PARM: Adaptive Resource Allocation for Datacenter Power Capping

Haoran Qiu[1] Linghao Zhang[1] Chen Wang[2] Hubertus Franke[2]
Zbigniew T. Kalbarczyk[1] Ravishankar K. Iyer[1]
[1]*University of Illinois Urbana-Champaign* [2]*IBM Research*

Abstract

Energy efficiency is pressing in today's cloud datacenters. Various power management strategies, such as oversubscription, power capping, and dynamic voltage and frequency scaling, have been proposed and are in use by datacenter operators to better control power consumption at any management unit (e.g., node-level or rack-level) without breaking power budgets. This paper systematically investigates the impact of power capping on both latency-critical datacenter workloads and learning-based resource management solutions (i.e., reinforcement learning or RL). We show that even a 20% reduction in power limit (power capping) leads to an 18% degradation in resource management effectiveness (i.e., defined by an RL *reward* function) which causes 50% higher application latency. We then propose PARM, an adaptive resource allocation framework that provides graceful performance-preserving transition under power capping for latency-critical workloads. Evaluation results show that PARM achieves 10.2–99.3% improvement in service-level objective (SLO) preservation under power capping while improving utilization by 3.1–5.8%.

1 Introduction

Over the past decade, the demand for datacenter computing and power has been growing at an increasing rate, driven by the rise of web services and machine learning-related workloads [4]. To improve efficiency (by achieving high levels of utilization), a variety of *power capping* solutions have been leveraged to safely allow oversubscription of available power [5, 13, 35, 17]. Power capping is a mechanism that enforces the power limits of a datacenter at any level via processor voltage/frequency throttling. The capability of power capping also enables sustainability-aware optimization by shifting compute demand to *when* and *where* the carbon intensity is the lowest [1, 25].

However, when power capping is enforced, application performance can be negatively impacted due to processor frequency throttling. Workload-aware power capping is done either in a coarse-grained manner or relies on accurate forecasting and careful workload placement. For instance, Meta's Dynamo [35] relies on predefined workload priority groups and throttles the whole server based on the workload priority. For unknown workloads and their server placements, throttling at a coarse granularity (the entire server) would impact latency-critical workloads (e.g., user-facing web services). Microsoft proposes per-VM capping [17] but suffers from misprediction (due to dynamic workload patterns) and requires changes to the cluster job placement scheduler.

In addition, numerous learning-based resource management solutions [19, 20, 22, 33, 28, 36, 38, 10, 24] have been proposed and are used in cloud datacenters for performance-aware resource allocation (e.g., Sizeless [10]) and autoscaling (e.g., AutoPilot [28] and DeepScaling [33]). Unfortunately, without coordination with datacenter power management systems, such ML/RL-based approaches suffer from suboptimal decisions (worse performance due to frequency throttling) or even cascading degradation due to scaling out and thus a need for further reducing power demand (more throttling).

Our Contribution. We present a measurement study on the impact of datacenter power capping on both application performance and an RL-based autoscaler (in both training and policy-serving). We then introduce PARM, an adaptive resource allocation framework that provides graceful transitions for latency-critical workloads under power capping. Evaluation results show that PARM achieves

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans.

10.2–99.3% improvement in service-level objective (SLO) preservation for latency-critical workloads under power capping while improving CPU utilization by 3.1–5.8% (compared to a vanilla RL-based autoscaler from FIRM [20]).

2 Background and Motivation

2.1 RL-based Resource Management

Due to the sequential nature of the decision-making process, RL is well-suited for learning resource management policies by providing a tight feedback loop for exploring the state-action space and generating optimal policies without relying on inaccurate assumptions (i.e., heuristics or rules) [19, 33, 20]. In addition, since the decisions made for workloads are highly repetitive, an abundance of data is generated to train such RL algorithms even with deep neural networks. By directly learning from the actual workload and operating conditions to understand how the allocation of resources affects application performance, the RL agent can optimize for a specific workload and adapt to varying conditions in the learning environment. RL has been shown to automate resource management and outperform heuristics-based approaches in terms of meeting workload SLOs and achieving higher resource utilization [20, 16, 37, 22, 33, 27, 24, 23].

