structure of collected data.*

mechanisms, we have validated that the changes in energy consumption we observe using RAPL are accurate and impactful[6].

2.5 Study Results

In our results, we compare and contrast the performance and energy consumption achieved by three OS configurations:

1. **Linux**, which has both its dynamic ITR-delay and DVFS algorithms enabled - DVFS is set by Linux `ondemand` governor [34], while ITR-delay is set by Intel's dynamic policy [59])
2. **Linux-static** and **EbbRT-static** where ITR-delay and DVFS are set to specific fixed values.

For both ***-static** OSes, we conducted a study sweeping to 340 [7] static ITR-delay, DVFS pairs, and repeated up to 10 times for stability; our gathered statistics show a standard deviation error of less than 0.01%. In each experiment, we measure a software stack's performance (elapsed time for closed-loop and 99% tail latency for open-loop applications) and overall energy usage. While our study generated over 5 TB of data across multiple runs, we will concentrate on presenting three representative findings based on the results of NetPIPE [119] and memcached [40] experiments in the following sections.

Closed Loop NetPIPE is a simple network ping-pong application of fixed-size messages over a single TCP connection and is an example of a closed loop application [10–12, 39, 87, 92]. For closed-loop applications, the work to be done is a sequence of requests that have an inter-dependency on each other. Linux runs NetPIPE-3.7.1 while EbbRT uses a custom version ported to its interfaces.

Open Loop Memcached is an example of an open-loop application, characterized by an external request rate considered largely independent of the time required for request servicing. In our setup, an unloaded client, running mutilate [63][8], interfaces with five agent nodes generating requests to the memcached server. Each agent node operates on a 16-core machine, with each core establishing 16 connections, resulting in a total of 1280 connections. Linux executes memcached-1.6.6, while EbbRT utilizes a re-implemented version tailored to its native interfaces, supporting the standard memcached binary protocol. We run the representative ETC workload from Facebook [7]. It uses 20 to 70-byte keys and 1-byte to 1-KB values and contains 75% GET requests. We use a stringent SLA objective where the 99% tail latency < 500 μs.

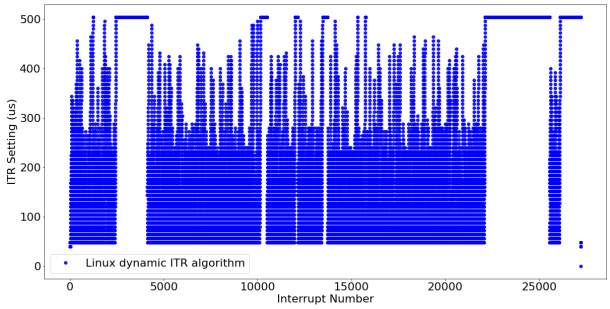

Figure 2: ITR-delay values set by Linux's dynamic ITR-delay algorithm. This is captured during a live run of NetPIPE at 64 KB message size.

[6] While we have validated that ITR-delay and DVFS also impact rack-level energy savings, we use RAPL instead because the granularity of the rack-level measurements (on the order of seconds) made it difficult to attribute detailed energy use to specific system events.

[7] This is due to the experimental scope and also to cover a broad range of possible pairs out of 2 million.

[8] Mutilate is configured to pipeline up to four connections to enhance its request rate.

2.5.1 ITR-delay and DVFS impact on batched packet processing

While the focus of our work is on open-loop style SLA-driven applications, we begin our study with a NetPIPE server. NetPIPE's closed-loop style and simple application protocol allow us to explore how different message sizes, ITR-delay and DVFS affect the overall performance and energy of different OSes. Fig. 1 shows that at 64 KB message size, the static ITR-delay in Linux demonstrates a performance improvement of over 60%, and a 50% reduction in energy consumption compared to dynamic policies in Linux. To understand why this is dramatic, consider the dynamic ITR-delay policy, visualized in fig. 2, which reveals extreme variability at a per-interrupt granularity[9]. This indicates that the current policy, designed for general use cases, operates at an inappropriate timescale for NetPIPE and that significant advantages can be gained through specialization. Moreover, fig.1 illustrates the Pareto-optimal performance-energy curve for various message sizes in both Linux and EbbRT. As the NetPIPE message size increases from 8KB to 64KB and 512KB, the fixed ITR-delay values yielding optimal energy efficiency also increase toward 26μs and 28μs at 512KB for EbbRT and Linux, respectively (labeled in red and green boxes). Intuitively, this result indicates that ITR-delay effectively batches processing by controlling the amount of payload transmitted from the NIC to the OS within a time window. Optimal ITR-delay settings suggest a "sweet spot" where the OS paces packet processing, saving energy through a combination of idling and CPU frequency control.

Lastly, though (ITR-delay, DVFS) pairs in fig. 1 have different values for the different OSes explored, the performance-energy curves for the OSes follow a common 'V' shape. The lowest point in this 'V' shape represents a setting that consumed the least energy while being competitive in performance while the left points represent settings that sacrificed energy for better performance. This 'V' shape also illustrates that it is essential to be strategic in tuning, as while some static settings can outperform dynamic control, there can also be sub-optimal static settings, as shown by points to the right of Linux in Fig. 1.

2.5.2 OS Specialization on Energy and Performance

Next, we consider experiments that explore the performance and energy trade-offs of memcached with varying requests-per-second (QPS) loads under the same SLA objective. Our key findings are that 1) different OS designs can impose different trade-offs between performance and energy, and 2) specialized systems can achieve dramatic efficiency in both. This can be seen in fig. 3 which illustrates the Pareto-optimal curves [10] of EbbRT and Linux. Fig. 3 shows that as QPS increases, EbbRT exhibits a consistent vertical structure, suggesting effective energy savings without impacting latency. Conversely, Linux's curves become more horizontal, indicating performance degradation as QPS rises due to increased trade-offs between performance and energy. Notably, EbbRT's optimized stack allows it to handle higher peak QPS (2000K) compared to Linux (800K). In particular, fig. 4 shows the impact of ITR-delay on the total amount of instructions needed to run a single memcached server in both Ebbrt and Linux. This figure shows how a large ITR-delay (e.g. 400 μs) can reduce overall instruction count by up to 30% in Linux. It also shows the drastic differences in instruction count between the two OSes, as EbbRT uses up to 2.5X fewer instructions to support the same load as Linux does. This implies that a greater fraction of EbbRT's instructions were spent getting actual work done rather than traversing the network stack, which suggests that combining ITR-delay and DVFS control with EbbRT's optimized network paths presents substantial opportunities for maximizing *race-to-idle* energy benefits [22, 93].

2.5.3 Revealing ITR-delay and DVFS Performance-Energy Trade-offs

To help build intuition of the impact of specific ITR-delay and DVFS settings on the performance and energy of open-loop style applications, we present an example in fig. 5 featuring a Linux memcached server with a load of 400K QPS. Using colored gradients, the figure visually represents the effects of each ITR-delay, and DVFS pair; each data point is divided in half, providing insights into their respective impacts on 99% latency and energy.

In fig. 5, one can see the trend that as DVFS decreases from 2.9 GHz: the energy gradient becomes lighter, indicating a more pronounced impact on reducing energy use. Simultaneously, increasing ITR horizontally has an immediate effect on increasing measured 99% latency, evident in the darkening of colored gradients. Further, we observe two notable behaviors at a DVFS frequency of 1.3 Ghz: 1) a fast ITR-delay (0 μs) triggers a spike in tail latency, violating the 500 μs SLA objective due to the slow CPU frequency's insufficient processing of incoming requests, and 2) as ITR-delay increases, this induces additional queuing which enables efficient request batching, thereby facilitating additional energy savings.

These observations indicate that the combination of ITR-delay and DVFS enables one to select different operating points that can move within this space. This is evident in the common "L" shapes seen in fig. 3; which while they differ in absolute performance and energy, illustrates how ITR-delay and DVFS can be combined to reduce energy while still meeting SLA objectives in both OSes.

[9] We instrumented a simple log in Linux's network device driver to save every updated ITR-delay value during a single run of NetPIPE for a 64 KB message.

[10] We filter out (ITR-delay, DVFS) pairs that resulted in SLA violations.

