Towards Performance and Energy Aware Kubernetes Scheduler

HAN DONG, Hamilton College, USA
PARUL SINGH, Red Hat Inc, Netherlands
YARA AWAD, Boston University, USA
FELIX GEORGE, IBM Research, India
KRISHNASURI NARAYANAM, IBM Research, India
SANJAY ARORA, Red Hat Inc, USA
JONATHAN APPAVOO, Boston University, USA

As cloud services become increasingly latency-sensitive and data center energy usage rises, there is an urgent need to address both operational and embodied carbon cost. However, data centers often overprovision resources, resulting in resource under-utilization. These inefficiencies not only waste energy but also accelerate hardware refresh cycles, exacerbating embodied emissions. In this work, we present PAX, a performance and energy aware Kubernetes scheduler that leverages machine learning techniques. Specifically, we present preliminary results from using Bayesian optimization to optimize microservices across a heterogeneous cluster. PAX improves application performance compared to modern schedulers and enables carbon-conscious scheduling by dynamically placing workloads on old and new servers based on performance sensitivity. The results illustrate an opportunity to reduce operational carbon while extending server lifetimes to mitigate embodied emissions. Our approach highlights the potential of ML-enhanced scheduling as a mechanism for improving both resource efficiency and sustainability in modern cloud infrastructures.

CCS Concepts: • **General and reference → Experimentation**; • **Networks → Cloud computing**; • **Hardware → Impact on the environment**.

1 INTRODUCTION

Latency-sensitive cloud services [18, 40] play a critical role in the interactivity of many user-facing applications. Supporting these services requires a growing fleet of servers to represent hundreds of clusters which encompass hundreds of thousands of cores [40].

As global data center energy use rises [16, 23, 31, 34, 36], it is critical to find ways to meet the requirements of applications while reducing their carbon emissions. This issue is further exacerbated as research has shown that the embodied carbon of servers can account for almost half of data center emissions [28]. This is even more important given the strain on data center power by generative AI. Sustainable computing is becoming paramount [13, 32], especially in light of projections that carbon emissions from data centers could account for as much as 8% of global emissions within a decade [22].

Our work focuses on addressing these challenges in the area of cluster resource scheduling and allocating compute resources to microservices. Previous work [10] has shown that resource under utilization is a widespread phenomenon in data centers, with the majority of servers operating at less than 25% CPU utilization. Addressing this resource under-utilization is fundamental to improving data center sustainability, whether by making better use of existing hardware or by prolonging the utility of older hardware. Improving the performance of applications means further reduction in newly purchased hardware. Being able to maintain application performance through judicious use of older hardware platforms helps encourage hardware reuse and reduces embodied carbon. Improving the performance and energy efficiency of applications can also allow more of the data center power budget to be directed towards power intensive applications such as generative AI.

In this work, we present study results from the PEAKS project [35] of an experimental system PAX (PEAKS + AX), a performance and energy aware Kubernetes scheduler. PAX leverages insights from modern schedulers such as Cilantro [6] and the Horiziontal Pod Autoscaler (HPA) [26] to dynamically scale up the number of Kubernetes pods as demand and workload changes. PAX improves upon these schedulers by treating both the scaling of pods and their placement on different server hardware as avenues for optimization. Prior work[14, 39] has illustrated the opportunity of mixing old and new hardware to maintain performance while reducing carbon, and our work focuses on exploring this in Kubernetes schedulers. In this paper we study the potential for using hardware heterogeneity to exploit trade-offs in performance and carbon. For example, PAX may preferentially place latency-critical pods on newer servers while allocating less demanding pods to older servers that would otherwise be underutilized. PAX leverages the Adaptive Experimentation Platform (AX-platform) [29], a machine learning system for black-box parameter tuning, and applies it to the problem of pod scheduling across heterogeneous cluster resources. This paper makes the following contributions:

(1) Our results demonstrate that PAX is able to yield better performance and is robust to hardware heterogeneity. Our exploration of PAX highlights new opportunities to improve upon performance of modern hardware platforms. The approach we explore yielded up to 5X lower P99 latency for our chosen workload when compared to Cilantro and HPA across hardware clusters.
(2) Our results indicate the potential performance and carbon benefits of mixing different hardware. For example, across all three schedulers studied, we find that deploying pods on a heterogeneous cluster which consists of old and new hardware can even outperform deploying on new hardware while reducing energy consumption. Specifically, the mixed hardware cluster achieved up to 2.3X lower P99 latency and up to 45% lower power use.

Authors' addresses: Han Dong, hdong@hamilton.edu, Hamilton College, USA; Parul Singh, parsingh@redhat.com, Red Hat Inc, Netherlands; Yara Awad, awadyn@bu.edu, Boston University, USA; Felix George, Felix.George@ibm.com, IBM Research, India; Krishnasuri Narayanam, knaraya3@in.ibm.com, IBM Research, India; Sanjay Arora, saarora@redhat.com, Red Hat Inc, USA; Jonathan Appavoo, jappavoo@bu.edu, Boston University, USA.

(3) In contrast to Cilantro and HPA, our approach applies black-box parameter tuning to both scale the number of pods as well as the **placement of pods on specific server hardware**. Our results indicate that combined approach yields interesting trade-offs in performance, operational, and embodied carbon. They also underscore the need for more work to help quantify the conditions under which these trade-offs are achievable.

