Cloud Computing and Data Science
https://ojs.wiserpub.com/index.php/CCDS/

UNIVERSAL WISER PUBLISHER

Research Article

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

Han Dong[1*], **Sanjay Arora**[2], **Yara Awad**[3], **Orran Krieger**[3], **Jonathan Appavoo**[3]

[1]Department of Computer Science, Hamilton College, 198 College Hill Road, Clinton, NY, 13323, USA
[2]Red Hat Inc., 300 A St, Boston, MA, 02210, USA
[3]Department of Computer Science, Boston University, Commonwealth Ave, Boston, MA, 02215, USA
E-mail: hdong@hamilton.edu

Received: 28 October 2025; **Revised:** 15 January 2026; **Accepted:** 3 February 2026

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 a Service-Level Agreement (SLA) target. We find that it is possible to identify "sweet spot" settings for packet batching and processing rate control. These settings represent optimal trade-offs between the software stack and hardware. Specifically, they account for both the arrival rate and the composition of requests being served. By testing a few combinations of these settings on the live system, a proof-of-concept controller can dynamically find settings that reduce energy consumption while meeting a desired tail latency for the request rate. Our work demonstrates three key findings. First, without software changes, energy savings of up to 60% are achievable across diverse hardware systems by controlling batching and processing rates. Second, specialized research Operating Systems (OSes) can leverage this to achieve a further 40% energy savings over general-purpose OSes. Finally, we show that a controller that is agnostic to the application, system, and hardware, can find energy-efficient settings for different request rates while meeting performance objectives.

***Keywords*:** operating systems, distributed systems, experimental measurements, energy efficiency, applied machine learning

1. Introduction

Latency-sensitive cloud services play a critical role in the interactivity of many user-facing applications. Supporting these services requires a growing fleet of servers which represent hundreds of clusters that encompass hundreds of thousands of cores. The significance of this work lies in its practical and effective approach to tackle important problems in modern data center computing. This work addresses the challenges of improving the sustainability of data centers nodes that run these applications by creating a method to help reduce their energy consumption. The proposed method works by controlling existing hardware mechanisms externally, without requiring changes to the application code or the operating system. The experimental results, demonstrated through a diverse set of hardware clusters and using real world experimental benchmarks, help illustrate that Linux's existing default policies are inefficient for these data center workloads, and the proposed approach is able to save over 60% energy in comparison. The key innovation of our approach is the use of a black-box machine learning-based algorithm that can dynamically adapt to different systems,

DOI: https://doi.org/10.37256/ccds.7220269014

workloads, and request rates. This helps to eliminate the need for manual, application-specific tuning or offline profiling, therefore, the system can adapt and optimize itself automatically in a live environment. By proving the method works on diverse hardware almost a decade apart, our results show that this is a fundamental and widely applicable technique and provides a universal strategy for energy management in data center applications.

Today latency-sensitive cloud applications play a critical role; in many cases, fleets of servers are dedicated to running a single instance of these applications such as kev-value stores, search, and image and speech recognition [1, 2]. 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—E.g. 99% of all requests must be completed within some time limit. 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 Operating System (OS) with these applications to improve their performance [3-5]. As global data center energy use continues to rise [6-9], 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 request rates over extended periods, ranging from multiple hours to days. Such stability arises from recurring diurnal patterns and use of load balancers [2, 10]. 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 application stability (i.e., comprising of 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 [11], and even introduce idle periods in which the system can enter low-energy sleep states [12].

However, for a specific request rate, application, 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 Central Processing Unit (CPU) caches), maybe the right choice. It is also possible that a specific combination of hardware and software favors a "pace-to-idle" policy. In this approach, the system executes more slowly and either enters a light sleep state or avoids entering a sleep state altogether [13].

Our research adds to the body of work on energy management [14, 15] 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. There is a combination of queuing and CPU frequencies for a particular request rate and system (application, OS, hardware) that yields "sweet spots" where one can achieve an acceptable latency distribution while reducing energy consumption.

2. Despite complex interactions between software and hardware, the "sweet spot" setting for a system and requests 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 request rate, 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. 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.

We explore the first two conjectures in an experimental study (§3) on two applications across two distinct Operating Systems (OSes): Linux and an OS specialized for latency-sensitive applications (Elastic building block Runtime (EbbRT) [16]). We use existing hardware mechanisms: network Interrupt delay (ITR-delay) [17] and Dynamic Voltage Frequency Scaling (DVFS) [18] to externally sweep queuing and CPU frequency on the server for a fixed set of request rates. 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% (§3.5.1) in closed-loop applications. For open-loop applications (§3.5.2, §3.5.3), these mechanisms can lower energy use by 76% in Linux (in contrast to its ondemand DVFS policy) while meeting SLA objectives. We've studied the available Linux governors and found ondemand to the most appropriate for these workloads. 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 request rate. We successfully model (§4) 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 (§5) using an established machine learning technique, Bayesian optimization [19], and we illustrate its use in exploiting the stable mean demand curve of a publicly available Key-Value (KV) trace [20] 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 (§5.3) we demonstrate the generality of our approach, finding savings up to 36% on applications different from our study (Tailbench [21] and on radically different hardware platforms released almost a decade apart (Intel E5-2640-released Q1'12 and Ampere ARMv8 released Q4'21) with different interrupt coalescing mechanisms.

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

1. There are possible energy savings of up to 60% 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 energy savings of up to 40% compared to general purpose systems 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: We present related works in §2, then §3 details our study and some key experimental findings, §4 presents a subset of our modeling results as motivation towards the design and evaluation of our controller in §5. We then discuss future work in §6 and conclude in §7.

2. Related works

Our work falls within a wider space of research on energy proportional computation in datacenters [22]. Much of this research stems from the challenges of improving the performance of network-bound data center workloads [23, 24] while keeping energy consumption at bay. These challenges can be attributed to complex diurnal trends that are characteristic of datacenter-level utilization. In these environments idle time is common and must be optimized for [25]. Simultaneously, these systems need to maintain the ability to support high-utilization peaks and strict latency constraints [24, 26]. 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 headroom's can be exploited to minimize the overall energy consumption of a system [26, 27], 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 [28], processor power limiting mechanisms such as Running Average Power Limit (RAPL) [29], and idle power states [25, 27] (c-states) by applying feedback control mechanisms and relying on activity models. The authors of [30, 31] 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 finds 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 [32-34]. In recent years, there has been an explosion in using Machine Learning (ML)-based techniques [35, 36] to uncover more subtle system heuristics for resource management [37], hardware and system configuration [38], high-performance computing [39], and data-center-scale applications [40]. 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 [41] or focused only on software parameters only [37, 39]. Instead, our work is the first to apply an ML technique towards exploiting stability in request rates to find energy-efficient "sweet spots" in different software and hardware.

Our work on finding settings for ITR-delay and DVFS for SLA-driven network applications is similar to Co-PI [34]. Co-PI focused on the hardware and software specific nature of optimizing ITR-Delay and DVFS on an Intel platform through off-line profiling. In contrast our work demonstrates further improvements by illustrating how external control of interrupt coalescing and CPU frequency are novel mechanisms that can be applied across different request rates, applications, OSes, and hardware through the use of a black-box tuning approach.

3. Energy study

We begin with an energy 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.

Table 1. Operating System (OS), Application (App), 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

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

3.1 *Study setup*

Our infrastructure consists of seven nodes, featuring a mix of 16-core Intel(R) Xeon(R) CPU E5-2690 @ 2.90 GHz with 126 GB Random-Access Memory (RAM) and 12-core Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40 GHz processors with 256 GB RAM, all equipped with Intel 82599ES 10-Gigabit SFI/SFP+ Network Interface Cards (NICs). The single node used for booting into both baremetal EbbRT and Linux includes a 16-core Intel(R) Xeon(R) CPU E5-2690 @ 2.90 GHz, 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 Model Specific Registers (MSRs), processor-specific MSRs (refer to Tables 35-2 and 35-18 in [42]), and NIC features, including disabled Direct-Cache Access (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 [12]. Below, we first discuss the two hardware settings explored in our study to manipulate queuing (ITR-delay) and CPU frequency (DVFS).

3.2 *Hardware mechanisms*

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

3.2.1 *Interrupt delay (ITR-delay)*

Most modern NICs have a hardware feature to control per-interrupt rates 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 in the Linux study. 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.

3.2.2 *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. Normally, p-states are set dynamically by a policy governor in Linux [18]. In this study, we disable dynamic DVFS through Linux's userspace governor and write directly to the IA32_PERF_CTL MSR register instead. We replicate this in EbbRT by writing to the same register.

3.3 *OS software*

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.

3.3.1 *Linux*

We build a set of application-specific Linux appliances [42, 43] 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 [44]. 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.

3.3.2 *EbbRT*

Specialized systems aimed at accelerating network applications have seen significant research [45]. However, these systems often overlook importance of their energy efficiency. Prior work [46] has illustrated how the optimized instruction code paths of specialized library OSes can both improve performance while reducing energy use in applications. To explore the performance and energy implications of such a specialized system, we chose EbbRT [16], an open-source platform for building per-application library OSes (around 20 K Lines of Code (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 (around 3 K LOC).

3.4 *Per-interrupt log collection*

For the study, we built a per-interrupt logging framework, intlog (Access to our data and logging scripts can be found at https://github.com/SESA/intlog), 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. The 1 millisecond (ms) rate is due to sampling granularity of RAPL. Using rack-level energy measurement mechanisms, we have validated that the changes in energy consumption we observe using RAPL are accurate and impactful. 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.

3.5 *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 [18], while ITR-delay is set by Intel's dynamic policy [17].