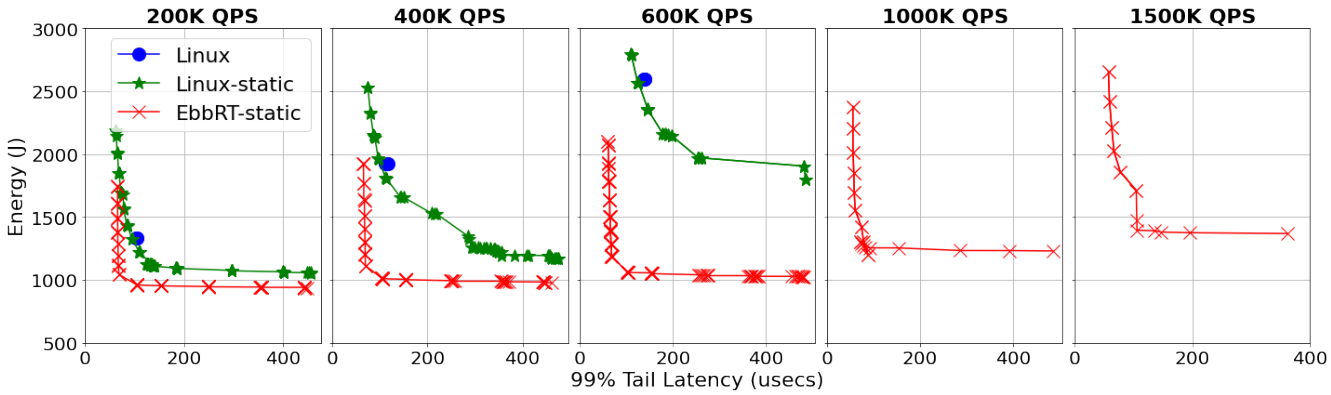

Figure 3: Memcached: Each point represents a single experimental run. The **-static* data points use a unique (ITR-delay, DVFS) pair. We only illustrate data that lie on the Pareto-optimal curve. The X-axis shows performance measurement (lower is better) and the Y-axis shows total energy consumed. *Linux results for 1000K and 1500K QPS loads are not shown as Linux could not support them without violating SLA.*

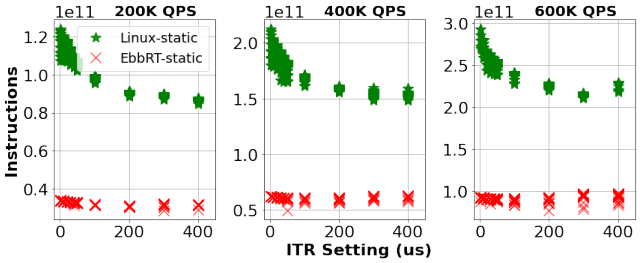

Figure 4: Memcached: ITR-delay impact on instruction count (1*e*11). *Not drawn to scale to show structure in data.*

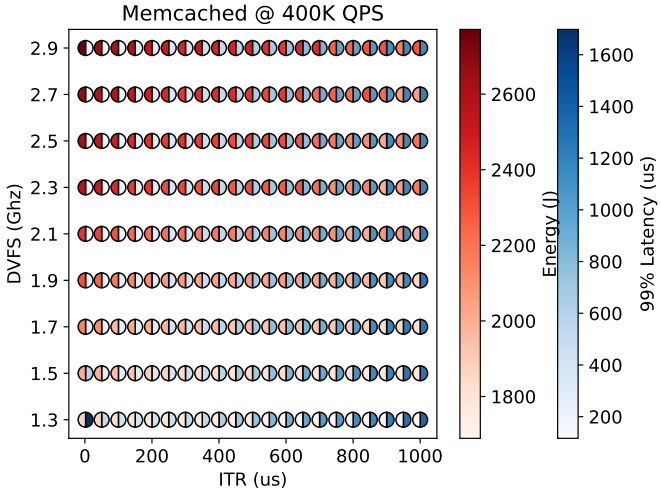

Figure 5: Illustrates the change in energy and 99% latency as different ITR-delay, DVFS pairs are explored for Linux memcached.

3 Modeling ITR-delay and DVFS Effects

A key takeaway from §2 is that fig. 1 and fig. 3 reveal common shapes ("V" and "L") that are OS-agnostic and share a stable structure in response to changes to ITR-delay and DVFS. This suggests one can develop a formal model that captures OS-agnostic performance and energy profiles and that generic external control mechanisms can then be made feasible for both OSes.

3.1 Memcached Model Fitting

Motivated by the implications of these OS shapes, we formulated a mathematical model to explore fitting our experimental data with a set of free parameters and ITR-delay, DVFS settings. We assume a simple model where the offered load is light enough that requests don't batch up in the receive queue and can be treated independently. We model the performance as 99% tail latency as well as energy consumed [11].

3.1.1 Performance:

We define $\triangle t$ as the time it takes to handle a single request:

$$\triangle t = t_{work} + t_{interrupt}$$

We parameterize t_{work} as a function of DVFS values:

$$t_{\text{work}} = \frac{Z}{DVFS^{1+\alpha}} \quad (1)$$

Z and α are free parameters that change for both the OS and application load. In this model, Z acts as a maximum time limit that each request can take (i.e. SLA objective). α represents a system's dependence on DVFS to trade off performance for energy. For example, if $\alpha = -1$, then that particular system has no dependence on DVFS and can largely use DVFS to lower energy use without sacrificing performance - this is inspired by the study results in §2.5.2 that illustrate

[11] We have also collected other tail latency values and found that our model can accurately fit them as well.

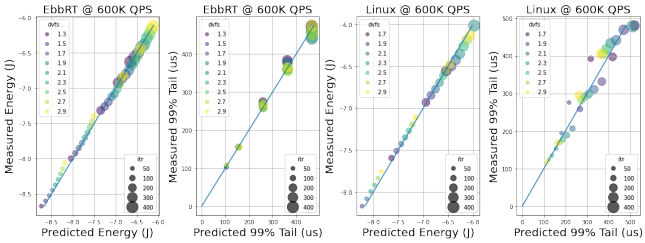

Figure 6: Prediction of energy and performance across both OSes using our model for Memcached @ 600K QPS. The diagonal line indicates a perfect model fit. The negative energy values are due to `log()` transformations during modeling for regression analysis.

how DVFS affects Linux and EbbRT differently in trading off energy for latency.

We parameterize $t_{interrupt}$ as a function of ITR-delay values:

$$t_{\text{interrupt}} = \phi * ITRdelay \tag{2}$$

As ITR-delay greatly affects the measured tail latency, ϕ represents the location in the receive queue where a packet is placed before being processed. For example, if an unlucky packet arrives just as the NIC's ITR value starts counting down, then it will have to artificially wait for a full ITR before being processed, thereby delaying overall request processing time.

3.1.2 Energy:

We define $\triangle J$ as the amount of energy it takes to process a single request. This is affected by the voltage and frequency states of DVFS and how ITR-delay can induce prolonged idle periods:

$$\triangle J = \gamma * (\phi * ITR) * DVFS^{\beta} \tag{3}$$

Note that ϕ used here is the same variable from eq. (2). γ (units of watts) acts to convert the interactions of ITR-delay and DVFS into energy used. The variable β acts as a dependence factor on DVFS in a similar way to α in eq. (1).

Fig. 6 illustrates one example result of the model fitting against memcached data for a QPS of 600K. The x-axis shows the set of energy and performance predictions and the y-axis shows their measured values. The diagonal lines show where ideal points would lie if our model's calculations were exact. We use the Adam optimizer from PyTorch [109] in this process and run each fit several times to check the stability of the inferred parameters and avoid getting stuck in local minima. Overall, we find that despite complicated interactions between the hardware and software, our models are expressive enough to exploit the common OS response behaviors to accurately fit both performance and energy data.

However, the limitation of our model is that it is only practical in highly constrained settings. To replicate this approach for new software and hardware, one would need to exhaustively re-gather data while tuning ITR-delay and DVFS. Nevertheless, the accuracy of our model suggests the viability of using similar approaches to search this space.

4 Proof-of-Concept Controller

Motivated by our prior findings and modeling work, we present an example controller to help validate our conjecture of a black-box search strategy. This controller uses an established machine learning technique, Bayesian optimization [42, 45], to find energy-efficient interrupt coalescing and CPU frequency settings that can adapt to the specific application, OS, and hardware while exploiting the stability in offered loads.