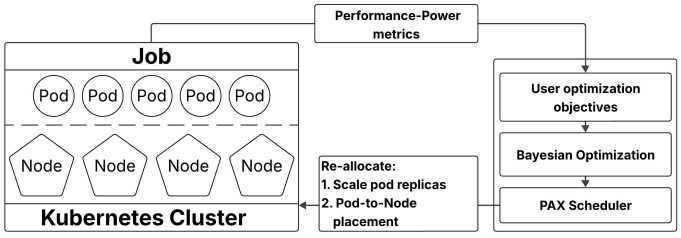


Fig. 1. High-level system design of PAX.

2 PAX OVERVIEW

Kubernetes is an open-sourced orchestration platform [8] for automating the deployment, scaling and management of containers. A node is a worker machine that is part of the Kubernetes cluster. A pod can be a group of one or more containers. Each node can have multiple pods deployed on it, and this pod-to-node placement is handled by the Kubernetes scheduler. Modern jobs deployed in a Kubernetes cluster typically consist of a diverse suite of microservices [18] spanning multiple pods and nodes.

In this work, PAX is used to optimize the 99th percentile tail latency in the hotel reservation benchmark from DeathStarBench [18]. This benchmark consists of 19 microservices, including 6 MongoDB databases, 3 key-value stores, an nginx web server using consul, and various other book-keeping services. This benchmark is representative of a modern job that can be deployed in Kubernetes, as each of the microservices is deployed in an individual pod. In such an application, there is typically a complicated inter-dependence between microservices, since handling a request can trigger multiple downstream queries to multiple backends running in different pods and nodes. The hotel reservation requests are generated by the wrk2 [20] workload generator and we use the default mixed query load provided by DeathStarBench [18].

The overarching goal of PAX is to scale the number of pods within a fixed budget of CPUs, where one pod consumes one CPU, to minimize latency. In our current design, a running job periodically exports its current performance and/or power metrics to a shared file system accessible by PAX. Upon receiving new data, the scheduler will use user-defined optimization objectives to select the relevant parameters for optimization. This step is done by the ax-platform, where a custom Bayesian optimization [17] phase is initiated to select the Kubernetes parameters that optimize the chosen objective - i.e. minimizing P99. The scheduler communicates directly with Kubernetes API client to apply the parameter changes in the cluster.

Bayesian optimization (BO) is a black-box tuning approach that allows one to tune parameters in relatively few iterations by building a smooth model from an initial set of parameterizations (typically quasi-random) in order to predict the outcomes for yet unexplored parameterizations. BO is an adaptive approach where the observations from previous evaluations are used to decide what parameterizations to evaluate next. In PAX, the parameters include the number of replicas for each pod type and the specific node on which each pod type is deployed; this is determined by the cluster configuration. After these parameters are tuned, the objective function in BO will be to evaluate the resulting P99 latency in order to make better predictions on how pod scaling and node placement can be used to minimize latency.

3 EXPERIMENTAL FINDINGS

3.1 Setup

To expose the opportunities for improvement, we compare our approach against two modern Kubernetes schedulers: 1) the Horizontal Pod Autoscaler (HPA) [26] as it is the industry-standard scheduler provided by Kubernetes and 2) a modern research system, Cilantro [6], which has shown performance benefits when compared to systems such as Quasar [11], Minerva [30], Parties [9], and other evolutionary algorithms.

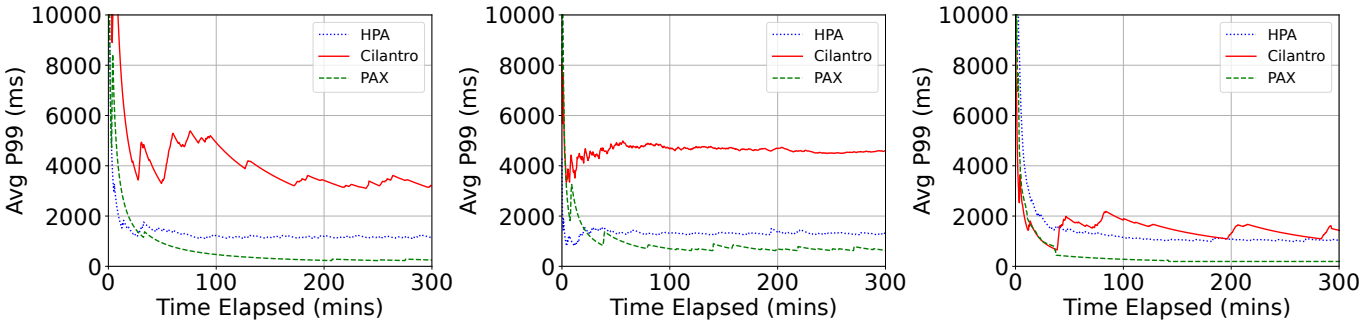
HPA is a widely-used default scheduler that automatically adjusts pod replicas based on current resource usage (such as CPU utilization, memory, etc), therefore enabling Kubernetes to scale the number of pods up and down as demand changes. We follow standard procedures to create a metrics server [27] API to collect pod resource usage, which is then pushed into HPA for scaling decisions. Cilantro is a research system that utilizes a feedback-based online learning approach for performance aware resource allocation. As a comparison point, we replicate Cilantro's experimental environment for the hotel reservation benchmark by using its best performing policy, upper confidence bound (UCB). PAX differs from HPA and Cilantro by using the `patch_namespaced_deployment` API call to directly scale both the of the number of pods and the placement of pods on specific nodes. Given our use of `patch_namespaced_deployment`, the search space of PAX is constrained to place all replicas of a pod type to a single node. In future we will explore extending Kubernetes to allow for node placement of per-pod replicas.