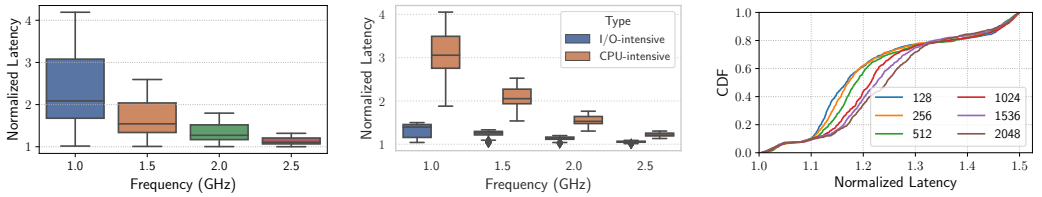
Specifically, we took the open-source implementation of an RL-based workload autoscaler from FIRM [20], which is a state-of-the-art RL-based autoscaling solution on container orchestration platforms such as Kubernetes [24] and OpenWhisk [22]. FIRM uses an actor-critic RL algorithm called DDPG [18]. The RL agent monitors the system- and application-specific measurements and learns how to scale the allocated resources horizontally (by adjusting the number of containers) and vertically (by adjusting the resource limits). Table 1 shows the model's ***state*** and ***action*** spaces. The goal is to achieve high resource utilization (RU) while maintaining application SLOs (if there are any). SLO preservation (SP) is defined as the ratio between the SLO metric [1] and the measured metric. If no SLO is defined for the workload (e.g., best-effort jobs) or the measured metric is smaller than the SLO metric, $SP = 1$. The ***reward function*** is then defined as $r_t = \alpha \cdot SP_t \cdot |\mathcal{R}| + (1-\alpha) \cdot \sum_{i \in \mathcal{R}} RU_i$, where $\mathcal{R}$ is the set of resource types and $\alpha = 0.7$ in all our experiments. The reward represents the loss/gain when transitioning to the current state, and thus the agent's goal is to optimize its policy so as to maximize the expected cumulative reward in controlling the resource autoscaling.

Table 1: RL state-action space of FIRM [20].

State Space (s_t)
Resource Limits (CPU, RAM), Resource Utilization (CPU, Memory, I/O, Network), SLO Preservation Ratio (Latency, Throughput), Observed Load Changes
Action Space (a_t)
Resource Limits (CPU, RAM), Number of Replicas

2.2 Impact of Power Capping

Power capping is a mechanism that ensures the power drawn by a datacenter at any level (e.g., server-level or rack-level) stays below a predefined power budget (i.e., a limit) [5, 13, 35, 17]. Power capping has been mainly leveraged by datacenter operators to safely oversubscribe available power [5, 13, 35, 17] or carbon footprint optimization (by load and compute/power demand shifting) [1, 25]. At the core of power capping is a monitoring loop, which takes in power readings, computes the amount of power capping needed (i.e., a power limit), and then enforces the capping in a variety of techniques depending on the scale and type of the hardware component. From the intra-server perspective, capping is enforced by throttling processors (e.g., Intel RAPL [34] uses voltage/frequency down-scaling to limit power consumption on a server node).



(a) All applications. **(b)** CPU vs. I/O-intensive. **(c)** Sensitivity to CPU allocations.

Figure 1: Performance degradation analysis under power capping and core frequency reduction. Latency for frequency reduction to 1.0–2.5 GHz is normalized to the frequency of 3.0 GHz

Setup. We generated 1000 synthetic applications based on the open-source application generator, using production serverless function segments in Sizeless [10] as inputs. We focus on serverless

[1] An SLO metric can be either request serving latency (e.g., the 99th percentile of the requests are completed in 100ms) or throughput (e.g., request processing rate is no less than 100/s).

workloads as an example because they are highly dynamic and thus require autoscaling. The 16 representative function segments are based on a survey of 89 industry use cases of serverless computing applications [11], including CPU-intensive tasks (e.g., floating-point number computation), image manipulation, text processing, data compression, web serving, ML model serving, and I/O services (e.g., read, write, and streaming). For RL agent training and inference, we used real-world datacenter traces [29] released by Microsoft Azure, collected over two weeks in 2021. All applications were deployed on a five-node OpenWhisk [12] cluster on a local rack with five physical nodes, each of which contains an Intel x86 Xeon E5 processor 2.0 GHz (maximum 3.0 GHz) with 56 CPU cores and 500 GB RAM. We deployed FIRM (i.e., the RL agent for autoscaling on OpenWhisk) on a separate node in the same cluster by following [22].