In 1) §4.2, we illustrate its applicability in optimizing the energy efficiency of a server that supports a realistic datacenter workload trace [65] over 24 hours by periodically adjusting ITR-delay, and DVFS settings as offered load changes, and in 2) §4.3 demonstrates the generality of the controller as we apply it across different types of NICs and CPUs (table 2) when run on three applications from Tailbench [70].

As a proof-of-concept controller, we made simplfying assumptions in its design and leave addressing real deployment scenarios for future work. Our assumptions include ad-hoc thresholds for when to trigger Bayesian optimization and the number of subsequent trials to run. However, our results show that even using straightforward assumptions can yield significant advantages, leaving ample room for improvement.

The architecture of our controller also facilitates the integration of more advanced policies for initiating the Bayesian process. We envision the deployment of this technique in data centers through collaboration with load-balancers that make use of historical usage data. This collaboration would help distribute incoming loads to energy-optimized servers, which have been pre-configured with specific settings, while still meeting SLA objectives. In addition, the load balancers can help mitigate the potential gaming of the learning agent's behavior in response to changing request rates.

4.1 Design

Fig. 7 illustrates the design of our controller: (1) A live system running memcached services requests arriving at varying QPSes from an external source. (2) It then triggers a set of performance and energy measurements of the live system to be shared with an external Ax [8, 9] Bayesian optimization platform. (3) This process then computes a penalty Rp of the current (ITR-delay, DVFS) setting and (4) then recommends an update to a new (ITR-delay, DVFS) configuration on the live system that minimizes Rp. Once this process completes, the live system is set with a fixed (ITR-delay, DVFS) configuration until the next set of measurements is triggered.

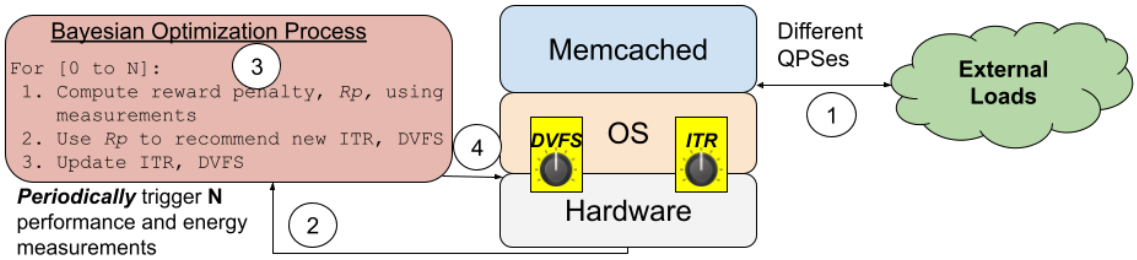

Figure 7: Our controller designed to optimize energy efficiency for a memcached server.

4.1.1 Penalty Function

We use a simple function that penalizes the optimization process by the amount of measured energy and magnifies that penalty when measured latency violates the SLA objective:

$$Rp = m\_energy * max((m\_latency - SLA + 1), 1) \quad (4)$$

For example, with an SLA objective of 99% tail latency < 500 μs, where measured latency (*m_latency*) is 600 μs, the reward *Rp* will be scaled up by a factor of 100, such that $Rp = m\_energy * (600 - 500)$. If *m_latency* is less than *SLA*, then *Rp* will evaluate to *m_energy*. Minimizing *Rp* is indicative that Bayesian optimization is selecting (ITR-delay, DVFS) pairs to meet performance/energy objectives. This reward function enables an operator to express their preference to optimizing for different combinations of energy and performance objectives.

The possibilities of customizing this function further are also ripe for exploration: such as using new combinations of performance/energy or known metrics such as energy-delay-product [15, 51]. One can also imagine developing a rich set of reward functions that capture preferences a service operator might have. In this way, the controller can be reconfigured as priorities change by selecting and tuning from the set of reward functions.

4.2 Applicability to *cache-trace*

This section presents the results of running our controller against a publicly available KV store workload trace (*cache-trace* [65]) which exhibits the stable demand curve behavior for our controller.

4.2.1 Experimental Setup

We used the same infrastructure of our study (§2) but modified the *mutilate* workload generator to generate QPSes following from *cache-trace* instead: first, we extracted a 24-hour sequence of QPS rates from a single trace and binned the data into hourly divisions to capture the mean QPS rate at an hourly basis. As cache-trace QPS rates were often in the tens of thousands of QPS as it was running on limited vCPUs, we scaled up the rates to match our hardware capability. However, 2.5.3 shows that even at low QPS rates where DVFS is fixed at the lowest CPU frequency, ITR-delay can still be used to further reduce energy use. Therefore, we then generate these scaled-up mean QPSes to our live memcached server for which we capture energy-per-second measurements over the entire 24-hour period.

The controller is configured to trigger its periodic measurements at an hourly rate and run Bayesian optimization for a default of 30 trials - this is due to overheads in our single-thread Python package; which takes around 5 minutes to run. In contrast to our initial energy study (§2), which was limited to only using up to 340 (ITR-delay, DVFS) pairs due to experimental scope, our controller allows Bayesian optimization to choose from all available ITR-delay, and DVFS values (a total of *2 million* possible combinations).

We evaluate our controller by comparing the energy and performance behavior of five different system configurations:

- **Linux**: Operating in its default state, where the dynamic ITR-delay and DVFS algorithms are enabled.
- **Linux-BayOp and EbbRT-BayOp**: Operating with Bayesian Optimization to tune both ITR-delay and DVFS, with a target of minimizing overall energy use while maintaining SLA objectives.
- **Linux-DVFS-BayOp and Linux-ITR-BayOp**: Operating with Bayesian Optimization to tune only one of the two settings. We were motivated to explore these configurations to better understand the limitations of the two hardware mechanisms individually. *Linux-DVFS-BayOp* tunes DVFS while enabling the dynamic ITR-delay algorithm. *Linux-ITR-BayOp* tunes ITR-delay while enabling the dynamic DVFS algorithm.

4.2.2 Evaluation

We evaluate our controller's energy impact across two applications, namely memcached and silo, in both Linux and EbbRT[12]. Silo [106, 107] is a compute and memory-intensive application that is extended with a web front-end such that

[12]The controller's penalty can also be modified to minimize latency, details can be found in appendix A

every request triggers a corresponding set of TPC-C transactions on an in-memory database [128]. We ported Silo to EbbRT and the workload mix and SLA constraints of Silo follow from those used in memcached.

4.2.2.1 Memcached Results Fig. 8 illustrates our controllers evaluation against three different SLA objectives: 99% latency < 500 μs, 90% latency < 500 μs, and a even more stringent 99% latency < 200 μs. The QPS values, shown on the right, change on an hourly basis, as shown by black line segments. At the beginning of each hourly QPS change, we see spikes in energy usage of **-BayOp* systems which results from the Bayesian Optimization process searching through (ITR-delay, DVFS) settings on the memcached server to meet its optimization objective. After this initial energy spike, the system settles into a steady energy consumption state until the next hourly trigger. A key result of this application is the importance of using both ITR-delay and DVFS to meet SLA objectives for optimizing energy efficiency rather than individually.

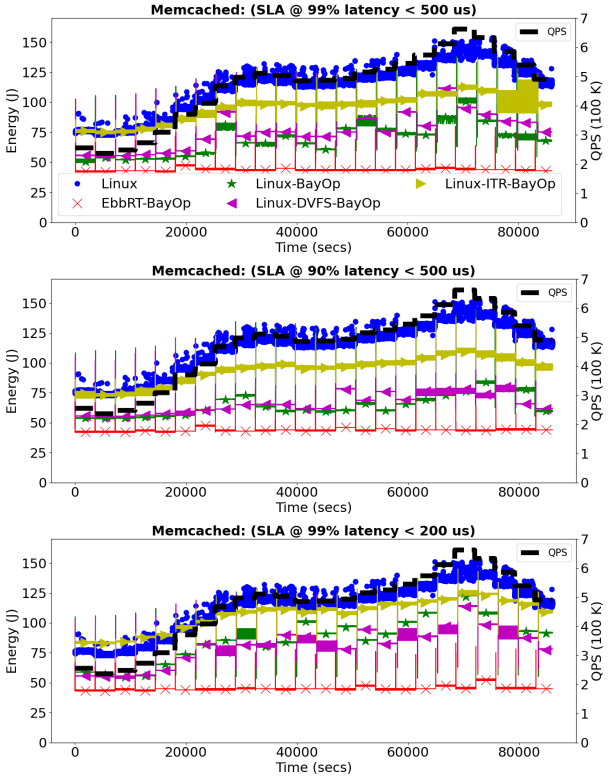

Figure 8: `BayOp` applied to memcached; the **QPS** label is shown on the right side of the graph and the QPS lines show the different offered loads on a per-hour basis. We present results from different SLA objectives studied and illustrate the measured power (energy/second) on the Y-axis as QPS changes across the five system configurations studied over 24 hours (X-axis).