Alongside performance, and in order to capture the operational and embodied carbon differences between the three systems, we explore two classes of servers in our experiments as shown in table 1. `Server-2014` is an older server with a CPU that was released in Q3'14 while `Server-2021`'s CPU was released in Q2'21. The CPU technology node is also different, 22 nm vs 10 nm, which impacts its thermal design point (TDP). TDP is an Intel provided metric that represents the average power the processor dissipates under load. Beside these differences, there is a 2X difference in embodied carbon costs that was calculated using ACT [22]. Lastly, due to the age differences, there is a price difference of 10X between purchasing a refurbished `Server-2014` and `Server-2021`.

Our work seeks to expose the performance and TDP differences between these two servers as a performance and energy trade-off when deciding how to best scale and schedule pods. Prior research [14, 39] has shown that older hardware remains viable for

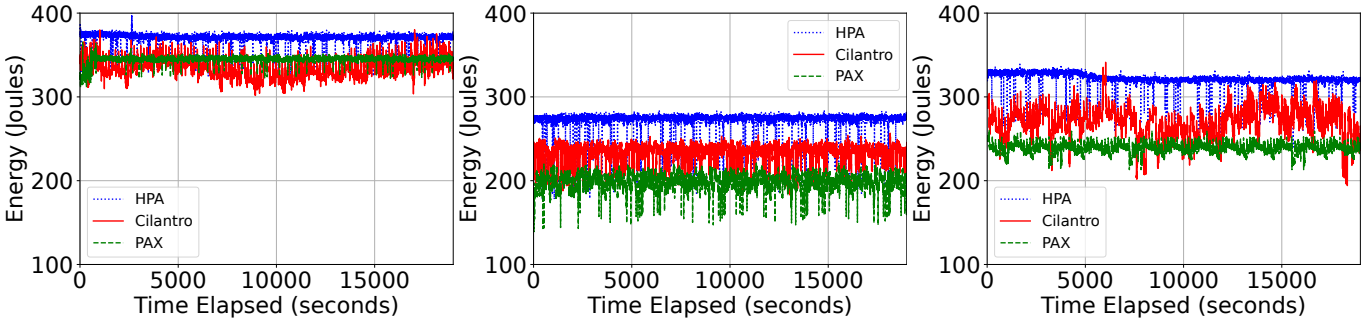
Name	Processor (Intel)	Node	Release	CPUs	TDP (W)	NIC	RAM	SSD	CO2 (kg)	Cost
Server-2014	E5-2630 v3	22 nm	Q3'14	2 x 16	2 x 85	10GbE	128GB	480 GB	118.4	$599 [21]
Server-2021	Xeon Silver 4314	10 nm	Q2'21	2 x 32	2 x 135	40GbE	256GB	960 GB	221.9	$6080 [5]

Table 1. Different hardware explored.



(a) 2X-Server-2021 cluster (b) 4X-Server-2014 cluster (c) 2X-Server-2014, 1X-Server-2021 cluster

Fig. 2. Average P99 latency over 5 hours



(a) 2X-Server-2021 cluster (b) 4X-Server-2014 cluster (c) 2X-Server-2014, 1X-Server-2021 cluster

Fig. 3. CPU package power consumption over 5 hours

workloads under certain load conditions and challenges the assumption that upgrading to newer server generations is always more efficient[1]. In fact, doing so can lead to increases in embodied carbon when functionally working old hardware is prematurely retired and replaced. Further, extending server lifetimes and reusing components has been shown to have significantly more impact on reducing carbon emissions than recycling [28, 38]. As modern cloud applications grow in complexity with multiple interdependent microservices, a more hardware aware resource management strategy is needed. PAX promotes an adaptive system where the right hardware is used for the right workload and moves cloud infrastructure beyond static provisioning and toward carbon-aware scheduling across heterogeneous systems.

3.2 Results

We set up a cluster using CloudLab [15]. As baselines, we first deploy the hotel reservation benchmark on 2X-Server-2021 nodes, which provide a total of 128 allocatable CPUs on a Kubernetes cluster. We deploy the same experiment on 4X-Server-2014 nodes (providing the same number of allocatable CPUs). Lastly, we explore how mixing old and new hardware impacts all three systems by using a cluster consisting of 2X-Server-2014 nodes and 1X-Server-2021 node (128 allocatable CPUs). Each server is deployed with Ubuntu 22.04 LTS running Linux kernel 5.15 and Kubernetes v1.28. Similar to Cilantro [6], we ran each experiment for over five hours and gather P99 latency as generated by wrk2 load generator. This allows us to capture the long-term trend of request latency as each request can interact with multiple sub-services with varying compute costs. We use Kepler [4] to report the dynamic power consumption on each node in the cluster. Kepler uses eBPF probes to gather performance counters and other hardware metrics and exposes this data via Prometheus [2].