Impact on Workload Performance. There is a linear relationship between the power consumption of a server *power* and the core frequency f (i.e., $power \propto fV^2$) at any voltage V [5, 26]. We run workloads at different frequencies starting from the base frequency to the Turbo frequency that the server supports (i.e., 1.0 to 3.0 GHz). We then measure the end-to-end latency of each application under the default resource allocation in OpenWhisk (i.e., 256 millicpu and 256 MB RAM). As shown in Fig. 1(a), the average latency increase across different applications is 1.1–2.4× for frequency reduced to 1.0–2.5 GHz. The 99th percentile latency degradation is 1.3–4×. CPU-intensive workloads are more sensitive to core frequency reduction under power capping, as shown in Fig. 1(b). For instance, the degradation for CPU-intensive workloads is 2.2× worse than that of I/O-intensive workloads at 2.0 GHz. In addition, performance degradation exists at all CPU allocation levels and workloads with higher CPU allocations tend to experience slightly worse degradation under power capping. For instance, the median degradation increases from 1.1 to 1.3× for applications with CPU allocation varying from 128 to 2048 millicpus at frequency 2.0 GHz (as shown in Fig. 1(c)).

Impact on RL-based autoscaling agents is deferred to Appendix B due to page limit.

Key Takeaway. Datacenter power capping has a negative impact on both the workloads running on the core-frequency-throttled servers and the learning-based resource management solutions. *A new design is needed to adaptively capture the impacts of power capping and augment existing learning-based resource managers in providing latency-critical workloads with seamless transitions to continuously operate and meet SLOs under power capping.*

3 PARM

We introduce PARM, an adaptive resource allocation framework that monitors power capping and provides graceful (SLO-preserving) transitions for latency-critical workloads by actively re-scaling the resource allocation. We propose two designs (alternative design in Appendix C), depending on whether core frequencies are measurable and accessible for the application deployment (i.e., in containers/VMs). We then integrate, implement, and evaluate both designs of PARM on OpenWhisk [12], a production-grade open-source serverless platform (as described in §2.2). Fig. 2 shows an overview of the architecture and adaptive resource allocation workflow in PARM under power capping.

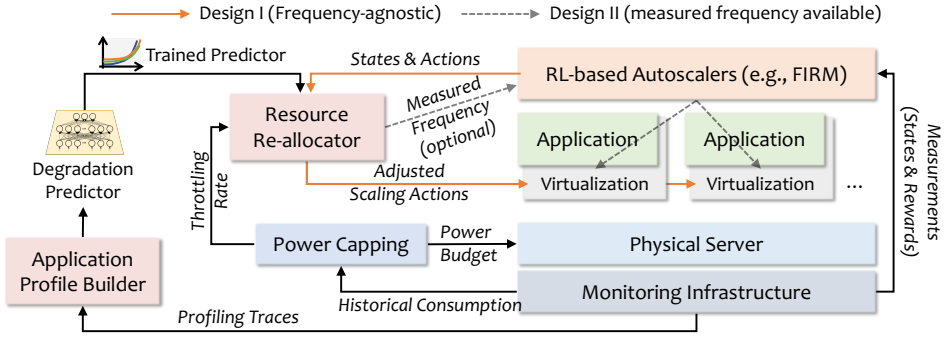


Figure 2: Overview of the adaptive resource allocation workflow in PARM.

In the first design, we assume that core frequency measurements are not available because, in shared cloud platforms, VM/container-to-core assignments can be dynamic and non-dedicated (time-sharing). PARM serves as a co-agent that runs together with the original RL agent (e.g., from FIRM [20]) that is designed and trained for autoscaling. PARM consists of two main components: (1) a *Degradation Predictor* that is trained based on the application profile builder, and (2) a *Resource Re-allocator* that adjusts the scaling action from the RL agent to continuously meet application SLOs. We name the first design the "frequency-agnostic" design, which does not require any changes to the RL agent.

Degradation Predictor. We formulate a *Multi-target Regression Modeling* problem to predict the degradation of application performance under power capping. The degradation predictor outputs a

trained degradation model that can predict the latency changes and CPU utilization changes (both in percentage) compared to no power capping. We start with a simple fully-connected neural network with three layers of 64 neurons (with ReLU as the activation function) trained for 80 epochs. The inputs to the predictor include the power capping percentage (from *Power Capping* module in Fig. 2), CPU allocation, memory allocation, CPU utilization, and latency while the outputs are the predicted CPU utilization and latency if there were no power capping. The degradation predictor training is on application profiles by setting different power limits from [10%, 100%) using a step size of 5%.