We find that, for an SLA objective of 99% latency < 500 μs, *Linux-BayOp* can result in energy savings of up to 50% over *Linux*. Relaxing the SLA objective to 90% < 500 μs enables our controller to find (ITR-delay, DVFS) configurations that yield even more energy savings of over 60%. At the most stringent SLA of 99% latency < 200 μs, our controller can still adapt while enabling energy savings of up to 30%. The energy savings of *EbbRT-BayOp* are similar to those found in our energy study of memcached (Fig. 3). Our controller is robust enough to adapt to the software stack of EbbRT and find energy-efficient configurations that consistently result in the lowest energy use (over 2X lower than *Linux*). The measured energy-per-second variability of EbbRT is often lower in contrast to that of Linux (indicated by the thinner red plot in Fig. 8), a byproduct of EbbRT's simplified and more optimized network paths.

For *Linux-ITR-BayOp*, allowing our controller to tune only ITR-delay still generally improved energy savings over *Linux*. However, for a stringent SLA of 99% latency < 200 μs, the reduced SLA headroom prevents the controller from trading off latency for energy as effectively as it can when tuning alongside DVFS. At the lower QPSes, *Linux-ITR-BayOp* performed worse than *Linux*.

Allowing our controller to tune only DVFS (*Linux-DVFS-BayOp*) results in energy savings comparable with *Linux-BayOp* across SLA objectives. This is further supported by 2.5.3 which illustrates the significant influence of DVFS on overall energy consumption. However, though it may seem that under a more stringent SLA of 99% latency < 200 μs, *Linux-DVFS-BayOp* results in the highest energy savings, we found instances where the measured 99% latency violated the SLA of 200 μs, as shown in Fig. 10; revealing the weakness of relying on DVFS only.

4.2.2.2 Silo Results We selected another trace from *cache-trace* that was akin to a more computationally intense server. Fig. 9 shows that the trace peak QPS rates are often lower than those of Fig. 8 (peak 250K QPS versus 750K QPS). Fig. 9 does not show results for SLA of 99% latency < 200 μs, as the inherent computational cost of Silo's TPC-C transactions resulted in a lower bound of measured latency values that were consistently greater than the SLA objective of 200 μs. A key result of this application is that it helps expose in computationally intensive cases the limitation of ITR-delay to affect energy savings.

Fig. 9 illustrates that even for a computationally intensive application with different SLA objectives, *Linux-BayOp* was able to find (ITR-delay, DVFS) settings that enable 30% energy savings in Linux for various QPS rates and higher winnings when the SLA is relaxed to 90% latency < 500 μs.

The controller was able to adapt to a different OS and application stack and found configurations of *EbbRT-BayOp* that consistently had the lowest energy use over Linux. In contrast to Fig. 8, one can see larger variations in energy saved from one QPS to the next (more hilly behavior). This can be partly attributed to the complicated database work that

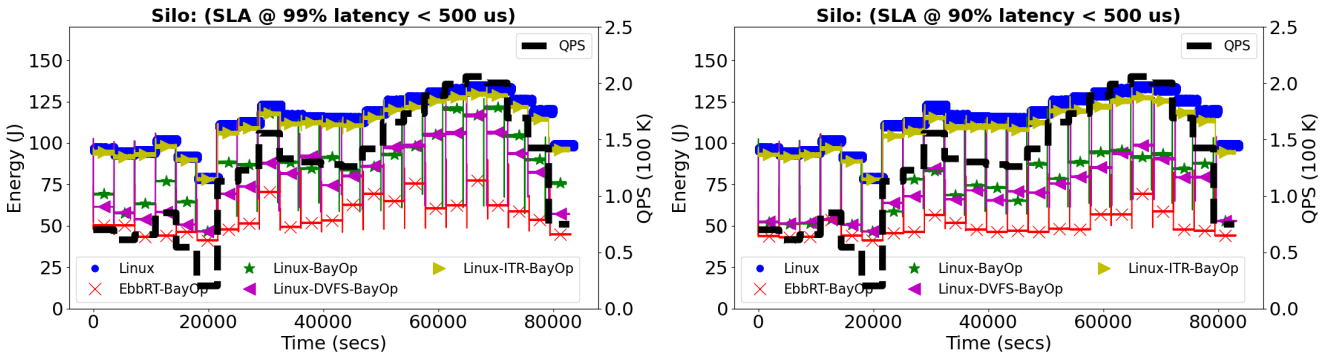

Figure 9: Controller applied to *cache-trace* for Silo. We show two different SLA objectives. The **QPS** line shows the change in QPS offered load on a per-hour basis. The consumed power (energy/second) of each system configuration on the Y-axis is shown over 24 hours on the X-axis.

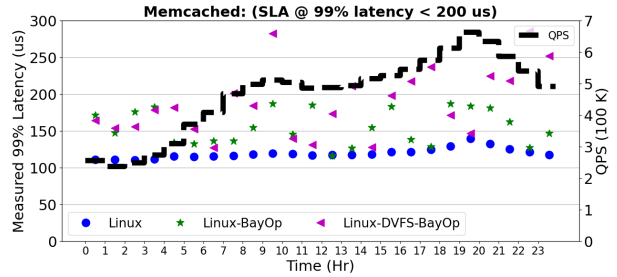

Figure 10: Measured 99% latency across Linux for an SLA of 200 μs. The latency is shown on a per-hour basis due to how *mutilate* reports its resultant latency measurements. We find that *Linux-DVFS-BayOp* often violates the SLA which suggests only tuning DVFS is not enough to achieve stable system behavior.

must now be done per request.

In contrast to memcached, we find that tuning ITR-delay alone (*Linux-ITR-BayOp*) while enabling Linux's default DVFS mechanism is largely ineffective at reducing energy. This is likely due to the increased computational cost for each request which limits the potential energy savings gained from interrupt coalescing and prolonged sleep states that are induced by the ITR-delay mechanism.

We find that tuning DVFS alone (*Linux-DVFS-BayOp*) while enabling Linux's default ITR-delay mechanism works surprisingly well for Silo and, in most cases, achieves a slight energy saving over *Linux-BayOp*. This result suggests an interesting compromise between enabling a degree of energy savings that controlling ITR-delay provides to a computationally-driven network application versus abandoning ITR-delay control so that Bayesian optimization can focus on tuning DVFS to maximize energy savings.

4.3 Black-Box Generality: Diverse Apps and Hardware

In this section, we further demonstrate the versatility of the controller by applying it to optimize energy efficiency for three applications from Tailbench [70]. Our motivation was to reveal how externally controlling interrupt coalescing and CPU frequency can be applied agnostically on hardware even across multi-generational divides[13].

Name	CPU	Cores	NIC	RAM
N0	Intel E5-2640	8	Mellanox 25GbE	62GB
N1	Intel E5-2660	20	Solarflare 10GbE	128GB
N2	AMD EPYC 7452	32	Mellanox 40GbE	128GB
N3	Ampere ARMv8	80	Mellanox 25GbE	124 GB

Table 2: Different hardware explored to run Tailbench.

4.3.1 Experimental Setup

For these experiments, we selected four hardware platforms as shown in table 2. Nodes N0, N1, and N2 are provided by CloudLab [37] and we disable hyperthreads and TurboBoost on all processors to minimize system noise. For each node type, we create a cluster consisting of a single server node, three client nodes that generate traffic to the server node, and an external bootstrap node that launches experiments and runs the `BayOp` controller to tune interrupt coalescing (ITR-delay) and CPU frequency (DVFS) on the server. All of the nodes were running Linux 5.15 kernel; we only examined Linux as EbbRT does not have the necessary device driver support for Solarflare and Mellanox NICs. Notably, while we used `ethtool` to set static interrupt rates across all three NICs in this paper, the fundamental implementation may be different depending on the hardware's capability. On the Intel processors, we use the RAPL hardware registers [26] to report its dynamic energy use while for AMD, we use `amd_energy` hardware monitor driver [97].