Ultimately, in a service-oriented scenario like that modeled by DeathStarBench, one wants to ensure that most clients (99%) obtain timely responses to their requests (low latency). To alleviate

[1]Due to advances such as better performance-per-watt

Cluster Config	System	Avg P99 (ms)	Avg Power (W)	System	Avg P99 (ms)	Avg Power (W)	System	Avg P99 (ms)	Avg Power (W)
2X-Server-2021	**HPA**	1171	371	**Cilantro**	3169	334	**PAX**	237	345
4X-Server-2014	**HPA**	1305	274	**Cilantro**	5188	235	**PAX**	680	199
2X-Server-2014, 1X-Server-2021	**HPA**	994	321	**Cilantro**	1373	260	**PAX**	193	238

Table 2. Average P99 latency and power at the end of the 5 hour experimental run.

programmer and operator burden, be responsive to changes in demand, and apply to multiple services and compositions of servers, one would like the system to automatically choose the number of replicas of each pod type the service is composed of and map those replicas to the available server CPUs. We view this joint problem as the scheduling task – jointly choosing the number of pod replicas and what server they get placed on. The typical approach taken by Kubernetes decomposes the problem into two independent steps. The first, a component such as HPA or Cilantro chooses and requests the number of replicas of each pod type it deems appropriate. The k8's pod scheduler, aware of the servers (nodes) available, creates the requested number of pod replicas on a particular node. Static constraints can accompany the creation requests on where the pod replicas can be placed with respect to the available nodes [25].

The decision space explodes very quickly. As a simple baseline, to launch p individual pods on n nodes, this configuration space is on the order of $O(n^p)$. In the hotel reservation benchmark, there are 19 pod types and trying to schedule them on 2 nodes results in around half a million possible configurations. Assuming 2 minutes is required to evaluate each configuration, then evaluating all possible configurations will take around 2 years of compute time. Further, suppose one were given a budget of 128 CPUs to utilize in a cluster. In this scenario, the replica of each pod type can grow from a min of 1 to some max value which ensures that the total CPUs used across all pods does not exceed the 128 CPU budget. The search space will grow faster with the different number of replicas required. Given this, previous works have shown that sample efficient techniques like Bayesian optimization (BO) are well-suited for addressing the challenges posed by this type of large search space [6, 12, 19].

Below, we evaluate three schedulers: 1) HPA, 2) Cilantro, and 3) PAX in their net ability to yield a good 99% tail latency (Fig. 2) while measuring power consumption (Fig. 3). Table. 2 lists the average results at the end of each experimental run.

3.3 Comparing schedulers

Fig. 2 shows that all three scheduler's rapidly reduced P99 latency early in the benchmark. They quickly decided that the one-pod-per-microservice starting point was insufficient and scaled pod replicas to improve performance. Fig. 3 illustrates that over the run of the benchmark, HPA and PAX show more stable power behavior while Cilantro displays more variance.

Analyzing and comparing Cilantro and HPA's behaviors helps motivate PAX's design. Overall, Cilantro's UCB algorithm resulted in the poorest performance. To explain why, Fig. 4 illustrates how each scheduler's distribution of pod replicas changes in the first hour of the benchmark for the mixed cluster[2]. Each bar represents the number of pod replicas at a particular moment in time. The colors are ordered in each bar and each color indicates a distinct microservice and each color's size represents the portion of allocatable CPUs assigned to the pod. Inspecting Cilantro, we find that it uses a constant two-minute event timer to reevaluate and select a new pod configuration (Fig. 4a), and this constant reconfiguration may contribute to the variance in its power use. In contrast, while HPA reevaluates configurations even more aggressively (every 15 seconds), it quickly settles on a configuration to which it only makes minor modifications (Fig. 4b). The stability in configuration that HPA settles upon results in its ability to satisfy the benchmark requests with an average P99 latency that is 4X lower than Cilantro. However, this comes at an energy cost as illustrated in Fig. 3.

The above findings lead us to conjecture whether aggressive reconfiguration is even necessary, as HPA roughly settles on one configuration for the entire benchmark. Perhaps one just needs to spend the effort to find a good configuration and then stick to it. To explore this, PAX adopts a simpler approach by using BO with a fixed number of trials to find configurations that converge and minimize tail latency. The best configuration found in these trials is then used for the remainder of the benchmark. Fig. 4c illustrates an example of this. Despite the simplicity of this approach, Fig. 2 shows that lowered P99 latency by 5X when compared to HPA and Cilantro, both of which have the opportunity to adapt over the entire benchmark run. Clearly, adversarial scenarios could be devised that would expose PAX's lack of adaptability. However, the results are a good starting point for devising a more robust scheme that intelligently adapts only when needed. Furthermore, PAX's use of BO to navigate the more complex joint configuration space opens the opportunity to exploit hardware heterogeneity, as discussed next.