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 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%. We selected 340 pairs to study due to the experimental scope and also to cover a broad range of possible pairs out of 2 million. 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 [47] and Memcached [48] experiments in the following sections:

Closed Loop: NetPIPE is a simple network ping-pong application of fixed-size messages over a single Transmission Control Protocol (TCP) connection and is an example of a closed loop application [49]. 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 [50], interfaces with five agent nodes generating requests to the Memcached server. Mutilate is configured to pipeline up to four connections to enhance its request rate. Each agent node operates on a 16-core machine, with each core establishing 16 connections, resulting in a total of 1,280 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 [50]. 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.

3.5.1 *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. Figure 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. The reason for this dramatic performance can be seen in Figure 2, which shows how the existing dynamic policy reveals extreme variability at a per-interrupt granularity. We illustrate this by instrumenting 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. Our results indicate that the current policy, designed for general use cases, operates at an inappropriate timescale for NetPIPE and that more advantages can be gained through specialization. Moreover, Figure 1 illustrates the pareto-optimal performance-energy curve for various message sizes in both Linux and EbbRT. As the NetPIPE message size increases from 8 KB to 64 KB and 512 KB, the fixed ITR-delay values yielding optimal energy efficiency also increase toward 26 μs and 28 μs at 512 KB 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.

Figure 1 shows different (ITR-delay, DVFS) pairs for the different OSes explored. Further, one can see similar performance-energy curves for the OSes that follow a common 'V' shape. The lowest point in this 'V' shape represents the setting that consumed the least energy while remaining competitive in performance. Conversely, the points to the left represent settings that sacrificed energy to achieve even better performance. This 'V' shape illustrates that strategic tuning is essential. While some static settings can outperform dynamic control, there are also sub-optimal static settings, as shown by the points to the right of Linux in Figure 1.

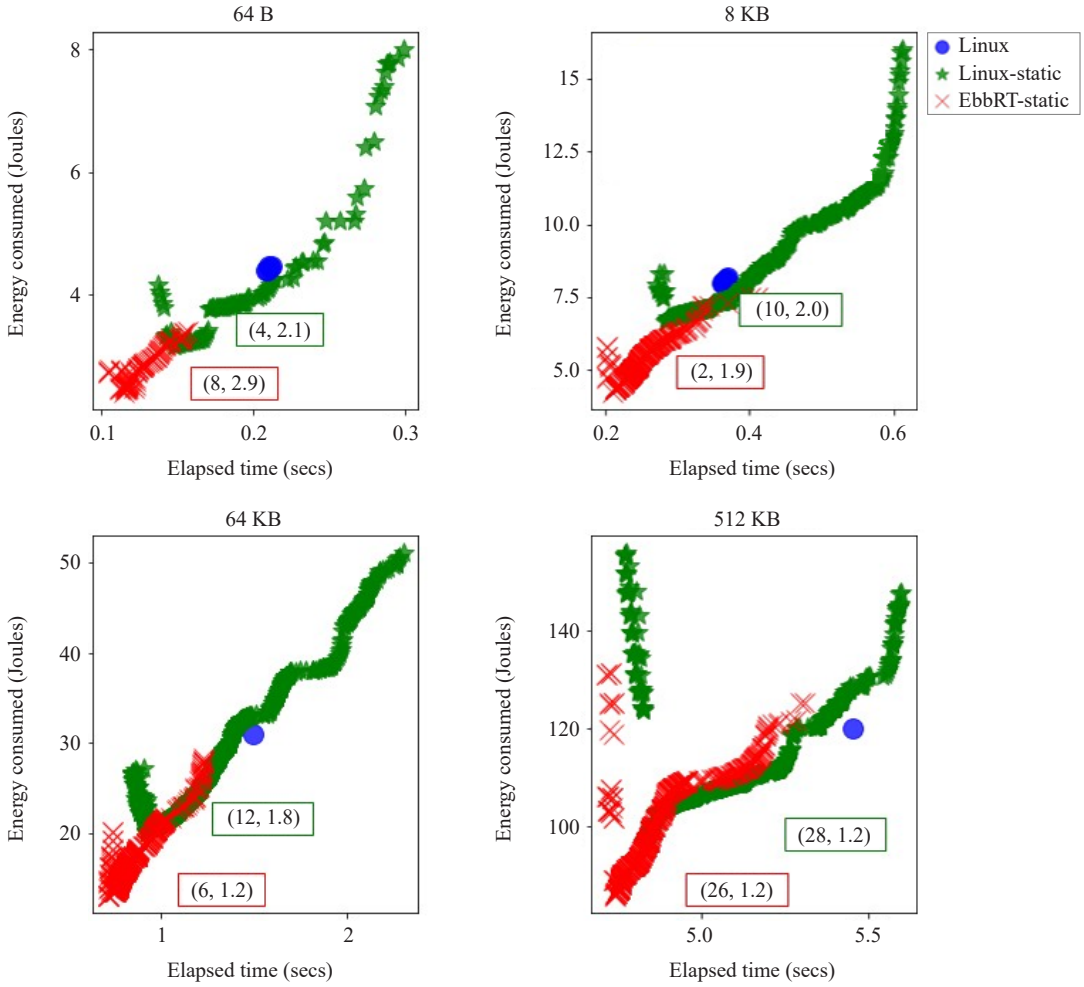


Figure 1. NetPIPE performance and energy results for different message sizes. Every *-static datapoint is the result of a single experimental run with a unique ITR, DVFS combination while Linux has dynamic ITR-delay, DVFS algorithms enabled instead. The X-axis is a measure of performance (lower is better) 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

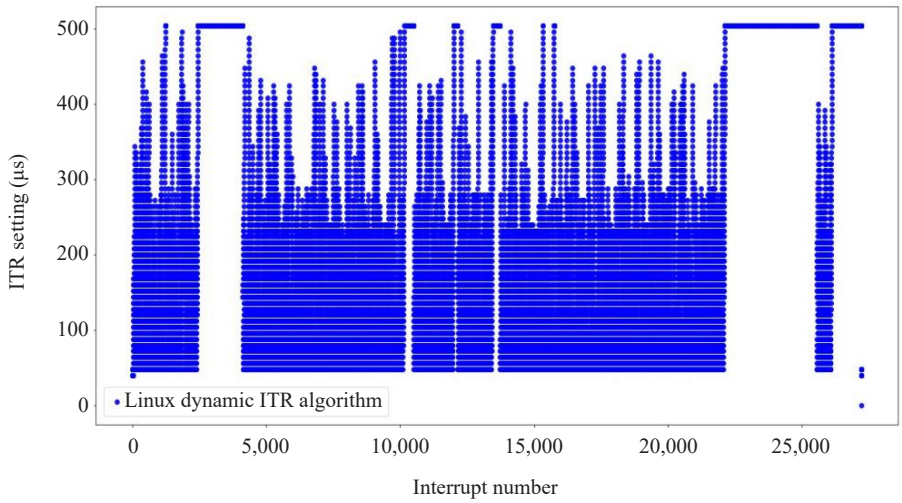


Figure 2. ITR-delay values set by Linux's dynamic ITR-delay algorithm. This is captured during a live run of NetPIPE at 64 KB message size

3.5.2 *OS specialization on energy and performance*

Next, we discuss experiments that explore the performance and energy trade-offs of Memcached with varying reQuests-Per-Second (QPS) rates under the same SLA objective. In this section, our key findings are that 1) different OS designs can impose different trade-offs between performance and energy, and 2) specialized systems can achieve better efficiency in both. This can be seen in Figure 3 which illustrates the pareto-optimal curves of EbbRT and Linux. We filter out (ITR-delay, DVFS) pairs that resulted in SLA violations. Figure 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. In particular, Figure 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, 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 request rate as Linux does. This implies that a greater fraction of EbbRT's instructions were spent on actual work rather than traversing the network stack. This efficiency suggests that combining ITR-delay and DVFS control with EbbRT's optimized paths presents opportunities for maximizing race-to-idle energy benefits [51-53].

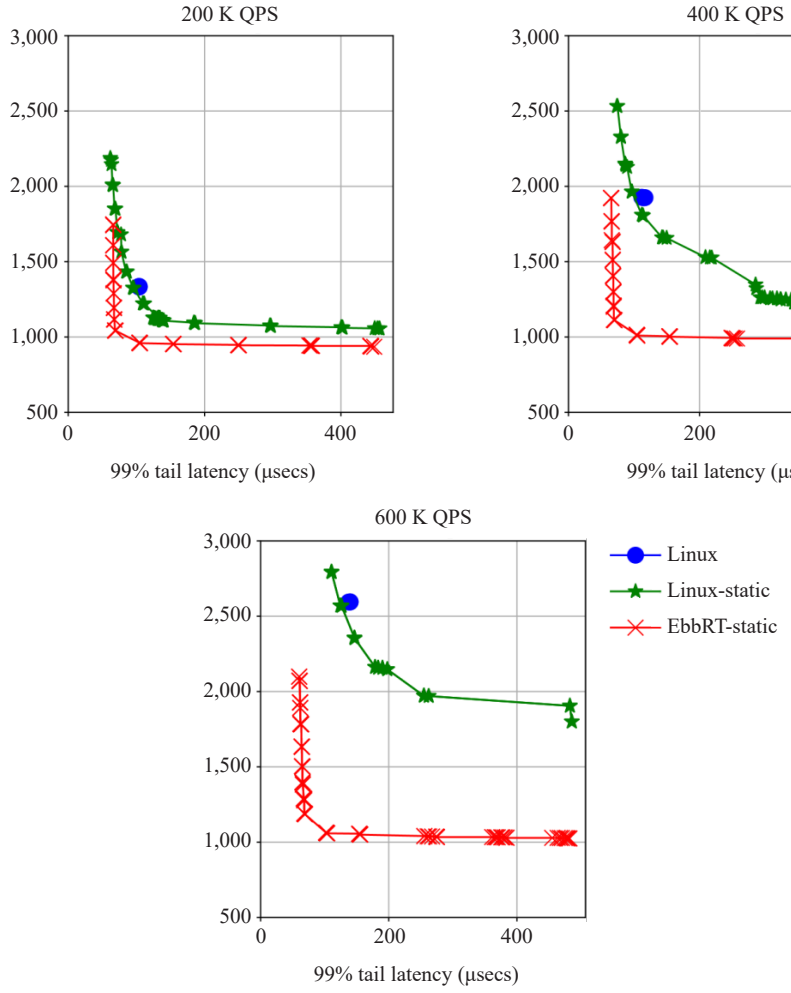


Figure 3. Memcached: Each point represents a single experimental run. The *-static data points use a unique (ITR-delay, DVFS) pair. We only illustrate data that lie on the Pareto-optimal curve. The X-axis shows performance measurement (lower is better) and the Y-axis shows total energy consumed