Node N3 is another experimental node that runs Linux 6.4.13 but we could only get a single client node to generate traffic[14]. The ARMv8 server provided `xgene-hwmon` [2] tool that enabled us to report its power readings.

[13] Scripts to reproduce results at https://anonymous.4open.science/r/bayop-188B

[14] Due to the computation-heavy nature of Tailbench applications, we found this was still able to saturate the single server

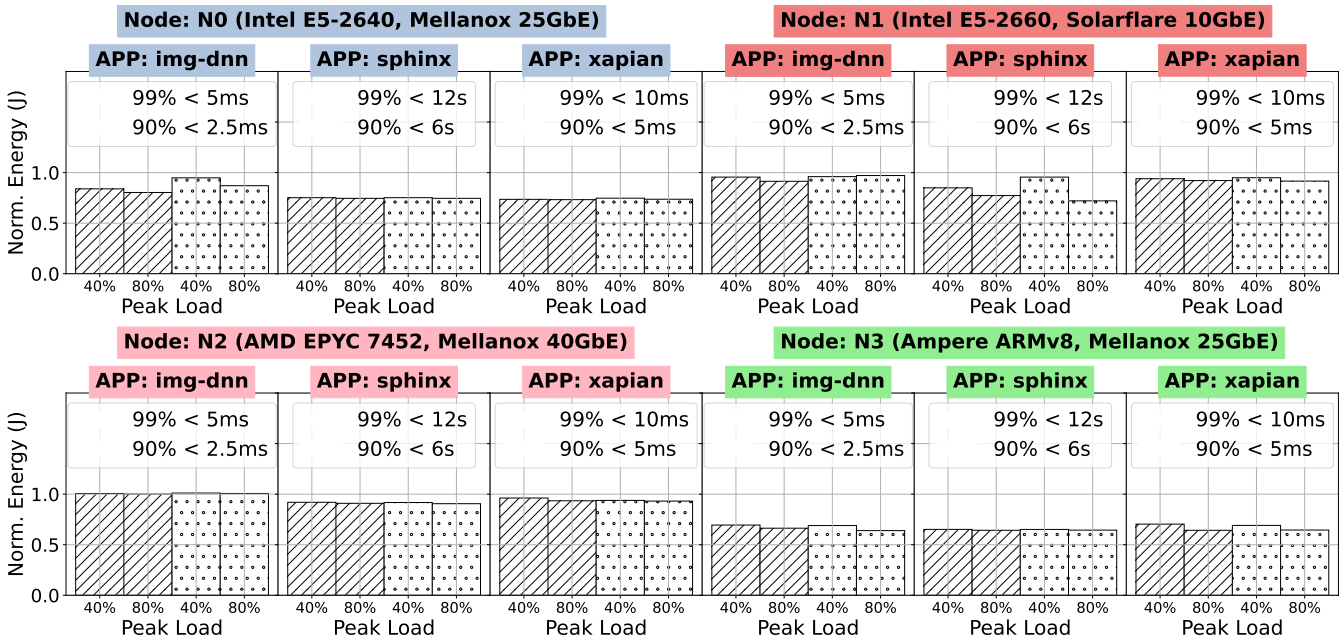

Figure 11: This figure illustrates the energy use of each application (**APP**) for each of the hardware platforms (**NODE: N0 to N3**). The energy is normalized (Y-axis) against Linux default, where lower is better. For each **APP**, we use two representative offered loads which are 40% and 80% of the measured **Peak Load** of Linux default. Within each representative offered load, we also selected two **SLAs** (as indicated by **/////** and **.....**) for the application to meet while our controller is optimizing its energy efficiency.

For each hardware category, we selected applications from Tailbench [70], each designed to fulfill distinct SLA objectives. These applications encompassed **img-dnn**, a handwriting recognition program built on OpenCV; **sphinx**, an open-source search engine; and **xapian**, a speech recognition system. These applications both represent a diverse suite of benchmarks in contrast to the previous examples from our study as well as providing new SLA objectives in the order of milliseconds to seconds. Overall, these selections allowed us to assess the impact of different SLAs and hardware platforms.

4.3.2 Experimental Results

In our experiments, we observed that the controller consistently achieves energy savings ranging from 5% to 36%, depending on the specific combination of software and hardware. Importantly, our findings underscore the fundamental nature of these two mechanisms, which can be effectively applied across a variety of hardware platforms in different SLA-driven application domains. Further, we found that the generic architecture of our controller meant that it was straightforward to simply deploy this technique in new hardware environments as long as it provided support for energy readings and exposed control of interrupt coalescing and CPU frequency.

Fig. 11 depicts the resulting energy consumption for Tailbench; we normalize the energy usage relative to the default Linux configurations under different scenarios:

1. We selected representative offered loads of 40% and 80% of each hardware platform's peak QPS capacity for running the respective applications.

2. For each of these offered loads, we applied two distinct SLA objectives tailored to each application, as indicated by the labels in each figure. These SLA objectives were derived from default values provided by the authors of Tailbench [70].

However, it is worth pointing out that the controller's ability to adapt to applications and offered loads is heavily influenced by the hardware's ability to offer a range of configurations for exploration within this space. One can see an example of this for **APP: img-dnn** in **Node: N2** where it did not manage to find an ITR-delay, DVFS pair that managed to further reduce energy consumption. We hypothesize this stems from a combination of the application type as well as the DVFS settings provided by the AMD EPYC 7452 processor. The processor uses AMD's Collaborative Processor Performance Control (CPPC) interface [54], which is an abstracted performance value that isn't tied to specific a CPU frequency; further, we were limited to only three settings in contrast to the hundreds and thousands available on the other processors. However, this limitation can also be mitigated by newer processors that support the AMD P-state EPP [101] driver, providing finer-grained CPU frequency settings.

To delve into the energy gains we detail the CDF of an example Tailbench application in fig. 12. In this figure, we

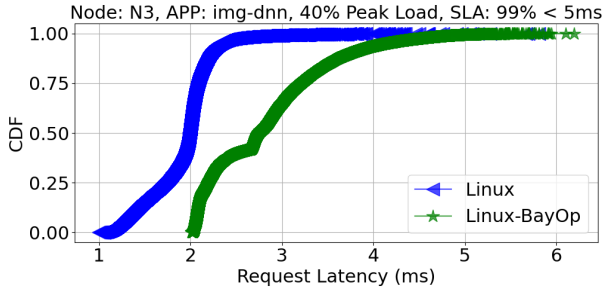

Figure 12: CDF of per request latency between Linux and Linux-BayOp from a single Tailbench application.

illustrate the per-request latency as provided by Tailbench when running the img-dnn application on our ARMv8 server (note this is at a particular peak load and SLA). As the figure shows, the overall request latency of *Linux-BayOP* is about 2X worse than Linux as the controller chose energy-efficient ITR-delay and DVFS settings. While we found Linux was able to support this workload with a 99% latency of 2.8ms, *Linux-BayOp* was still able to meet the SLA at 99% latency of 4.8ms while saving 31% energy.

5 Related Work

Our work falls within a wider space of research on energy proportional computation in datacenters [12, 39, 126]. Much of this research stems from the challenges of improving the performance of network-bound data center workloads [19, 86, 107] while keeping energy consumption at bay. These challenges can be attributed to complex diurnal trends that are characteristic of datacenter-level utilization, whereby idle time is common and must be optimized for [76, 91, 125] while simultaneously maintaining the ability to support high-utilization peaks and strict latency constraints [6, 21, 22, 52, 53, 69, 83, 87, 93, 107, 129, 135, 139]. Our goal was to gain better insight into these impacts on application performance and energy when ITR-delay and DVFS settings are precisely controlled. While prior work has shown how SLA headrooms can be exploited to minimize the overall energy consumption of a system [6, 21, 22, 52, 53, 69, 83, 87, 93, 107, 129, 135, 139], our controller demonstrates this process can be automated and customized on a wider range of hardware and applications than previously shown.