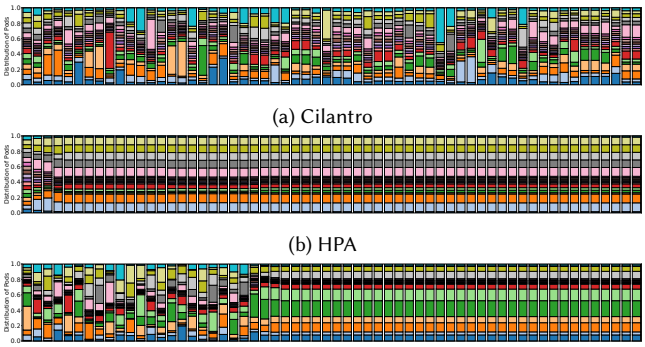
3.4 Explicitly Exploiting Hardware Heterogeneity

Given that PAX searches the joint space of pod replicas and their node placement, it then follows for us to ask what impact there is to purposefully exploit nodes of different technology generations. To this end, we evaluate all three scheduling approaches on a mixed cluster consisting of 2X-`Server-2014` and 1X-`Server-2021` nodes.

We conjectured that PAX's design enables it to find interesting configurations that could spread the pods to the different nodes in order to exploit tradeoffs in the technology generations. In particular, can the mixed cluster be used to find a configuration that cannot be easily found on either homogeneous cluster? While the results we have found support this conjecture, they are neither complete nor fully explained. Instead, they suggest an avenue for future exploration.

Before examining PAX with respect to the mixed cluster, we first compare HPA and Cilantro. On that cluster, HPA obtained a 15% lowered P99 latency and 15% reduced power when compared to the

[2]Similar behavior was seen for the other clusters and the remaining time



(a) Cilantro

(b) HPA

(c) PAX

Fig. 4. Pod replica scaling on 2X-Server-2014, 1X-Server-2021 cluster. The colors are ordered and different colors within each bar indicate distinct pods. The size of each color is representative of the proportion of CPU resources that each pod has been allocated. For brevity, only the first hour of the entire experiment is shown. Note, due to complexities of gathering pod placement data, this graph only shows a coarse-grained sampling.

	HPA		Cilantro		PAX	
Microservice	**Replicas**	**Node**	**Replicas**	**Node**	**Replicas**	**Node**
consul	1	2021-A	4	2021-B	6	2021-B
frontend	7	2021-A, 2021-B	4	2021-A, 2021-B	13	2021-A
jaeger	1	2021-B	17	2021-A, 2021-B	14	2021-A
search	6	2021-A, 2021-B	6	2021-A, 2021-B	17	2021-A
user	1	2021-A	4	2021-A, 2021-B	1	2021-B
mongodb-user	1	2021-B	2	2021-A, 2021-B	1	2021-B
geo	7	2021-A, 2021-B	29	2021-A, 2021-B	4	2021-A
mongodb-geo	1	2021-B	2	2021-A, 2021-B	1	2021-2
profile	7	2021-A, 2021-B	4	2021-B	7	2021-B
mongodb-profile	1	2021-B	2	2021-A, 2021-B	1	2021-A
memcached-profile	1	2021-B	7	2021-A, 2021-B	3	2021-B
rate	6	2021-A, 2021-B	4	2021-A, 2021-B	2	2021-B
mongodb-rate	1	2021-B	2	2021-A, 2021-B	1	2021-B
memcached-rate	2	2021-A, 2021-B	4	2021-A	1	2021-B
recommendation	6	2021-A, 2021-B	5	2021-A, 2021-B	16	2021-B
mongodb-recommendation	1	2021-A	2	2021-A, 2021-B	1	2021-B
reserve	6	2021-A, 2021-B	5	2021-A, 2021-B	10	2021-B
mongodb-reserve	2	2021-A, 2021-B	2	2021-A, 2021-B	1	2021-A
memcached-reserve	2	2021-A, 2021-B	8	2021-A, 2021-B	26	2021-A

Table 3. A snapshot towards the end of experimental run of the pod replicas and their node placement in the 2X-Server-2021 cluster. 2021-A and 2021-B refer to distinct 2021 servers in the cluster.