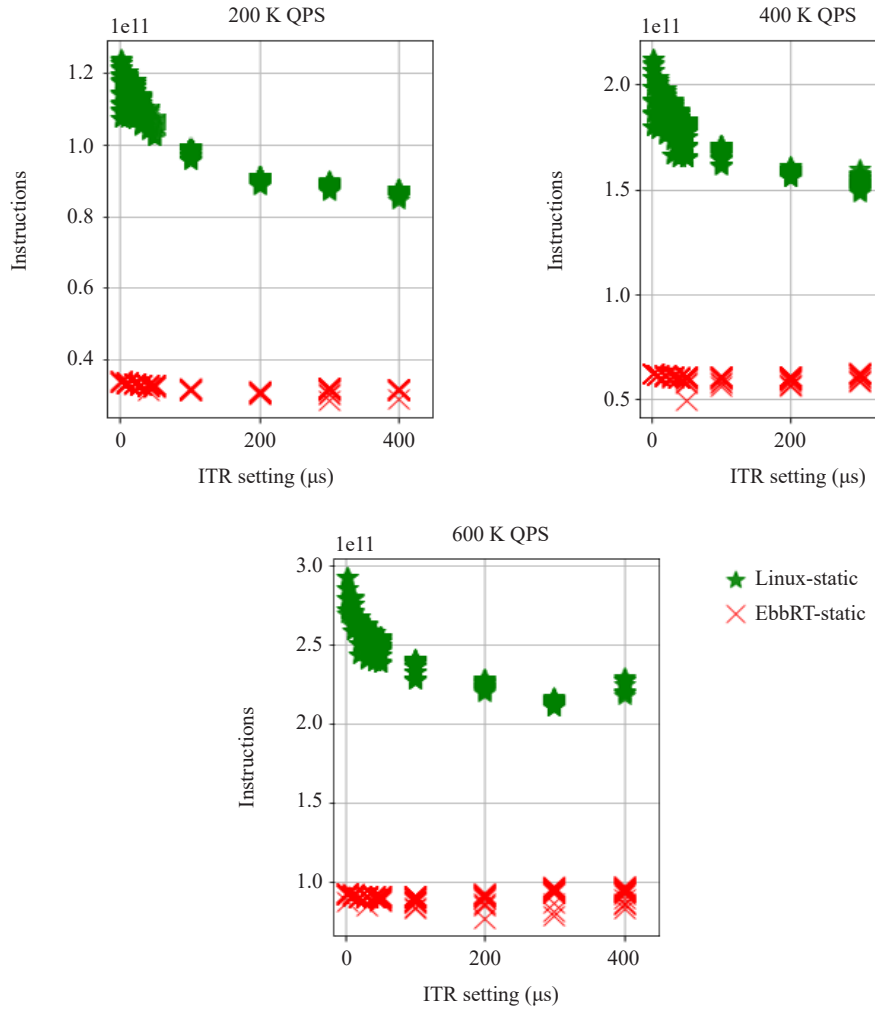


Figure 4. Memcached: ITR-delay impact on instruction counts (1e11)
Note: Not drawn to scale to show structure in data

3.5.3 *Revealing ITR-delay and DVFS performance-energy trade-offs*

We present an example in Figure 5 featuring a Linux Memcached server with a request rate of 400 K QPS. This example helps build intuition regarding the impact of specific ITR-delay and DVFS settings on the performance and energy of open-loop style applications. Using colored gradients, the figure visually represents the effects of each ITR-delay and DVFS pair. Each data point is divided in half to provide specific insights into their respective impacts on 99% latency and energy. In Figure 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. First, a fast ITR-delay (0 µs) triggers a spike in tail latency that violates the 500 µs SLA objective. This occurs because the slow CPU frequency cannot process incoming requests quickly enough. Second, as the ITR-delay increases, the resulting queuing enables efficient request batching, which facilitates 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 trend is evident in the common “L” shapes seen in Figure 3. While these shapes differ in absolute performance and energy, they illustrate how ITR-delay and DVFS can be combined to reduce energy while still meeting SLA objectives in both OSes.

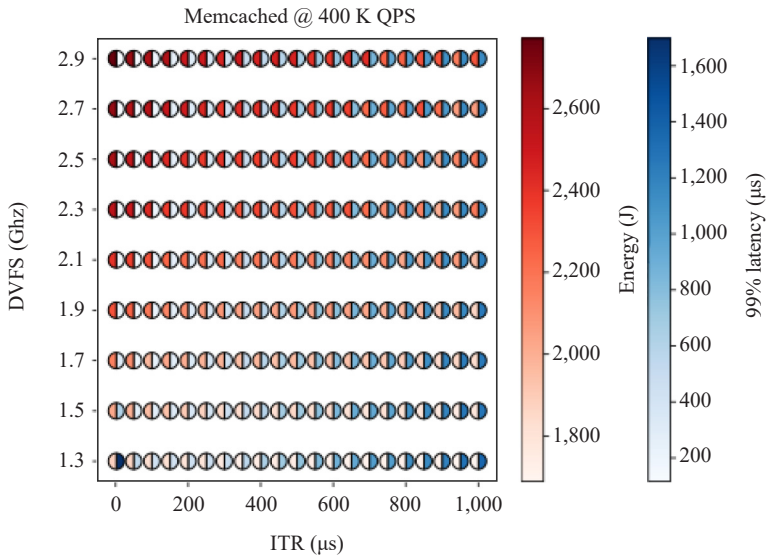


Figure 5. This figure shows the change in energy and 99% latency as different ITR-delay, DVFS pairs are explored for Linux Memcached

4. Modeling ITR-delay and DVFS effects

A key takeaway from §3 is that Figure 1 and Figure 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. Such a model would make generic external control mechanisms feasible for both OSes.

4.1 *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 request rate 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. We have also collected other tail latency values and found that our model can accurately fit them as well.

4.1.1 *Performance*

We define Δt as the time it takes to handle a single request:

$$\Delta t = t_{\text{work}} + t_{\text{interrupt}}$$

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

$$t_{\text{work}} = \frac{Z}{\text{DVFS}^{1+\alpha}} \tag{1}$$

Z and α are free parameters that change for both the OS and application. In this model, Z acts as a maximum time limit that each request can take, such as an 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 §3.5.2 that illustrate how DVFS affects Linux and EbbRT differently in trading off energy for latency.

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

$$t_{\text{interrupt}} = \phi \times \text{ITR-delay} \tag{2}$$

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-delay value starts counting down, then it will have to artificially wait for a full ITR-delay before being processed, thereby delaying overall request processing time.

4.1.2 *Energy*

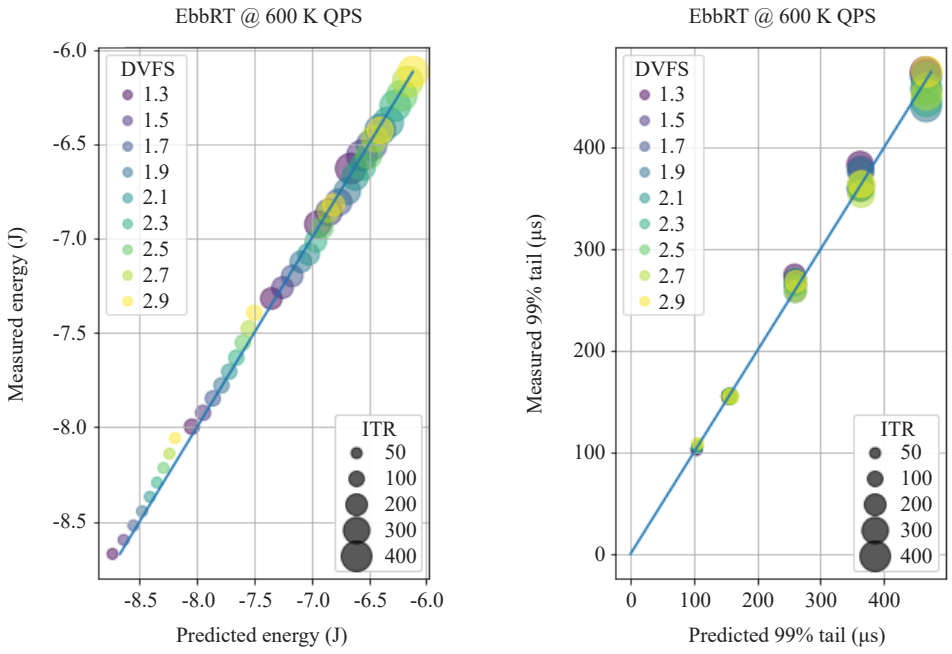
We define ΔJ as the amount of energy (or joules) it takes to process a single request. This is affected by the voltage and frequency states of DVFS and how ITR-delay can induce prolonged idle periods:

$$\Delta J = \gamma \times (\phi \times \text{ITR}) \times \text{DVFS}^{\beta} \tag{3}$$

Note that ϕ used here is the same variable from equation (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 equation (1).