There is a wide range of work that targets energy proportionality with a focus on designing OS policies and mechanisms for power management. Most of this work presents hardware level optimizations that manipulate processor speed mechanisms such as DVFS [23, 34, 38, 41, 43, 46, 62, 74, 75, 77, 81, 82, 85, 116, 121], processor power limiting mechanisms such as RAPL [47, 53, 57, 87, 88, 104, 135], and idle power states [6, 21, 67, 72, 93, 111] (c-states) by applying feedback control mechanisms and relying on activity models. The authors of [88] and [47] go a step further, exploring and characterizing the interference of co-located latency-critical versus best-effort tasks and high versus low CPU demand tasks when subject to energy tuning via DVFS and RAPL. In doing so, they highlight limitations in using hardware features alone for power management. Our work builds on this observation and asserts that specialization in the OS stack also plays a critical role in attaining even more energy efficiency.

Modern hardware components and software stacks expose a large number of parameters that govern internal system operations and interactions. There is a lot of work on defining heuristics to control hardware parameters that impact performance and energy consumption [16, 35, 55, 73, 95, 96]. In recent years, there has been an explosion in using ML-based techniques [30, 134] to uncover more subtle system heuristics for resource management [14, 18, 27, 28, 36, 44, 50, 61, 95, 96, 99, 105, 120, 142], hardware and system configuration [3, 17, 29, 36, 44, 79, 85, 127, 130, 136, 138, 141, 143], high-performance computing [3, 61, 80, 85, 96, 115, 140], and data-center-scale applications [17, 27, 28, 31–33, 84, 124, 130, 131, 138, 143]. Though ML is a natural solution for domains like image, video, and audio processing, the complexity of computer systems often requires extensive expertise to map systems problems to ML tasks. Therefore, prior research has either been limited to simulators [14, 18, 20, 29, 32, 36, 61, 78, 85, 133, 136] or focused only on software parameters only [3, 27, 28, 33, 84, 99, 131, 138, 140, 142, 143]. Instead, our work is the first to apply an ML technique towards exploiting stability in offered loads to find energy-efficient "sweet spots". Our work on finding settings for ITR-delay and DVFS for SLA-driven network applications is most similar to Co-PI [68]. Their approach focuses on the hardware and software specific nature of optimizing ITR-Delay and DVFS on an Intel platform; through off-line profiling, they construct lookup tables indexed base on three coarse gain load categories (low, medium and high). Our work demonstrates how external control of interrupt coalescing and CPU frequency are fundamental mechanisms that can be generally applied across offered load, application, OS, and hardware. Further, we demonstrate how Bayesian optimization can be used to dynamically reduce energy use across a variety of SLA objectives on a live server.

6 Conclusion

Our work seeks to validate a set of conjectures about how combining queuing and processing efficiency can result in diverse set of energy-efficient system settings. Further, to confirm our conjecture that one can exploit stable demand curves in SLA-driven applications; we have also proposed an example controller design that utilizes a black-box search strategy to automatically find these "sweet spots". Our results demonstrate its applicability across offered loads, applications, OSes and even hardware.

References

[1] Improving Measured Latency in Linux for Intel® 82575/82576 or X540/82598/82599 Ethernet Controllers. https://www.intel.com/content/www/us/en/embedded/products/networking/82575-82576-82598-82599-ethernet\-controllers-latency-appl-note.html.

[2] Kernel driver xgene-hwmon. https://docs.kernel.org/hwmon/xgene-hwmon.html.

[3] Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O'Reilly, and Saman Amarasinghe. Siblingrivalry: Online autotuning through local competitions. In *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '12, page 91–100, New York, NY, USA, 2012. Association for Computing Machinery.

[4] Antti Kantee, Justin Cormack. Rump Kernels: No OS? No Problem! https://www.usenix.org/publications/login/october-2014-vol-39-no-5.

[5] ARM. https://developer.arm.com/documentation/den0013/d/Power-Management.

[6] Esmail Asyabi, Azer Bestavros, Erfan Sharafzadeh, and Timothy Zhu. Peafowl: In-application cpu scheduling to reduce power consumption of in-memory key-value stores. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 150–164, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Atikoglu, Berk and Xu, Yuehai and Frachtenberg, Eitan and Jiang, Song and Paleczny, Mike. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[8] Ax:Adaptive Experimentation Platform. https://ax.dev/.

[9] Eytan Bakshy, Lili Dworkin, Brian Karrer, Konstantin Kashin, Benjamin Letham, Ashwin Murthy, and Shaun Singh. Ae: A domain-agnostic platform for adaptive experimentation. 2018.

[10] Luiz Andre Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23:22–28, 2003.

[11] Luiz Andre Barroso and Urs Hoelzle. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[12] Barroso, Luiz André and Hölzle, Urs. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[13] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 49–65, USA, 2014. USENIX Association.

[14] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 318–329, 2008.

[15] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November 2000.

[16] Aaron Carroll and Gernot Heiser. Mobile multicores: Use them or waste them. *SIGOPS Oper. Syst. Rev.*, 48(1):44–48, may 2014.

[17] Chi-Ou Chen, Ye-Qi Zhuo, Chao-Chun Yeh, Che-Min Lin, and Shih-Wei Liao. Machine learning-based configuration parameter tuning on hadoop system. In *2015 IEEE International Congress on Big Data*, pages 386–392, 2015.

[18] Jian Chen and Lizy Kurian John. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *Proceedings of the International Conference on Supercomputing*, ICS '11, page 192–201, New York, NY, USA, 2011. Association for Computing Machinery.

[19] Yanpei Chen, Sara Alspaugh, Dhruba Borthakur, and Randy Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 43–56, New York, NY, USA, 2012. Association for Computing Machinery.

[20] Seungryul Choi and D. Yeung. Learning-based smt processor resource distribution via hill-climbing. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 239–251, 2006.

[21] C. Chou, L. N. Bhuyan, and D. Wong. Î¼dpm: Dynamic power management for the microsecond era. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 120–132, Los Alamitos, CA, USA, feb 2019. IEEE Computer Society.

[22] Chih-Hsun Chou, Daniel Wong, and Laxmi N. Bhuyan. Dynsleep: Fine-grained power management for a latency-critical data center application. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, page 212–217, New York, NY, USA, 2016. Association for Computing Machinery.

[23] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, page 175–185, New York, NY, USA, 2011. Association for Computing Machinery.

[24] Daniel Ellis. https://netflixtechblog.com/ephemeral-volatile-caching-\in-the-cloud-8eba7b124589.

[25] Daniel Ellis. https://web.archive.org/web/20210205121832/https://redditblog.com/2017/1/17/caching-at-reddit/.

[26] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. Rapl: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, page 189–194, New York, NY, USA, 2010. Association for Computing Machinery.

[27] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 77–88, New York, NY, USA, 2013. Association for Computing Machinery.

[28] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 127–144, New York, NY, USA, 2014. Association for Computing Machinery.

[29] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Frederic T. Chong. Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in nvms. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 232–244, New York, NY, USA, 2017. Association for Computing Machinery.

[30] Yi Ding, Nikita Mishra, and Henry Hoffmann. Generative and multi-phase learning for computer systems optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 39–52, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Yi Ding, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. Generalizable and interpretable learning for configuration extrapolation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 728–740, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Yi Ding, Avinash Rao, Hyebin Song, Rebecca Willett, and Henry Hoffmann. Nurd: Negative-unlabeled learning for online datacenter straggler prediction, 2022.

[33] Yi Ding, Alex Renda, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. Cello: Efficient computer systems optimization with predictive early termination and censored regression, 2022.

[34] Dominik Brodowski, Nico Golde, Rafael J. Wysocki, Viresh Kumar. CPU frequency and voltage scaling code in the Linux(TM) kernel. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[35] Han Dong, Sanjay Arora, Yara Awad, Tommy Unger, Orran Krieger, and Jonathan Appavoo. Slowing down for performance and energy: An os-centric study in network driven workloads. https://arxiv.org/abs/2112.07010, 2021.

[36] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F.P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 485–496, 2010.

[37] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh

Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[38] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, page 8, USA, 2003. USENIX Association.

[39] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, page 13–23, New York, NY, USA, 2007. Association for Computing Machinery.

[40] Brad Fitzpatrick. Distributed Caching with Memcached. *Linux Journal*, 2004(124):5, August 2004.