Resource Re-allocator. Based on the what-if predictions (i.e., the predicted CPU utilization and latency if there were no power capping) from the degradation predictor, the resource re-allocator replaces the CPU utilization and latency variable in the RL state vector at the current step and queries the RL policy network. The output resource scaling action then represents the corrected action assuming that there is no power capping. The corrected scaling action is then applied to the horizontal and vertical resource scaler in OpenWhisk for actuation.

4 Evaluation

Our evaluation addresses the following main questions: (a) Can the degradation predictor in PARM's frequency-agnostic design achieve high accuracy regarding latency and utilization prediction? (See Appendix E) (b) Can PARM provide adaptive and performance-preserving resource allocation based on the degradation prediction? (c) Can PARM's alternative design converge with the RL policy now conditioning on the measured core frequency? (d) How does the alternative design compare with the frequency-agnostic design in terms of both SLO preservation and resource utilization?

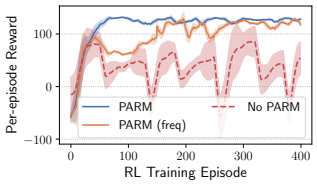


Figure 3: Training convergence analysis of different RL agents.

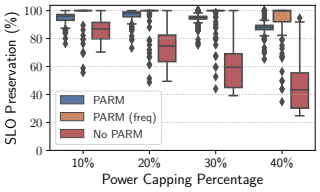


Figure 4: SLO preservation ratio under power capping.

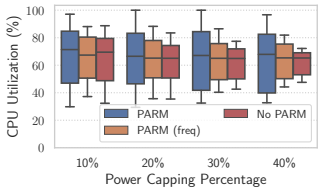


Figure 5: Utilization comparison under power capping.

We evaluate PARM's effectiveness in workload autoscaling when handling different extents of power capping during policy training and serving stages. Fig. 3 shows the training convergence analysis. Both the RL agent of PARM with measured core frequency (i.e., the alternative design, labeled as "PARM (freq)") and the RL agent of PARM in the frequency-agnostic design can converge. However, the RL agent of PARM (freq) has slower convergence compared to that of PARM (frequency-agnostic) by around 90 episodes because PARM (freq) essentially learns a policy that is conditioning on the current frequency in the state vector while PARM (frequency-agnostic) does not require changes to the RL states and its training is under no power capping. A larger state-action space leads to slower convergence behavior. The vanilla RL agent (FIRM [20], labeled as "No PARM") is unable to converge due to environment non-stationarity since the agent is unaware of power capping or core frequency reduction. Especially when there is a power capping change in the RL environment, there can be up to 300% reward drop, which indicates that the learned policy cannot be transferred to an environment with a different level of power capping.

For RL policy-serving, we evaluate the SLO preservation and the CPU utilization under power capping, as the two are the direct factors specified in the reward function. As shown in Fig. 4, PARM achieves 10.2–99.3% improvement in SLO preservation compared to the vanilla RL-based autoscaler from FIRM [20] because of the resource re-allocation with degradation predictor (i.e., the only difference between PARM and the RL agent in FIRM). We also find that as the MSE of degradation prediction increases to 0.21 for 40% capping (recall Fig. 8), the average SLO preservation percentage drops to 87.5% ($>$ 47.9% for "No PARM", though). In the alternative design, PARM (freq) achieves an additional 2.4–3.7% higher SLO preservation than PARM but has worse SLO preservation in tail cases (PARM (freq) has 1.5–2.6× higher variance than PARM), especially at the transition when power capping changes. As shown in Fig. 5, PARM improves 3.1–5.8% CPU utilization on average but results in higher utilization variation (mainly due to degradation predictor's over- or under-prediction). The difference in utilization is not significant because the utilization increase (with lower core frequencies) is offset by the utilization decrease with adjusted scaling (scaling out/up) in reaction to power capping. *Overall, we show that the frequency-agnostic design of* PARM *is preferred to the alternative design because of faster convergence, more effective performance preservation at the tail, and a more practical assumption about core frequency availability.*

A Discussion and Future Work

This paper proposes and evaluates an adaptive performance-preserving solution under power capping from the perspective of *resource management* but not from the power capping module itself (an independent block in Fig. 2) in the cloud datacenter. While we show how *power management* affects resource management in this paper, on the other hand, resource management can also influence the decisions of power management. A holistic design that coordinates both resource and power management is needed to achieve optimality in both performance and power.