4.2 *Modeling results*

Figure 6 illustrates one example result of the model fitting against Memcached data for a QPS of 600 K. 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 predictions were exact. We use the Adam optimizer from PyTorch in this process and run each fit several times to check the stability of the inferred parameters and avoid getting stuck in local minima. The negative energy values shown on (Y-axis) in Figure 6 are due to the use of log transformations in our regression analysis. 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.



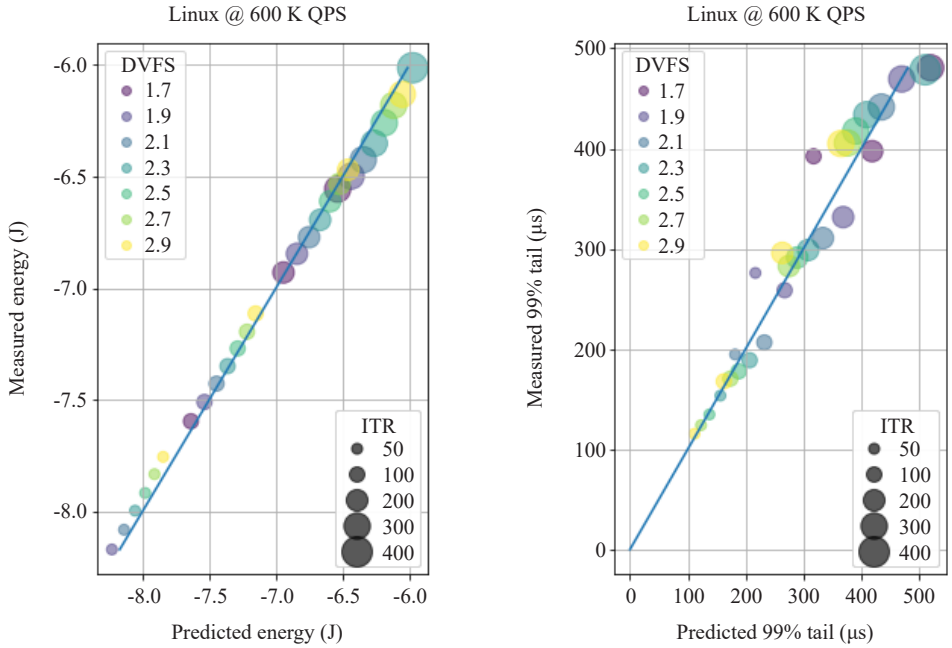


Figure 6. Prediction of energy and performance across both OSes using our model for Memcached @ 600 K QPS. The diagonal line indicates a perfect model fit

4.3 *Modeling takeaways*

In highly constrained settings, our models are able to characterize the relationship between ITR-delay, DVFS, and energy efficiency. While the accuracy of these models suggests a clear path for optimization, the static modeling approach used in work is ultimately impractical for general deployment as it would require exhaustive data re-gathering for every new application, OS, and hardware.

Therefore, the primary role of this model is to inform the ML-based controller design (in §5) by helping to validate that the system response surface is structured rather than stochastic. Our modeling results demonstrate that while the optimal configuration is distinct, their specific settings are sensitive to the underlying software stack and hardware platform. This insight directly dictates our controller design such that it can exploit the stability of request demand curves with a generic, black-box approach. Consequently, our results below show that the controller can autonomously find optimal settings across diverse environments, providing the benefits of a high-fidelity model without the overhead of platform-specific training.

5. A 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 [19, 54]. We use this method to find energy-efficient ITR-delay and DVFS settings that adapt to the specific application, OS, and hardware. This approach successfully exploits the inherent stability in request rates. In §5.2, we illustrate the controller's applicability in optimizing a server supporting a realistic, 24-hour datacenter workload trace [20]. The controller achieves energy efficiency by periodically adjusting ITR-delay and DVFS settings as the request rate changes. Furthermore, §5.3 demonstrates the generality of the controller. We apply it across different types of NICs and CPUs (Table 2) while running three applications from Tailbench [21].

Table 2. Different hardware explored to run Tailbench

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

5.1 *Design*

Figure 7 illustrates the design of our controller. First, a live system running Memcached services requests arriving at varying QPS levels from an external source. Second, the system triggers a set of performance and energy measurements to be shared with an external Ax [55, 56] Bayesian optimization platform. Third, this process computes a penalty, *Rp*, for the current (ITR-delay, DVFS) setting. Fourth, the platform recommends an update to a new configuration on the live system that minimizes *Rp*. Once this process completes, the live system maintains a fixed configuration until the next set of measurements is triggered.

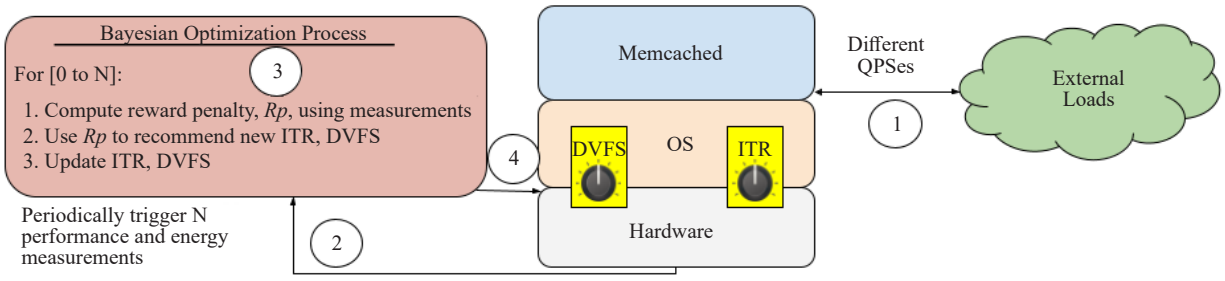


Figure 7. Our controller designed to optimize energy efficiency for a Memcached server

5.1.1 *Penalty function*

Our custom penalty function (*Rp*) is designed to prioritize SLA adherence while minimizing energy use. Specifically, it is constructed to create a performance cliff. In this scenario, as long as the measured latency (*m_latency*) remains below the SLA, the penalty is driven solely by *m_energy*. However, as soon as the SLA is violated, the energy penalty increases at an accelerated rate due to *m_energy* × max((*m_latency* − SLA + 1), 1). This ensures that Bayesian optimization treats the SLA as a critical boundary. This approach is more robust than other metrics, such as Energy-Delay Product (EDP), because it prevents the controller from trading off unacceptable latency (violating SLA) for marginal energy gains.

$$Rp = m\_energy \times \max\left(\left(m\_latency - \mathrm{SLA} + 1\right),\ 1\right) \tag{4}$$

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 + 1). 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.

5.2 *Applicability to QPS rates from a real-world trace*

This section presents the results of running our controller against a publicly available KV store workload trace (cachetrace [20]) which exhibits stable demand curves.

5.2.1 *Experimental setup*

We used the same infrastructure as our study (§3) but modified the mutilate workload generator to follow the cachetrace. First, we extracted a 24-hour sequence of QPS rates from a single trace. We then binned the data into hourly divisions to capture the mean QPS rate on an hourly basis. As cachetrace QPS rates were often in the tens of thousands of QPS as it was running on limited virtual CPUs (vCPUs), we scaled up the rates to match our hardware capability. Figure 8 illustrates both the raw QPS rates from a single cluster as well as the running mean QPS across the 5-day trace data. From this figure, one can see the repetitive change in QPSes across multiple days and within a single day. Therefore, we were motivated to explore a subset of these QPS rates within a single day. Differing from our experimental study where we explored only three QPS rates, we extracted a set of twenty-four mean QPSes from cachetrace. We use these to quantify the ability of Bayesian optimization to dynamically adapt to realistic QPS rates. Our goal is to demonstrate that the controller can save energy while still meeting SLA objectives.

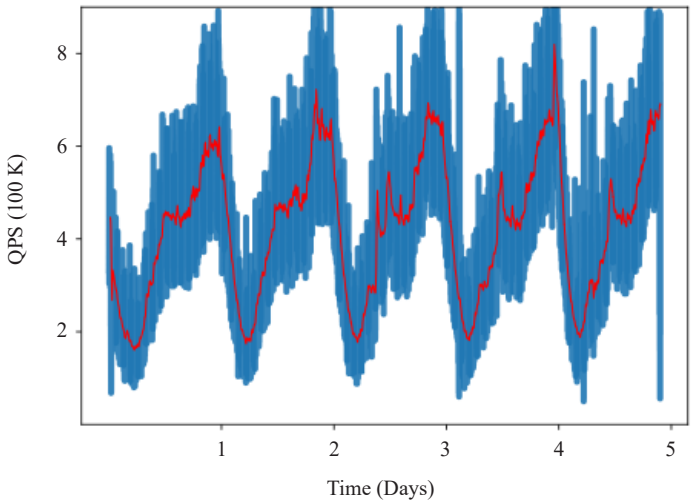


Figure 8. This illustrates the raw QPS rates in a cluster from cachetrace over 5 days. The red line tracks the running mean QPS as request rate change throughout the days

We configured Bayesian optimization to run for only thirty trials to explore different parameters. This stands in contrast to our energy study (§3), which was limited to 340 (ITR-delay, DVFS) pairs due to experimental scope. Our controller allows Bayesian optimization to choose from all available values, totaling 2 million possible combinations. After the thirty trials, we configured the server with the ITR-delay and DVFS settings that yielded the most energy savings. We then used mutilate to generate each QPS rate for one hour and measured the server's resulting power consumption. While we set mutilate to generate request rate at a specific mean QPS rate, the inter-request latency is actually more dynamic given that it is drawn from a distribution.