[41] Krisztián Flautner, Steve Reinhardt, and Trevor Mudge. Automatic performance setting for dynamic voltage scaling. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 260–271, New York, NY, USA, 2001. Association for Computing Machinery.

[42] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[43] Vincent W. Freeh, Tyler K. Bletsch, and Freeman L. Rawson. Scaling and packing on a chip multiprocessor. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.

[44] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. A case for machine learning to optimize multicore performance. In *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, HotPar'09, page 1, USA, 2009. USENIX Association.

[45] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

[46] Rong Ge, Xizhou Feng, Wu-chun Feng, and Kirk W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *2007 International Conference on Parallel Processing (ICPP 2007)*, pages 18–18, 2007.

[47] Akhil Guliani and Michael M. Swift. Per-application power delivery. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[48] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing carbon: The elusive environmental footprint of computing, 2020.

[49] Han Dong, Jonathan Appavoo. A Tutorial on Building Custom Linux Appliances. https://www.usenix.org/publications/loginonline/building-linux-appliances.

[50] Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 198–214, New York, NY, USA, 2015. Association for Computing Machinery.

[51] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Proceedings of 1994 IEEE Symposium on Low Power Electronics*, pages 8–11, 1994.

[52] C. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 271–282, 2015.

[53] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 535–548, New York, NY, USA, 2018. ACM.

[54] Huang Rui. amd-pstate CPU Performance Scaling Driver. https://docs.kernel.org/admin-guide/pm/amd-pstate.html.

[55] C. Imes, D. K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *2015 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–86, Los Alamitos, CA, USA, apr 2015. IEEE Computer Society.

[56] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/.

[57] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B:System Programming Guide, Part 2. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/

64-ia-32-architectures-software-developer-\
vol-3b-part-2-manual.pdf.

[58] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3C:System Programming Guide, Part 3. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-\vol-3c-part-3-manual.pdf.

[59] Intel. Tuning Throughput Performance for Intel® Ethernet Adapters. https://www.intel.com/content/www/us/en/support/articles/000005811/network-and-i-o/ethernet-products.html.

[60] Intel 82599 10 Gigabit Ethernet Controller: Datasheet. https://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html.

[61] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. Self-optimizing memory controllers: A reinforcement learning approach. *SIGARCH Comput. Archit. News*, 36(3):39–50, jun 2008.

[62] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, page 347–358, USA, 2006. IEEE Computer Society.

[63] J. Leverich. Mutilate: high performance memcached load generator. https://github.com/leverich/mutilate.

[64] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. mtcp: a highly scalable user-level TCP stack for multicore systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 489–502, Seattle, WA, April 2014. USENIX Association.

[65] Rashmi Vinayak Juncheng Yang, Yao Yue. A large scale analysis of hundreds of in-memory cache clusters at twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, November 2020.

[66] Kan Liang, Andi Kleen, and Jesse Brandenburg. Improve Network Performance By Setting Per-queue Interrupt Moderation In Linux. https://01.org/linux-interrupt-moderation.

[67] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 31–40, 2014.

[68] Ki-Dong Kang, Hyungwon Park, Gyeongseo Park, and Daehoon Kim. Co-adjusting voltage/frequency state and interrupt rate for improving energy-efficiency of latency-critical applications. *IEEE Access*, 8:201028–201039, 2020.

[69] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez. Rubik: Fast analytical power management for latency-critical systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 598–610, 2015.

[70] Harshad Kasture and Daniel Sanchez. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, 2016.

[71] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 67–81, New York, NY, USA, 2016. Association for Computing Machinery.

[72] David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *Proceedings of the 2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, CPSNA '15, page 78–85, USA, 2015. IEEE Computer Society.

[73] David H.K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 78–85, 2015.

[74] Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, 2008.

[75] Masaaki Kondo, Hiroshi Sasaki, and Hiroshi Nakamura. Improving fairness, throughput and energy-efficiency on a chip multiprocessor through dvfs.

SIGARCH Comput. Archit. News, 35(1):31–38, March 2007.

[76] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy Katz. Napsac: Design and implementation of a power-proportional web cluster. *SIGCOMM Comput. Commun. Rev.*, 41(1):102–108, January 2011.

[77] Etienne Le Sueur and Gernot Heiser. Slow down or sleep, that is the question. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, page 16, USA, 2011. USENIX Association.

[78] Benjamin C. Lee and David Brooks. Applied inference: Case studies in microarchitectural design. *ACM Trans. Archit. Code Optim.*, 7(2), oct 2010.

[79] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, page 185–194, New York, NY, USA, 2006. Association for Computing Machinery.

[80] Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '07, page 249–258, New York, NY, USA, 2007. Association for Computing Machinery.

[81] Jungseob Lee and Nam Sung Kim. Optimizing throughput of power- and thermal-constrained multicore processors using dvfs and per-core power-gating. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, page 47–50, New York, NY, USA, 2009. Association for Computing Machinery.

[82] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Server-level power control. In *Fourth International Conference on Autonomic Computing (ICAC'07)*, pages 4–4, 2007.

[83] Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery.

[84] Chi Li, Shu Wang, Henry Hoffmann, and Shan Lu. Statically inferring performance properties of software configurations. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[85] J. Li and J.F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pages 77–87, 2006.

[86] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 429–444, Seattle, WA, April 2014. USENIX Association.

[87] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, page 301–312. IEEE Press, 2014.

[88] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. *SIGARCH Comput. Archit. News*, 43(3S):450–462, June 2015.

[89] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 461–472, New York, NY, USA, 2013. ACM.

[90] Ilias Marinos, Robert N.M. Watson, and Mark Handley. Network stack specialization for performance. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 175–186, New York, NY, USA, 2014. ACM.

[91] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, page 205–216, New York, NY, USA, 2009. Association for Computing Machinery.

[92] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services.

In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, page 319–330, New York, NY, USA, 2011. Association for Computing Machinery.

[93] David Meisner and Thomas F. Wenisch. Dreamweaver: Architectural support for deep sleep. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, page 313–324, New York, NY, USA, 2012. Association for Computing Machinery.

[94] Mellanox. https://community.mellanox.com/s/article/understanding-interrupt-moderation.

[95] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *SIGPLAN Not.*, 53(2):184–198, mar 2018.

[96] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. *SIGPLAN Not.*, 50(4):267–281, mar 2015.

[97] Naveen Krishna Chatradhi . Kernel driver amd_energy. https://www.kernel.org/doc/html/v5.9/hwmon/amd_energy.html.

[98] Nicola Jones. How to stop data centres from gobbling up the world's electricity. https://www.nature.com/articles/d41586-018-06610-y.

[99] Adam J. Oliner, Anand P. Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, New York, NY, USA, 2013. Association for Computing Machinery.

[100] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high cpu efficiency for latency-sensitive datacenter workloads. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 361–377, USA, 2019. USENIX Association.

[101] Perry Yuan. Implement AMD Pstate EPP Driver. https://lwn.net/Articles/914431/.

[102] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 337–350, New York, NY, USA, 2012. Association for Computing Machinery.

[103] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. *ACM Trans. Comput. Syst.*, 33(4), November 2015.

[104] P. Petoumenos, L. Mukhanov, Z. Wang, H. Leather, and D. S. Nikolopoulos. Power capping: What works, what does not. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 525–534, 2015.

[105] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. *SIGARCH Comput. Archit. News*, 41(3):13–23, jun 2013.

[106] George Prekas. https://github.com/ix-project/servers/tree/master, 2017.

[107] George Prekas, Marios Kogias, and Edouard Bugnion. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 325–341, New York, NY, USA, 2017. Association for Computing Machinery.

[108] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 342–355, New York, NY, USA, 2015. Association for Computing Machinery.

[109] pytorch. Adam. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[110] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. Arachne: Core-aware thread management. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 145–160, Carlsbad, CA, October 2018. USENIX Association.

[111] Rafael J. Wysocki. CPU Idle Time Management. https://www.kernel.org/doc/html/v5.0/admin-guide/pm/cpuidle.html.

[112] Rajesh Nishtala and Hans Fugal and Steven Grimm and Marc Kwiatkowski and Herman Lee and Harry C. Li and Ryan McElroy and Mike Paleczny and Daniel Peek and Paul Saab and David Stafford and Tony Tung and Venkateshwaran Venkataramani. Scaling Memcache

at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, Lombard, IL, 2013. USENIX.