	HPA		Cilantro		PAX	
Microservice	**Replicas**	**Node**	**Replicas**	**Node**	**Replicas**	**Node**
consul	1	2021-A	20	2014-A, 2014-B, 2021-A	10	2014-A
frontend	7	2014-A, 2014-B, 2021-A	10	2014-A, 2014-B, 2021-A	5	2014-A
jaeger	1	2021-A	9	2014-A, 2014-B, 2021-A	10	2014-B
search	6	2014-A, 2014-B, 2021-A	5	2014-A, 2014-B, 2021-A	1	2021-A
user	1	2021-A	5	2014-A, 2014-B, 2021-A	3	2014-B
mongodb-user	1	2014-A	3	2014-A, 2014-B, 2021-A	1	2014-B
geo	7	2014-A, 2014-B, 2021-A	7	2014-A, 2014-B, 2021-A	15	2014-B
mongodb-geo	1	2014-A	3	2014-A, 2014-B, 2021-A	1	2014-A
profile	7	2014-A, 2014-B, 2021-A	4	2014-B, 2021-A	1	2014-A
mongodb-profile	1	2021-A	3	2014-A, 2014-B, 2021-A	1	2014-A
memcached-profile	1	2021-A	7	2014-A, 2014-B	26	2021-A
rate	6	2014-A, 2014-B, 2021-A	4	2014-A, 2014-B, 2021-A	2	2014-A
mongodb-rate	1	2014-B	3	2014-A, 2014-B, 2021-A	1	2014-B
memcached-rate	2	2014-B, 2021-A	6	2014-A, 2014-B, 2021-A	19	2014-B
recommendation	6	2014-A, 2014-B, 2021-A	8	2014-A, 2014-B, 2021-A	12	2014-A
mongodb-recommendation	1	2014-A	3	2014-A, 2014-B, 2021-A	1	2014-B
reserve	6	2014-A, 2014-B, 2021-A	3	2014-A, 2014-B, 2021-A	8	2021-A
mongodb-reserve	1	2021-A	3	2014-A, 2014-B, 2021-A	1	2014-A
memcached-reserve	2	2014-B, 2021-A	4	2014-A, 2014-B	1	2014-A

Table 4. Pod replicas and their node placement in the 2X-Server-2014, 1X-Server-20221 cluster. 2014-A and 2014-B refer to distinct 2014 servers and 2021-A is the 2021 server.

2X-Server-2021 cluster, and a 31% lowered P99 while using 17% **more** power compared to 4X-Server-2014 cluster. This suggests the mixed cluster contained interesting configuration points that interpolate between both homogeneous clusters and that HPA was able to find them by making the slight adjustments as shown in Fig. 4b.

Interestingly, Cilantro benefited considerably more from the mixed cluster as it was able to obtain 2.3X lower P99 and 28% lower power compared to 2X-Server-2021 cluster and 3.78X lower latency and 9% **more** power compared to 4X-Server-2014 cluster. Fig. 2c also illustrates another phenomenon where, around 40 minutes into the experiment, Cilantro seemed to be on track to achieve P99 latency that is even lower than HPA. However, instead of sticking with this configuration, Cilantro deviates into another sub-optimal configuration that caused a large increase in its P99 latency. This is different from the decisions of PAX, which decidedly settled on a performant configuration for the rest of the experimental run.

In contrast to HPA and Cilantro, which only scale replicas, PAX bypasses the default Kubernetes scheduler to both scale pod replicas and specify pod-to-node placement that can achieve over 5× lower P99 latency. However, it is important to note that the current implementation of PAX crudely uses a fixed trial budget. Given the initial stochasticity of BO, PAX could have been stuck in a poor performning local minimum configuration. However, Fig. 2 illustrates the benefits of the PAX approach across all three clusters and suggests that BO was able to approximate an objective function that is smooth and well structured so as to quickly converge to a performant configuration for the hotel reservation benchmark.

To shed more light on the nature of the configurations that the schedulers found on the mixed cluster versus the homogeneous clusters, we present Table 3 and Table 4. Both tables show snapshots of a single configuration across all three schedulers taken towards the end of the experimental run on the 2X-Server-2021 and 2X-Server-2014, 1X-Server-2021 clusters. The row colors are used to highlight pods that support similar functions, e.g. reservation service that contains both a MongoDB and Memcached component. Note that while HPA and PAX's configurations are stable for the majority of the experimental run, a snapshot towards the end is representative of its the best performing configuration. However, this is unclear for Cilantro due to its constant 2 minute reconfiguration window.

Interestingly, we find that HPA did not fully utilize all 128 CPUs as it only scaled up certain pods up toward a limit of 6 or 7, indicating a more conservative policy. However, HPA was still able to outperform Cilantro. We hypothesize that this is partly due to Cilantro constantly changing reconfigurations. These results also reveal a limitation of HPA: it does not provide interfaces to specify a CPU budget, and these tables show that PAX was able to aggressively scale up similar pods as HPA and achieve even better performance. Further, as pod replicas are scaled up by HPA and Cilantro, Table 3 and Table 4 illustrate the default behavior of the Kubernetes scheduler as it will try to spread the pods amongst available nodes. This behavior is different in PAX as it both scales replicas and bypasses Kubernetes to place these replicas on a single node. Table 3 and Table 4 also reveal that Cilantro and PAX found different pod replica counts for both hardware clusters, which underscores the design of both systems to adapt to their operating environment while optimizing for end-to-end latency.

Our preliminary analysis does not provide a definitive explanation for why the mixed cluster out performs the homogeneous clusters. However, combining Table 3 and Table 4 with experimental data in

Fig. 2 and Fig. 3 clearly highlights new opportunities for future work to systematically explore and quantify these behaviors in order to better promote hardware reuse and improve system sustainability. It should be noted that there is a large and complex configuration space and our work highlights the need for further ablation studies. Our results also suggest a systematic and manual path that one can take in a comprehensive study to investigate this phenomenon by conducting manual grid searches by slowly perturbing both bad and good configurations. Other approaches include profiling to find interesting underlying dependency behavior between microservices or fine-grained introspection on the BO process to unravel important tunable parameters.