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

1. Linux: Operating in its default state, where the dynamic ITR-delay and DVFS algorithms are enabled.

2. 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.

3. Co-PI: A recently published system [34] that co-adjusts ITR-delay and DVFS using an offline profiler. We

follow the example in Co-PI by profiling a coarse-grained granularity of different request rates (taken from our experimental study) and utilizing the profiled pareto-optimal settings for ITR-delay and DVFS.

5.2.2 *Evaluation*

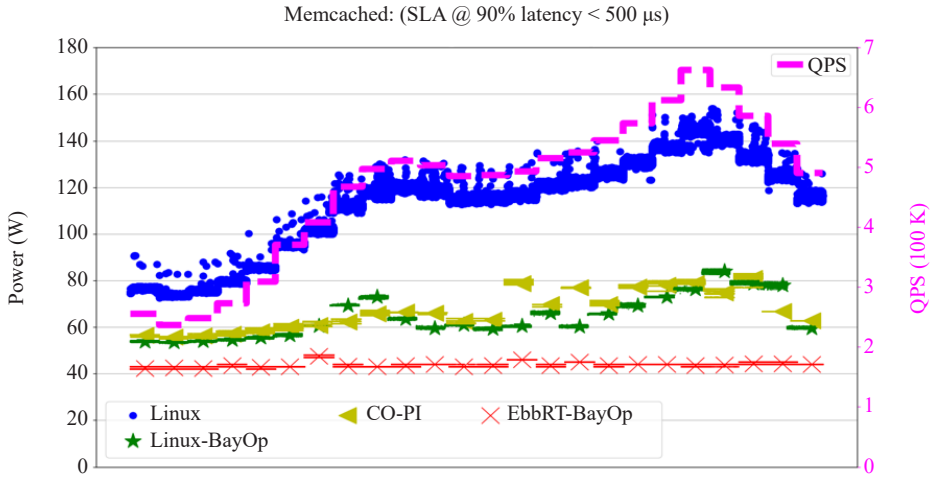
We evaluate Linux-BayOp and EbbRT-BayOp energy impact across two applications, namely Memcached and silo, in both Linux and EbbRT. Silo [23] is a compute and memory-intensive application that is extended with a web front-end such that every request triggers a corresponding set of Transaction Processing Performance Council Benchmark C (TPC-C) transactions on an in-memory database [57]. We ported Silo to EbbRT and the workload mix and SLA constraints of Silo follow from those used in Memcached.

5.2.2.1 *Memcached results*

Figure 9 illustrates our controller's evaluation against three different SLA objectives: 99% latency < 500 µs, 90% latency < 500 µs, and an even more stringent 99% latency < 200 µs. The QPS values are shown on the right of each figure and change on an hourly basis. A key result of this experiment is revealing the importance of tuning both ITR-delay and DVFS to meet SLA objectives for optimizing energy efficiency.

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 (Figure 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 than that of Linux. This is indicated by the thinner red plot in Figure 9. Such stability is a byproduct of EbbRT's simplified and more optimized network paths.

In Co-PI we find results in energy savings that are sometimes comparable with Linux-BayOp across SLA objectives, though typically it performed worse than Linux-BayOp. This is due to the coarse-grained ITR-delay, DVFS settings used from the profiler, whereas Linux-BayOp is able to optimize for the specific QPSes. Under a more stringent SLA of 99% latency < 200 µs, though we find that Co-PI results in the highest energy savings. However, we actually found that its coarse-grained offline profiling approach cannot account for the different QPSes and the resulting ITR-delay, DVFS settings typically ended up causing SLA violations as shown in Figure 10.



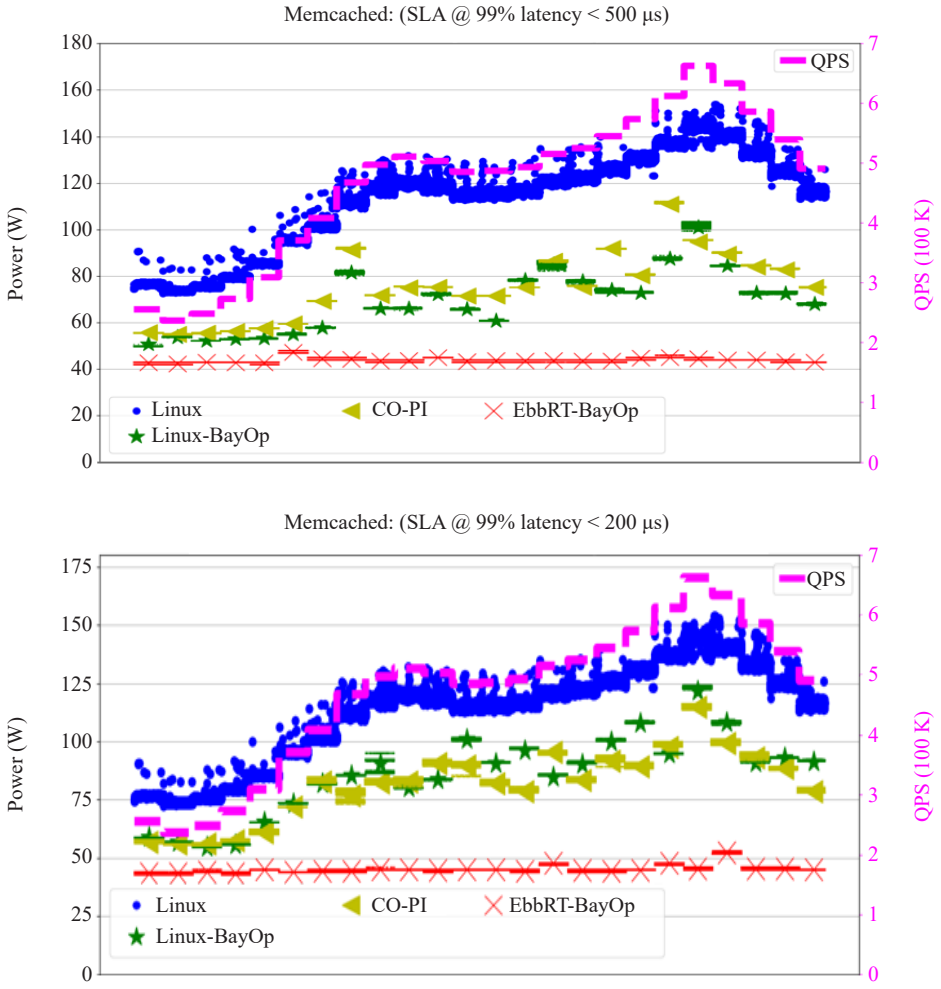


Figure 9. Bayesian optimization applied to Memcached; the QPS label is shown on the right side of the graph and the QPS lines show the different request rates on a per-hour basis. We present results from different SLA objectives studied and illustrate the measured power (energy/second) on the Y-axis as QPS changes across the four system configurations studied over 24 hours (X-axis)

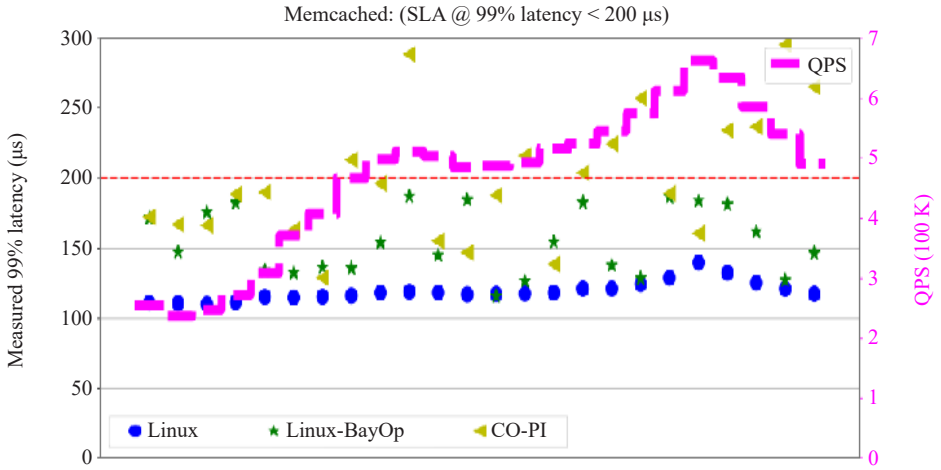


Figure 10. Measured 99% latency across Linux for an SLA of 200 µs. The latency is shown on a per-hour basis due to how mutilate reports its resultant latency measurements. We find that Co-PI often violates the changing SLA which suggests that only offline profiling is not enough to achieve stable system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

5.2.2.2 *Silo results*

We selected another trace from cachetrace that was akin to a more computationally intense server. Figure 11 shows that the trace peak QPS rates are often lower than those of Figure 9 (peak 250 K QPS versus 750 K QPS). Figure 11 does not show results for an SLA of 99% latency < 200 µs. The inherent computational cost of Silo's TPC-C transactions resulted in a lower bound of measured latency values that were consistently greater than this objective.

Figure 11 illustrates that Linux-BayOp found (ITR-delay, DVFS) settings that enable 30% energy savings in Linux for various QPS rates, even for a computationally intensive application. Furthermore, the controller achieved even higher savings when the SLA was 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 as well. Comparing to Co-PI, one also sees similar benefits from using Linux-BayOp. In contrast to Figure 9, 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 must now be done per request.