[113] Ali Raza, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, and Larry Woodman. Unikernels: The next stage of linux's dominance. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 7–13, New York, NY, USA, 2019. Association for Computing Machinery.

[114] Xiang (Jenny) Ren, Kirk Rodrigues, Luyuan Chen, Camilo Vega, Michael Stumm, and Ding Yuan. An analysis of performance evolution of linux's core operations. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 554–569, New York, NY, USA, 2019. Association for Computing Machinery.

[115] Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. Bliss: Auto-tuning complex applications using a pool of diverse lightweight learning models. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 1280–1295, New York, NY, USA, 2021. Association for Computing Machinery.

[116] Hiroshi Sasaki, Satoshi Imamura, and Koji Inoue. Coordinated power-performance optimization in manycores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, page 51–62. IEEE Press, 2013.

[117] Dan Schatzberg, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo. Ebbrt: A framework for building per-application library operating systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 671–688, GA, 2016. USENIX Association.

[118] Michael W. Shaffer. A linux appliance construction set. In *14th Systems Administration Conference (LISA 2000)*, New Orleans, LA, December 2000. USENIX Association.

[119] Quinn O Snell, Armin R Mikler, and John L Gustafson. Netpipe: A Network Protocol Independent Performance Evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, 1996.

[120] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, page 289–302, New York, NY, USA, 2009. Association for Computing Machinery.

[121] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, page 1–8, USA, 2011. IEEE Computer Society.

[122] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[123] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 787–803. USENIX Association, November 2020.

[124] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30, 2007.

[125] Niraj Tolia, Zhikui Wang, Manish Marwah, Cullen Bash, Parthasarathy Ranganathan, and Xiaoyun Zhu. Delivering energy proportionality with non energy-proportional systems: Optimizing the ensemble. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, page 2, USA, 2008. USENIX Association.

[126] Niraj Tolia, Zhikui Wang, Manish Marwah, Cullen Bash, Parthasarathy Ranganathan, and Xiaoyun Zhu. Delivering energy proportionality with non energy-proportional systems—optimizing the ensemble. In *Workshop on Power Aware Computing and Systems (HotPower 08)*, San Diego, CA, December 2008. USENIX Association.

[127] Erik Tomusk, Christophe Dubach, and Michael O'boyle. Four metrics to evaluate heterogeneous multicores. *ACM Trans. Archit. Code Optim.*, 12(4), nov 2015.

[128] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of*

the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13, page 18–32, New York, NY, USA, 2013. Association for Computing Machinery.

[129] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar. Timetrader: Exploiting latency tail to save datacenter energy for online search. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, page 585–597, New York, NY, USA, 2015. Association for Computing Machinery.

[130] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1009–1024, New York, NY, USA, 2017. Association for Computing Machinery.

[131] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. *Understanding and Auto-Adjusting Performance-Sensitive Configurations*, page 154–168. Association for Computing Machinery, New York, NY, USA, 2018.

[132] Matt Welsh, David Culler, and Eric Brewer. Seda: An architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, October 2001.

[133] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 29–39, 2010.

[134] Nan Wu and Yuan Xie. A survey of machine learning for computer architecture and systems. *ACM Computing Surveys*, 55(3):1–39, apr 2023.

[135] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's data center-wide power management system. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 469–480, Piscataway, NJ, USA, 2016. IEEE Press.

[136] Weidan Wu and Benjamin C. Lee. Inferred models for dynamic and sparse hardware-software spaces. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 413–424, 2012.

[137] Juncheng Yang, Yao Yue, and K. V. Rashmi. A large scale analysis of hundreds of in-memory cache clusters at twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 191–208. USENIX Association, November 2020.

[138] Nezih Yigitbasi, Theodore L. Willke, Guangdeng Liao, and Dick Epema. Towards machine learning-based auto-tuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 11–20, 2013.

[139] Xin Zhan, Reza Azimi, Svilen Kanev, David Brooks, and Sherief Reda. Carb: A c-state power management arbiter for latency-critical workloads. *IEEE Computer Architecture Letters*, 16(1):6–9, 2017.

[140] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 545–559, New York, NY, USA, 2016. Association for Computing Machinery.

[141] Yanqi Zhou, Henry Hoffmann, and David Wentzlaff. Cash: Supporting iaas customers with a sub-core configurable architecture. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 682–694, 2016.

[142] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24, 2013.

[143] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 338–350, New York, NY, USA, 2017. Association for Computing Machinery.

A Appendix

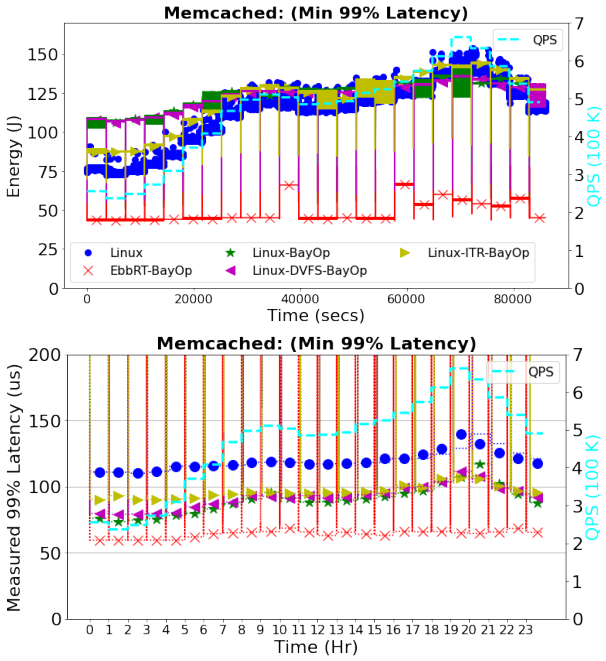

Figure 13: Controller applied to optimize **only** for minimizing 99% tail latency for memcached. We show show the energy per second consumption on the left figure and the measured latency on the right.

A.1 Optimizing for Latency in Memcached

In fig. 13 we demonstrate the flexibility of the controller's optimization criteria through a change of its reward penalty function to focus on minimizing tail latency instead at a cost to greater energy use. In this case, the function is simplified to $Rp = m\_latency$ in order to reflect performance optimization instead. Fig. 13 illustrates that the Bayesian process was also able to consistently performance-focused ITR-delay, DVFS settings that lowered the 99% tail latency by up to 30% in Linux, however at a higher energy cost of up to 40%. This result also demonstrates the potential of exploring alternate reward functions that may consist of different combinations of performance and energy criteria.

A.2 Optimizing for Latency in Silo

Fig. 14 shows that similar to memcached, the controller can still largely lower overall 99% latency by 50 % while increasing its overall energy use for a new application across both OSes.

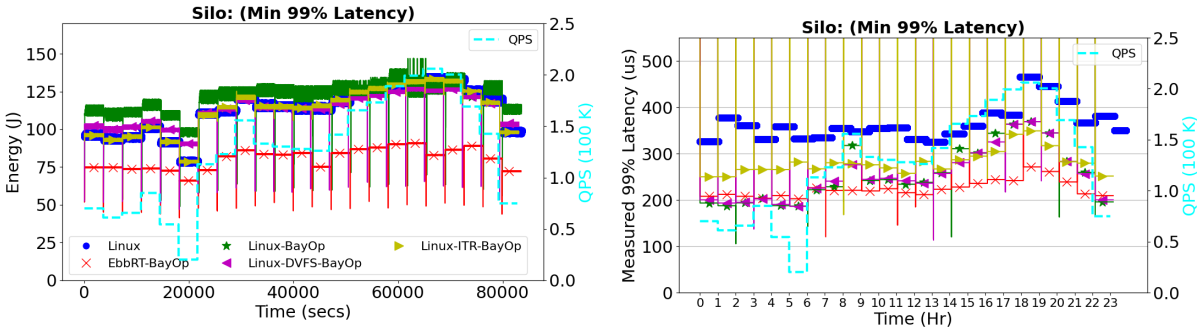

Figure 14: Controller applied to optimize **only** for minimizing 99% tail latency in Silo. We show show the energy per second consumption on the left figure and the measured latency on the right.