Active Resource Management to Save Power. There has been a line of work focused on efficient resource management for power consumption optimization [2, 3, 15, 8, 31, 7, 9, 21]. However, even with the existing efforts, power capping is *still needed* for the purpose of safe over-subscription of available power and carbon footprint optimization (by spatial/temporal load shifting). In contrast to existing work to optimize power consumption, PARM adopts a learning-based resource re-scaling solution for latency-critical workloads to preserve their performance SLOs under power capping.

Coordination with Power Management. Datacenter power management systems typically adopt a *top-down* approach for efficiency. For instance, Dynamo [35] monitors the entire power hierarchy and makes coordinated control decisions to provision power. Similarly, Azure's power capping sets a power budget for each chassis, where each server is then allocated its even share of the chassis budget [17]. PARM re-provisions resources to latency-critical workloads under power capping (within the power limit set for the server). However, we believe that a *bottom-up* approach should be explored for both performance-aware and power-aware workload management. PARM's degradation predictor (using the what-if analysis) can be used to estimate server-specific power limits to guide power capping and set heterogeneous power limits across the power management hierarchy.

Summary and Future Work. In this paper, we characterize the impact of power capping on both workload performance and learning-based resource management solutions. We introduce PARM, an adaptive resource allocation framework with RL that provides graceful (SLO-preserving) transitions for latency-critical workloads by actively re-provisioning resources under power capping. Preliminary results show that, compared to vanilla RL-based autoscaler (i.e., FIRM), PARM achieves 10.2–99.3% improvement in SLO preservation while saving 3.1–5.8% CPU utilization. In the future, we are evaluating the generalizability of PARM on larger-scale applications (e.g., microservices) and real-world applications. In addition, we are working on extending PARM to co-design a holistic, performance-aware power management and resource management framework that leverages the spatial and temporal flexibility of best-effort workloads across the datacenter.

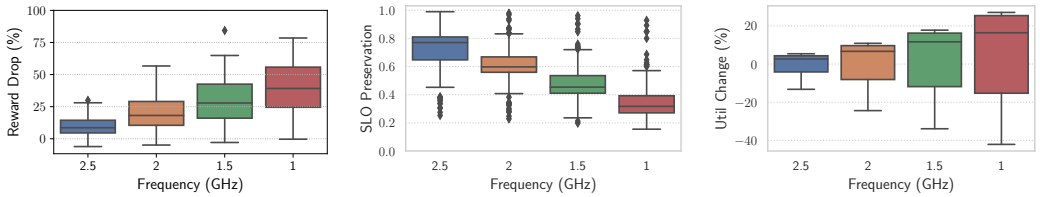
B Characterization Study

To study the impact of power capping on RL-based autoscaling agents, we quantify the RL agent performance by the *per-episode reward* (recall §2.1). We divide the experiments into RL policy training and policy-serving stages. For RL policy-serving evaluation, we train the RL agent with no power capping (i.e., the baseline for comparison) and test the RL agent policy-serving performance when managing the serverless function autoscaling in different power-capping environments. In Fig. 6, we quantify and present (a) the per-episode RL reward drop percentage, (b) SLO preservation ratio, and (c) CPU utilization change (as the Y axes) under power capping. We find that, with the increase of power capping (i.e., a lower power limit and thus lower core frequency level), the reward drop percentage increases, the SLO preservation ratio decreases, and the function container CPU utilization variation (compared to the baseline) increases. For instance, for performance under core frequency 1.5 GHz compared to 2.5 GHz, the median reward drop percentage increases from 8.6% to 27.9%, the median SLO preservation ratio reduced from 0.77 to 0.45, and the median CPU utilization change increases from 2.6% to 16.4%. The reward drop increases close to linearly from 2.5 to 1.5 GHz but becomes 30% worse when reducing from 1.5 to 1.0 GHz. Despite the variation of utilization change, the average utilization increases slightly with the decrease in core frequency (due to more CPU time needed to complete the same amount of CPU cycles). We also find that the lower bound of the utilization is the worst at lower frequencies because of overprovisioning decisions from the RL agent in reaction to power capping.

For RL policy training evaluation, we train the RL agent with power capping (i.e., with different extents of core frequency reduction from 1.0 to 3.0 GHz) and find that the RL policy is not able to converge (as shown in Fig. 3) due to RL environment non-stationarity [22].