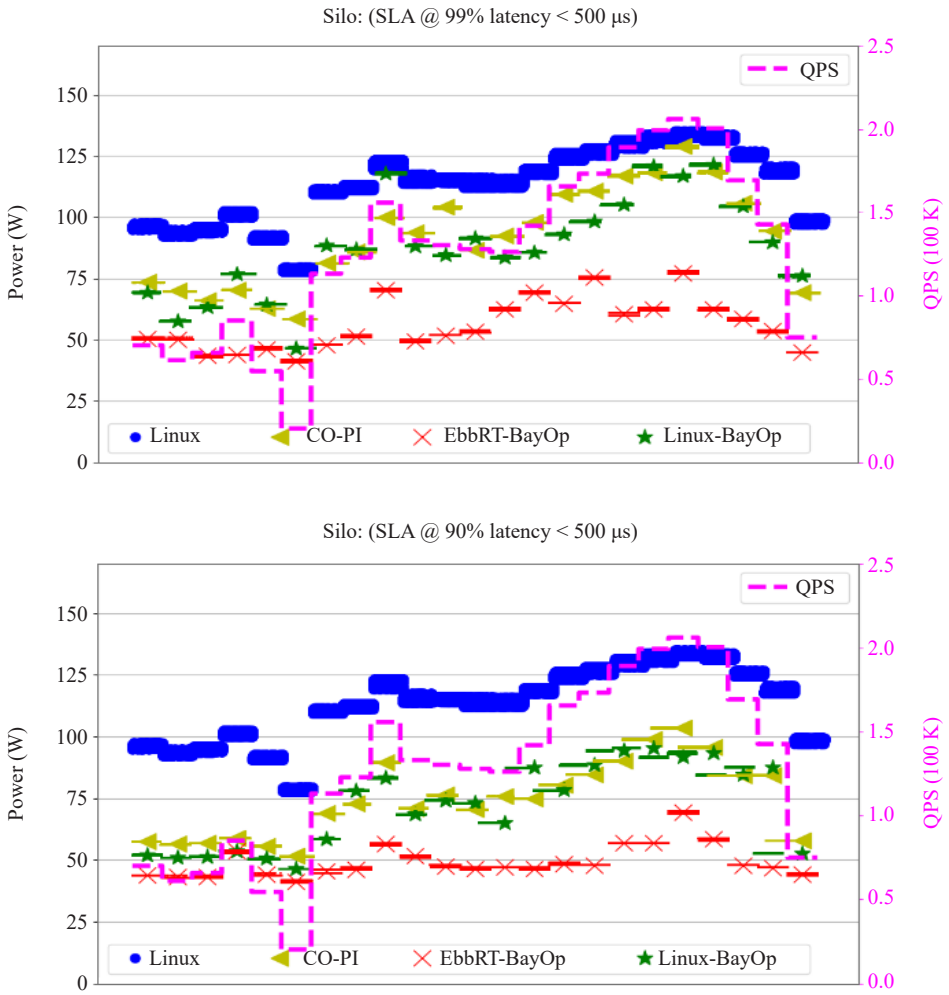


Figure 11. Controller applied to cachetrace for Silo. We show two different SLA objectives. The QPS line shows the change in QPS request rate on a per-hour basis. The consumed power (Joules/second) of each system configuration on the Y-axis is shown over 24 hours on the X-axis

5.3 *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 [21]. Our motivation was to reveal how externally controlling interrupt coalescing and CPU frequency can be applied agnostically on hardware even across generations.

5.3.1 *Experimental setup*

For these experiments, we selected four hardware platforms as shown in Table 2. Nodes N0, N1, and N2 are provided by CloudLab [58] 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 Bayesian optimization controller to tune Interrupt delay (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. In addition, 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 [59] to report its dynamic energy use while for AMD, we use amd_energy hardware monitor driver.

Node N3 is another experimental node that runs Linux 6.4.13 but we could only get a single client node to generate traffic. However, due to the computation-heavy nature of Tailbench applications, we found this was still able to saturate the single server. The ARMv8 server provided xgene-hwmon tool that enabled us to report its power readings. For each hardware category, we selected applications from Tailbench [22], each designed to fulfill distinct SLA objectives. These applications encompassed img-dnn, a handwriting recognition program built on OpenCV, and sphinx, an open-source speech recognition system. We also included xapian, which is an open-source search engine. 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.

5.3.2 *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. These 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 allowed for straightforward deployment in new hardware environments. This is possible as long as the environment provides support for energy readings and exposes control of interrupt coalescing and CPU frequency.

Figure 12 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 request rates of 40% and 80% of each hardware platform's peak QPS capacity for running the respective applications.

2. For each of these request rates, 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 [21].

It is worth pointing out that the controller's ability to adapt to applications and request rate is 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, which is an abstracted performance value that isn't tied to specific a CPU frequency. As this is a form of hardware throttling, 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 Energy Performance Preference (EPP) driver, providing finer-grained CPU frequency settings. Further, one can also address this through application-level request pacing, which can act as a form 'virtual' frequency settings.

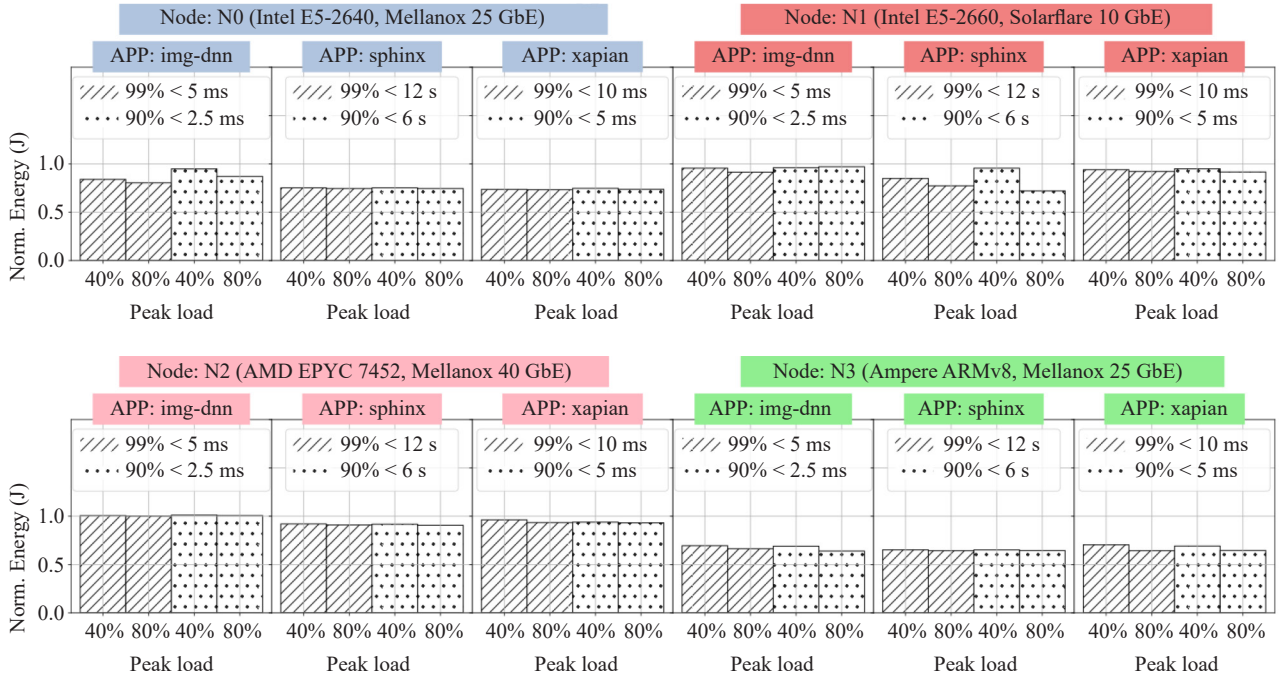


Figure 12. This figure illustrates the energy use of each Application (APP) for each of the hardware platforms (NODE: N0 to N3). The energy is normalized (Y-axis) against Linux default, where lower is better. For each APP, we use two representative request rates which are 40% and 80% of the measured Peak Load of Linux default. Within each representative request rate, we also selected two SLAs (as indicated by //// and ····) for the application to meet while our controller is optimizing its energy efficiency

To delve into reasons for the energy savings, we show the Cumulative Distribution Function (CDF) of an example Tailbench application in Figure 13. In this figure, we illustrate the per-request latency as provided by Tailbench when running the img-dnn application on our ARMv8 server (this is at a particular peak request rate 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 2.8 ms, Linux-BayOp was still able to meet the SLA at 99% latency of 4.8 ms while saving 31% energy.

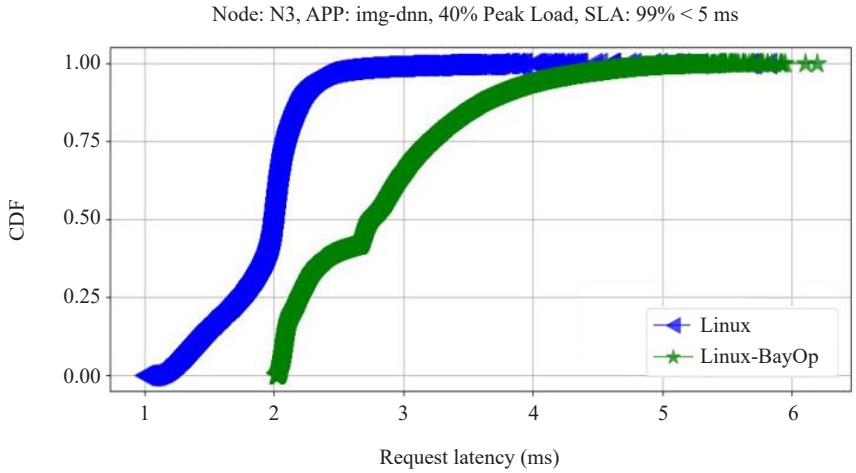


Figure 13. CDF of per request latency between Linux and Linux-BayOp from a single Tailbench application

6. Discussion

Below, we discuss the broader implications of our findings and outline future avenues of research. The core contribution of this work lies in identifying and quantifying the potential of ITR-delay and DVFS as fundamental mechanisms for energy efficiency. Our study reveals that these hardware parameters create a structured response surface that allows a black-box optimization approach to succeed across diverse applications, OSes, and hardware.