4 DISCUSSION

In this paper, we have used a single hotel reservation benchmark to reveal performance and power efficiency opportunities in Kubernetes resource allocation. There is still a need to validate that the same results hold for the other suite of benchmarks from DeathStarBench [18]. Further, the full potential of leveraging black-box tuning for this resource reallocation task is to explore it under the context of a multi-tenant setting which consists of different applications running multiple microservices across a heterogeneous cluster with different hardware types. One of the main benefits of the BO approach is that it can converge to its optimization objective in relatively few iterations.

Expanding on this further, ablation studies need to be done to quantify the benefits of BO under more performance metrics such as P50, and P90, as well as alongside different carbon metrics other than power consumption. Besides BO, the ax-platform provides a suite of other black-box tuning algorithms [3]. A study of these algorithms should yield interesting results of how they perform distinctly under differing application requirements and single vs multi-tenant settings or performance and/or power efficiency objectives.

We have demonstrated the utility of PAX using somewhat simplistic objectives such as P99. However, the possibilities of customizing these objectives further is ripe for exploration, e.g. using new combinations of performance/energy or known metrics such as energy-delay-product [7, 24]. One can also imagine developing a rich set of objectives that capture preferences that a service operator might have. In this way, PAX can be reconfigured as application priorities change.

There is also a need to quantify how PAX behaves in unpredictable environments such as unexpected load spikes or changing patterns of workloads. Cilantro tries to address this via its uncertainty-aware design and using 2 minute triggers for reallocation to dynamically adjust to changing loads. However, our results illustrate the performance limitation of this approach. To address this, one can use techniques such as maintaining a running window of average load that acts as triggers for the learning phase or provides simple heuristics (such as HPA) as a fallback when extreme fluctuations in load appear. Recent studies of certain data center applications [37, 40] have also shown that they experience repetitive mean demand curves which gradually change in offered loads over extended periods, ranging from hours to days. Such stability arises from recurring diurnal patterns and the use of load balancers [33, 40]. This suggests that learning phases can leverage historical usage patterns to strategically select when to do pod reallocation.

4.1 Multi-Objective Optimization

The ax-platform also provides an experimental feature [1] for multi-objective optimization (MOO) [3]. As an example, MOO can optimize both P99 latency and power at the same time. This is another interesting avenue to explore as one scenario can involve setting up a larger experimental cluster, such as 4X-`Server-2014` and 2X-`Server-2021` nodes (256 CPUs altogether) and utilize MOO to investigate if it could allocate 128 CPUs from this cluster and find a configuration of pods and nodes that outperformed the results in Table 2 while using the least energy (either by reducing the number of nodes it used or through strategic placement of pods on power-efficient nodes). We believe the potential for MOO can be further extended to optimize for other scenarios such as various proxy metrics, applications with service-level objectives (SLO), and for multi-tenant applications that have diverging requirements. Exposing MOO to end users also opens up interesting systems design challenges such as creating interfaces that enable the intuitive specification of multiple, potentially conflicting, optimization criteria.

5 CONCLUSION

In this work, we presented PAX, a performance and energy aware Kubernetes scheduler designed to optimize microservice placement in heterogeneous clusters. PAX integrates black-box Bayesian optimization to learn scheduling policies that account for diverse hardware. Our results highlight a largely untapped opportunity at the intersection of cluster scheduling and sustainability: by leveraging existing underutilized hardware and aligning workload demands with server capabilities, data centers can reduce carbon emissions without sacrificing performance.

6 ACKNOWLEDGMENTS

We would like to thank our shepherd Monica Vitali for help in preparing the final version. This work was supported by the Red Hat Collaboratory at Boston University under the award Red Hat Collaboratory Research Incubation Award, Discovering Opportunities for Optimizing OpenShift Energy Consumption, #2024-01-RH04.

REFERENCES

[1] 2025. [GENERAL SUPPORT]: Getting extremely slow in multi-objective optimization 2859. https://github.com/facebook/Ax/issues/2859. [Accessed April 28, 2025].
[2] 2025. Prometheus. https://prometheus.io/. [Accessed April 27, 2025].
[3] 2025. Welcome to Ax Tutorials. https://ax.dev/docs/tutorials/. [Accessed April 28, 2025].
[4] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. 2023. Kepler: A Framework to Calculate the Energy Consumption of Containerized Applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 69–71. https://doi.org/10.1109/CLOUD60044.2023.00017
[5] AvaDirect. 2025. Supermicro SuperServer SYS-220P-C9R. https://www.avadirect.com/Supermicro-SuperServer-SYS-220P-C9R-T-3rd-Generation-Intel-Xeon-Scalable-Processors-SATA-SAS-2U-Rackmount-Server-Computer/Configure/14501891?srsltid=AfmBOooTmHNQR4nu94ErwmC6sRPkBCnuNKcQxr-gJBNzDOTPL8I8j4MM [Accessed May 18, 2025].
[6] Romil Bhardwaj, Kirthevasan Kandasamy, Asim Biswal, Wenshuo Guo, Benjamin Hindman, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2023. Cilantro:

Performance-Aware Resource Allocation for General Objectives via Online Feedback. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 623–643. https://www.usenix.org/conference/osdi23/presentation/bhardwaj

[7] 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. 2000. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro* 20, 6 (Nov. 2000), 26–44. https://doi.org/10.1109/40.888701