In §5.2, we made certain simplifying assumptions, including the use of an hourly trigger to run Bayesian optimization and a default of 30 trials to sample measurements. However, our results illustrate that these straightforward assumptions can yield significant energy savings of up to 60%, leaving ample room for improvement. These variables can also be adjusted depending on changing request rate behavior of the application. Furthermore, our trial number of 30 was set in an ad hoc way based on the overheads in our Bayesian optimization library and we believe there is room to further optimize these overheads or use more advanced methods that further reduce the cost of sampling [60]. Further, while our Bayesian optimization approach runs on a single thread, we find it is still capable of finding energy efficient settings that outperform existing systems. We believe this approach can be further optimized through the use of accelerators.

The architecture of our controller also facilitates the integration of more advanced policies for initiating the Bayesian optimization. For example, it can respond to changes in QPS rates or leverage historical patterns in service request rates. We envision the deployment of this technique in datacenters through collaboration with load-balancers that make use of historical usage data. This collaboration would help distribute incoming requests to energy-optimized servers, which have been pre-configured with specific ITR and DVFS settings, while still meeting SLA objectives and minimizing energy consumption. In addition, the load-balancer can help mitigate potential gaming of the learning agent's behavior in response to changing request rates.

While we used a simplistic reward function in equation (4), the possibilities of customizing this function further are ripe for exploration. Future work could investigate new combinations of performance and energy or modifying and mixing with established metrics such as the energy-delay product [61, 62]. One can also imagine developing a set of different 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.

While our implementation utilizes Bayesian optimization to navigate this space, our work serves as a proof-of-concept for the viability of such control levers. Further, it does not preclude the use of alternative ML methods [57] or the inclusion of additional tunable system parameters. While we leave the complexities of large-scale, production-grade deployment for future research, this paper establishes the foundation and initial framework necessary to pursue such real-world implementations.

7. 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 request rates, applications, OSes and even hardware.

Acknowledgements

This work is supported through the Red Hat Collaboratory, Optimizing Kernel Paths for Performance and Energy Hardware, 2024-01-RH09.

Conflict of interest

The authors declare no competing financial interest.

References

[1] Tang C, Yu K, Veeraraghavan K, Kaldor J, Michelson S, Kooburat T, et al. Twine: A unified cluster management system for shared infrastructure. In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. Canada: USENIX Association; 2020. p.787-803. Available from: https://doi. org/10.5555/3488766.3488811.

[2] Zhang Y, Cai J, Wildani A, Klimovic A. Rethinking web cache design for the AI era. In: *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC 25)*. New York, USA: Association for Computing Machinery; 2026. p.535-542. Available from: https://doi.org/10.1145/3772052.3772255.

[3] Fried J, Chaudhry GI, Saurez E, Choukse E, Goiri Í, Elnikety S, et al. Making kernel bypass practical for the cloud with junction. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA, USA: USENIX Association; 2024. p.55-73. Available from: https://doi. org/10.5555/3691825.3691829.

[4] Szekely A, Belay A, Morris R, Kaashoek MF. Unifying serverless and microservice workloads with SigmaOS. In: *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. New York, NY, USA: Association for Computing Machinery; 2024. p.385-402. Available from: https://doi.org/10.1145/3694715.3695947.

[5] Raza A, Unger T, Boyd M, Munson EB, Sohal P, Drepper U, et al. Unikernel linux (UKL). In: *Proceedings of the Eighteenth European Conference on Computer Systems*. Rome, Italy: Association for Computing Machinery; 2023. p.590-605. Available from: https://doi. org/10.1145/3552326.3587458.

[6] Gupta U, Kim YG, Lee S, Tse J, Lee HHS, Wei GY, et al. Chasing carbon: The elusive environmental footprint of computing. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Seoul, Korea (South): IEEE; 2021. p.854-867. Available from: https://doi. org/10.1109/HPCA51647.2021.00076.

[7] Strubell E, Ganesh A, McCallum A. Energy and policy considerations for deep learning in NLP. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics; 2019. p.3645-3650. Available from: https://doi. org/10.18653/v1/P19-1355.

[8] Li B, Jiang Y, Tiwari D. Carbon in motion: Characterizing open-sora on the sustainability of generative AI for video generation. *ACM SIGENERGY Energy Informatics Review*. 2024; 4(5): 160-165. Available from: https://doi. org/10.1145/3727200.3727224.

[9] Maji D, Hanafy WA, Wu L, Irwin D, Shenoy P, Sitaraman RK. Data centers carbon emissions at crossroads: An empirical study. *ACM SIGENERGY Energy Informatics Review*. 2025; 5(2): 48-55. Available from: https://doi. org/10.1145/3757892.3757899.

[10] Lyerly R, Pruett S, Doherty K, Rogers G, Bronson N, Hugg J. Skybridge: Bounded staleness for distributed caches. In: *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. Boston, MA, USA: USENIX Association; 2025. p.187-204. Available from: https://doi. org/10.5555/3767901.3767912.

[11] Elnozahy M, Kistler M, Rajamony R. Energy conservation policies for web servers. In: *4th USENIX Symposium on Internet Technologies and Systems (USITS 03)*. Seattle, WA, USA: USENIX Association; 2003. Available from: https://doi. org/10.5555/1251460.1251468.

[12] Le Sueur E, Heiser G. Slow down or sleep, that is the question. In: *2011 USENIX Annual Technical Conference (USENIX ATC 11)*. Portland, OR, USA: USENIX Association; 2011. p.1-16. Available from: https://doi. org/10.5555/2002181.2002197.

[13] Kim DH, Imes C, Hoffmann H. 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*. Hong Kong, China: IEEE; 2015. p.78-85. Available from: https://doi. org/10.1109/CPSNA.2015.23.

[14] Park G, Kim M, Kang KD, Jeon Y, Kim S, Kim D. EcoCore: Dynamic core management for improving energy efficiency in latency-critical applications. In: *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. Seoul, South Korea: Association for Computing Machinery; 2025. p.1132-1146. Available from: https://doi. org/10.1145/3725843.3756021.

[15] Zhang J, Yu G, He Z, Ai L, Chen P. DeepPower: Deep reinforcement learning based power management for latency critical applications in multi-core systems. In: *Proceedings of the 52nd International Conference on Parallel Processing*. Salt Lake City, UT, USA: Association for Computing Machinery; 2023. p.327-336. Available from: https://doi. org/10.1145/3605573.3605612.

[16] Schatzberg D, Cadden J, Dong H, Krieger O, Appavoo J. EbbRT: A framework for building per-application library operating systems. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA, USA: USENIX Association; 2016. p.671-688. Available from: https://doi.org/10.5555/3026877.3026929.
[17] Salah K. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*. 2007; 61(4): 215-225. Available from: https://doi.org/10.1016/j.aeue.2006.04.007.
[18] Hou S, Ni W, Zhao K, Cheng B, Zhao S, Wan Z, et al. Fine-grained online energy management of edge data centers using per-core power gating and dynamic voltage and frequency scaling. *IEEE Transactions on Sustainable Computing*. 2023; 8(3): 522-536. Available from: https://doi.org/10.1109/TSUSC.2023.3250487.
[19] Garnett R. *Bayesian Optimization*. Cambridge, UK: Cambridge University Press; 2023.
[20] Yang J, Yue Y, Rashmi KV. A large-scale analysis of hundreds of in-memory key-value cache clusters at twitter. *ACM Transactions on Storage (TOS)*. 2021; 17(3): 1-35. Available from: https://doi.org/10.1145/3468521.
[21] Kasture H, Sanchez D. Tailbench: A benchmark suite and evaluation methodology for latency-critical applications. In: *2016 IEEE International Symposium on Workload Characterization (IISWC)*. Providence, RI, USA: IEEE; 2016. p.1-10. Available from: https://doi. org/10.1109/IISWC.2016.7581261.
[22] Barroso LA, Hölzle U, Ranganathan P. Energy and sustainability. In: Hill MD, Martonosi M. (eds.) *The Data Center as a Computer: Designing Warehouse-Scale Machines*. Switzerland: Springer Nature; 2025. p.213-267. Available from: https://doi.org/10.1007/978-3-031-99489-0_9.
[23] Prekas G, Kogias M, Bugnion E. ZygOS: Achieving low tail latency for microsecond-scale networked tasks. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. Shanghai, China: Association for Computing Machinery; 2017. p.325-341. Available from: https://doi.org/10.1145/3132747.3132780.
[24] Hu Y, Zhang Z, Liu Y, Gu Y, Lei S, Kasikci B, et al. Mitigating application resource overload with targeted task cancellation. In: *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*. Seoul, South Korea: Association for Computing Machinery; 2025. p.270-285. Available from: https://doi.org/10.1145/3731569.3764835.
[25] Olivier P, Mehrab AF, Errabelly S, Lankes S, Karaoui ML, Lyerly R, et al. HEXO: Offloading long-running compute-and memory-intensive workloads on low-cost, low-power embedded systems. *IEEE Transactions on Cloud Computing*. 2024; 12(4): 1415-1432. Available from: https://doi.org/10.1109/TCC.2024.3482178.
[26] Chou CH, Bhuyan LN, Wong D. μDPM: Dynamic power management for the microsecond era. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Washington, DC, USA: IEEE; 2019. p.120-132. Available from: https://doi.org/10.1109/HPCA.2019.00032.
[27] Yahya JH, Volos H, Bartolini DB, Antoniou G, Kim JS, Wang Z, et al. Agilewatts: An energy-efficient CPU core idle-state architecture for latency-sensitive server applications. In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Chicago, IL, USA: IEEE; 2022. p.835-850. Available from: https://doi.org/10.1109/MICRO56248.2022.00063.
[28] Noguchi H, Kaneko M. Application-independent DVFS method based on OS metrics monitoring for cloud servers. In: *2024 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*. Beijing, China: IEEE; 2024. p.1-6. Available from: https://doi.org/10.1109/CCCI61916.2024.10736461.
[29] Sun Y, Ding Z, Dehghanian P, Teng F. Learning-enabled adaptive power capping scheme for cloud data centers. *IEEE Transactions on Smart Grid*. 2025; 16(6): 4755-4767. Available from: https://doi.org/10.1109/TSG.2025.3598070.
[30] Lo D, Cheng L, Govindaraju R, Ranganathan P, Kozyrakis C. Heracles: Improving resource efficiency at scale. In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. Portland, Oregon, USA: Association for Computing Machinery; 2015. p.450-462. Available from: https://doi.org/10.1145/2749469.2749475.
[31] Guliani A, Swift MM. Per-application power delivery. In: *Proceedings of the Fourteenth EuroSys Conference 2019*. Dresden, Germany; Association for Computing Machinery; 2019. p.1-16. Available from: https://doi.org/10.1145/3302424.3303981.
[32] Mishra N, Imes C, Lafferty JD, Hoffmann H. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*. 2018; 53(2): 184-198. Available from: https://doi.org/10.1145/3296957.3173184.
[33] Shirvani S, Samanta A, Li Z, Liu C. DuoJoule: Accurate on-device deep reinforcement learning for energy and timeliness. In: *2024 IEEE Real-Time Systems Symposium (RTSS)*. York, United Kingdom: IEEE; 2024. p.109-122. Available from: https://doi.org/10.1109/RTSS62706.2024.00019.
[34] Kang KD, Park H, Park G, Kim D. Co-adjusting voltage/frequency state and interrupt rate for improving energy-efficiency of latency-critical applications. *IEEE Access*. 2020; 8: 201028-201039. Available from: https://doi.org/10.1109/ACCESS.2020.3035777.

[35] Wu N, Xie Y. A survey of machine learning for computer architecture and systems. *ACM Computing Surveys (CSUR)*. 2022; 55(3): 1-39. Available from: https://doi.org/10.1145/3494523.
[36] Ding Y, Mishra N, Hoffmann H. Generative and multi-phase learning for computer systems optimization. In: *Proceedings of the 46th International Symposium on Computer Architecture*. Phoenix, Arizona, USA: Association for Computing Machinery; 2019. p.39-52. Available from: https://doi.org/10.1145/3307650.3326633.
[37] Ding Y, Pervaiz A, Carbin M, Hoffmann H. 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*. Athens, Greece: Association for Computing Machinery; 2021. p.728-740. Available from: https://doi.org/10.1145/3468264.3468603.
[38] Truyen E, Joosen W. Optimal configuration of API resources in cloud native computing. *Electronic Proceedings in Theoretical Computer Science*. 2025; 438: 15-39. Available from: https://doi.org/10.4204/EPTCS.438.2.
[39] Roy RB, Patel T, Gadepally V, Tiwari D. 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*. Virtual, Canada: Association for Computing Machinery; 2021. p.1280-1295. Available from: https://doi.org/10.1145/3453483.3454109.
[40] Iqbal MS, Zhong Z, Ahmad I, Ray B, Jamshidi P. CAMEO: A causal transfer learning approach for performance optimization of configurable computer systems. In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. Santa Cruz, CA, USA: Association for Computing Machinery; 2023. p.555-571. Available from: https://doi.org/10.1145/3620678.3624791.
[41] Ding Y, Rao A, Song H, Willett R, Hoffmann HH. NURD: Negative-unlabeled learning for online datacenter straggler prediction. *arXiv:2203.08339*. 2022. Available from: https://doi.org/10.48550/arXiv.2203.08339.
[42] Shaffer MW. A linux appliance construction set. In: *14th Systems Administration Conference (LISA 2000)*. New Orleans, Louisiana, USA: USENIX Association; 2000.
[43] Dong H, Appavoo J. *A tutorial on building custom linux appliances*. Available from: https://www.usenix.org/publications/loginonline/building-linux-appliances [Accessed 21st January 2026].
[44] Ren X, Rodrigues K, Chen L, Vega C, Stumm M, Yuan D. An analysis of performance evolution of Linux's core operations. In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP '19*. Huntsville, Ontario, Canada: Association for Computing Machinery; 2019. p.554-569. Available from: https://doi.org/10.1145/3341301.3359640.
[45] Madhavapeddy A, Mortier R, Rotsos C, Scott D, Singh B, Gazagnaire T, et al. Unikernels: Library operating systems for the cloud. *ACM SIGARCH Computer Architecture News*. 2013; 41(1): 461-472. Available from: https://doi.org/10.1145/2451116.2451167.
[46] Dong H, Arora S, Krieger O, Appavoo J. Can OS specialization give new life to old carbon in the cloud? In: *Proceedings of the 17th ACM International Systems and Storage Conference*. Virtual, Israel: Association for Computing Machinery; 2024. p.83-90. Available from: https://doi.org/10.1145/3688351.3689158.
[47] Snell QO, Mikler AR, Gustafson JL. Netpipe: A network protocol independent performance evaluator. In: *IASTED International Conference on Intelligent Information Management and Systems*. Honolulu, Hawaii, USA: ACTA Press; 1996. p.49-54.
[48] Fitzpatrick B. Distributed caching with memcached. *Linux Journal*. 2004; 2004(124): 5. Available from: https://doi.org/10.5555/1012889.1012894.
[49] Barroso LA, Hölzle U, Ranganathan P. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 1st ed. San Rafael, CA, USA: Morgan and Claypool Publishers; 2009.
[50] Leverich J. *Mutilate: High-performance memcached load generator*. 2014. Available from: https://github.com/leverich/mutilate [Accessed 21st January 2026].
[51] Wang X, Jin P, Yuan Y. SEMU: Concurrency-optimized high-performance cache management for key-value caches. *IEEE Transactions on Computers*. 2025; 75(2): 734-747. Available from: https://doi.org/10.1109/TC.2025.3643461.
[52] Natori K, Fujimoto K, Shiraga A. Sleep control of packet receiving thread toward power saving. *Annals of Telecommunications*. 2025. Available from: https://doi.org/10.1007/s12243-025-01084-2.
[53] Asyabi E, Bestavros A, Sharafzadeh E, Zhu T. 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)*. New York, NY, USA: Association for Computing Machinery; 2020. p.150-164. Available from: https://doi.org/10.1145/3419111.3421298.
[54] Frazier PI. A tutorial on Bayesian optimization. *arXiv: 1807.02811*. 2018. Available from: https://doi.org/10.48550/

arXiv.1807.02811.
[55] Olson M, Santorella E, Tiao LC, Cakmak S, Garrard M, Daulton S, et al. Ax: A platform for adaptive experimentation. In: *AutoML 2025 ABCD Track*. New York, NY, USA; 2025.
[56] Bakshy E, Dworkin L, Karrer B, Kashin K, Letham B, Murthy A, et al. AE: A domain-agnostic platform for adaptive experimentation. In: *Conference on Neural Information Processing Systems*. Montréal, Canada: NeurIPS Foundation; 2018. p.1-8.
[57] Tu S, Zheng W, Kohler E, Liskov B, Madden S. Speedy transactions in multicore in-memory databases. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. Farminton, Pennsylvania, USA: Association for Computing Machinery; 2013. p.18-32. Available from: https://doi.org/10.1145/2517349.2522713.
[58] Duplyakin D, Ricci R, Maricq A, Wong G, Duerig J, Eide E, et al. The design and operation of CloudLab. In: *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. Seattle, WA, USA: USENIX Association; 2019. p.1-14.
[59] David H, Gorbatov E, Hanebutte UR, Khanna R, Le C. RAPL: Memory power estimation and capping. In: *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*. Austin, Texas, USA: Association for Computing Machinery; 2010. p.189-194. Available from: https://doi.org/10.1145/1840845.1840883.
[60] Ding Y, Renda A, Pervaiz A, Carbin M, Hoffmann H. Cello: Efficient computer systems optimization with predictive early termination and censored regression. *arXiv:2204.04831*. 2022. Available from: https://doi.org/10.48550/arXiv.2204.04831.
[61] Horowitz M, Indermaur T, Gonzalez R. Low-power digital design. In: *Proceedings of 1994 IEEE Symposium on Low Power Electronics*. San Diego, CA, USA: IEEE; 1994. p.8-11. Available from: https://doi.org/10.1109/LPE.1994.573184.
[62] Brooks DM, Bose P, Schuster SE, Jacobson H, Kudva PN, Buyuktosunoglu A, et al. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*. 2000; 20(6): 26-44. Available from: https://doi.org/10.1109/40.888701.