[8] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *Commun. ACM* 59, 5 (April 2016), 50–57. https://doi.org/10.1145/2890784

[9] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 107–120. https://doi.org/10.1145/3297858.3304005

[10] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 153–167. https://doi.org/10.1145/3132747.3132772

[11] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 127–144. https://doi.org/10.1145/2541940.2541941

[12] Yi Ding, Alex Renda, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. 2022. Cello: Efficient Computer Systems Optimization with Predictive Early Termination and Censored Regression. https://doi.org/10.48550/ARXIV.2204.04831

[13] Yi Ding and Tianyao Shi. 2024. Sustainable LLM Serving: Environmental Implications, Challenges, and Opportunities : Invited Paper . In *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*. IEEE Computer Society, Los Alamitos, CA, USA, 37–38. https://doi.org/10.1109/IGSC64514.2024.00016

[14] Han Dong, Sanjay Arora, Orran Krieger, and Jonathan Appavoo. 2024. 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) *(SYSTOR '24)*. Association for Computing Machinery, New York, NY, USA, 83–90. https://doi.org/10.1145/3688351.3689158

[15] 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. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[16] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power Provisioning for a Warehouse-Sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (San Diego, California, USA) *(ISCA '07)*. Association for Computing Machinery, New York, NY, USA, 13–23. https://doi.org/10.1145/1250662.1250665

[17] Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization. arXiv:1807.02811 [stat.ML] https://arxiv.org/abs/1807.02811

[18] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/3297858.3304013

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

[20] Gil Tene. 2024. wrk2: a HTTP benchmarking tool based mostly on wrk. https://github.com/giltene/wrk2 [Accessed Oct 22, 2024].

[21] GizmoGenie. 2025. CISCO UCS C220 M4 2x INTEL XEON E5-2630 V3 2.4 GHz 128 GB RAM. https://www.ebay.com/itm/175458691943?chn=ps&mkevt=1&mkcid=28&srsltid=AfmBOoqD-uwgiywqgtDR_UmN5ID8oj0TxnQjduCpkl273GXgMxkRIF50hU82 [Accessed May 18, 2025].

[22] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. https://doi.org/10.1145/3470496.3527408

[23] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2020. Chasing Carbon: The Elusive Environmental Footprint of Computing. arXiv:2011.02839 [cs.AR]

[24] M. Horowitz, T. Indermaur, and R. Gonzalez. 1994. Low-power digital design. In *Proceedings of 1994 IEEE Symposium on Low Power Electronics*. 8–11. https://doi.org/10.1109/LPE.1994.573184

[25] Kubernetes. 2025. Assigning Pods to Nodes. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/ [Accessed June 24, 2025].

[26] Kubernetes. 2025. Horizontal Pod Autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ [Accessed April 27, 2025].

[27] Kubernetes. 2025. Resource metrics pipeline. https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/. [Accessed April 27, 2025].

[28] Jialun Lyu, Jaylen Wang, Kali Frost, Chaojie Zhang, Celine Irvene, Esha Choukse, Rodrigo Fonseca, Ricardo Bianchini, Fiodar Kazhamiaka, and Daniel S. Berger. 2023. Myths and Misconceptions Around Reducing Carbon Embedded in Cloud Platforms. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 7, 7 pages. https://doi.org/10.1145/3604930.3605717

[29] Meta Platforms, Inc. 2025. Adaptive Experimentation Platform. https://ax.dev [Accessed April 27, 2025].

[30] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. 2019. End-to-end transport for video QoE fairness. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 408–423. https://doi.org/10.1145/3341302.3342077

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

[32] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2022. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. arXiv:2204.05149 [cs.LG] https://arxiv.org/abs/2204.05149

[33] 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. 2013. Scaling Memcache at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, Lombard, IL, 385–398. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/nishtala

[34] Brian Ramprasad, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. 2021. Sustainable Computing on the Edge: A System Dynamics Perspective. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21)*. Association for Computing Machinery, 64–70.

[35] Red Hat. 2025. Red Hat Emerging Technologies. https://next.redhat.com/projects-full/ [Accessed June 23, 2025].

[36] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 3645–3650. https://doi.org/10.18653/v1/P19-1355

[37] 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. 2020. Twine: A Unified Cluster Management System for Shared Infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 787–803. https://www.usenix.org/conference/osdi20/presentation/tang

[38] Jaylen Wang, Daniel S. Berger, Fiodar Kazhamiaka, Celine Irvene, Chaojie Zhang, Esha Choukse, Kali Frost, Rodrigo Fonseca, Brijesh Warrier, Chetan Bansal, Jonathan Stern, Ricardo Bianchini, and Akshitha Sriraman. 2024. Designing Cloud Servers for Lower Carbon. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 452–470. https://doi.org/10.1109/ISCA59077.2024.00041

[39] Jaylen Wang, Udit Gupta, and Akshitha Sriraman. 2023. Peeling Back the Carbon Curtain: Carbon Optimization Challenges in Cloud Computing. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 8, 7 pages. https://doi.org/10.1145/3604930.3605718

[40] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2020. 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, 191–208. https://www.usenix.org/conference/osdi20/presentation/yang