C Alternative Design

Given the fact that physical cores are typically shared among VMs/containers, it is challenging for cloud users to get an accurate measurement of the CPU frequency (especially for serverless platforms



(a) RL reward degradation. **(b)** SLO presentation ratio. **(c)** CPU utilization change.

Figure 6: RL policy-serving degradation analysis under power capping and core frequency reduction.

where function containers are dynamically allocated to shared CPU cores) [32, 6, 30]. However, with ML-assisted core frequency estimation (e.g., Kepler [14] for Kubernetes) or for cases when a fixed set of cores is assigned to a VM/container, an alternative design is to include the measured core frequency as part of the RL state vector (as shown in Table 1 with the dashed arrow) during both training and policy-serving. The hypothesis is that the RL agent would be able to learn power-capping-specific autoscaling policies, conditioning on the perceived core frequency. We implement the alternative design in OpenWhisk (by pinning function containers to cores) and show evaluation results in §4.

D Training

In the frequency-agnostic design, the training of the degradation predictor and the RL agent are independent of each other. We train the vanilla RL agent (from FIRM [20]) from scratch with 700 out of 1000 synthetic applications (the remaining ones are used as the test dataset) on the setup as described in §2.2 with no power capping. We train the degradation predictor (supervised learning) using the application profiles from the same set of applications. In the alternative design, we train the refined RL agent from scratch with the same setup as the vanilla RL agent under power capping (i.e., with power limits from [10%, 100%) using a step size of 5%).

E Degradation Predictor Evaluation

We first evaluate the training cost and the prediction accuracy of the changes in latency and utilization under varying power capping scenarios, compared to no power capping. The application pool is split in 7:3 for training and testing. Fig. 7 shows the convergence analysis of the latency/utilization degradation predictor during training. The prediction for utilization converges faster than that for latency and both converge within around 40 epochs ($<$ 2 minutes). Fig. 8 shows the prediction MSE (mean square error) for various testing scenarios. Overall, PARM's degradation predictor has an average of 0.09 and 0.01 MSE for latency and utilization prediction. The prediction MSE for utilization is less than 0.03 in all scenarios. For latency degradation prediction, PARM predicts more accurately for I/O-intensive workloads compared to CPU-intensive workloads since I/O-intensive workloads are less sensitive to power capping (less performance variation) (as shown in §2.2). We also find that the prediction MSE increases as the power capping percentage increases (e.g., 0.21 for 40% power capping), which is due to the fact that application performance has higher variability (especially for tail latency) under higher capping percentages. More intensive power capping (throttling core frequency to $<$1.0 GHz) leads to SLO violations that are unable to be mitigated with scaling up/out and thus we stop at 40%. We next evaluate if such prediction accuracy is sufficient in the end-to-end evaluation of PARM regarding SLO preservation and utilization.

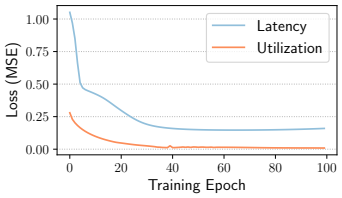


Figure 7: Training loss.

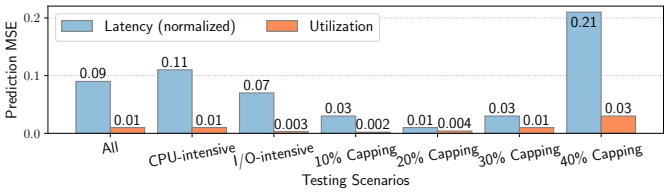


Figure 8: Prediction MSE under various testing scenarios.

References

[1] B. Acun, B. Lee, F. Kazhamiaka, K. Maeng, U. Gupta, M. Chakkaravarthy, D. Brooks, and C.-J. Wu. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2023)*, pages 118–132, 2023.

[2] N. Akhter and M. Othman. Energy-aware resource allocation of cloud data center: Review and open issues. *Cluster Computing*, 19:1163–1182, 2016.

[3] S. Alanazi and B. Hamdaoui. Energy-aware resource management framework for overbooked cloud data centers with SLA assurance. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018. doi: 10.1109/GLOCOM.2018.8647884.

[4] L. A. Barroso, U. Hölzle, and P. Ranganathan. *The Datacenter as a Computer: Designing Warehouse-scale Machines*. Springer Nature, 2019.

[5] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar. The need for speed and stability in data center power capping. In *2012 International Green Computing Conference (IGCC)*, pages 1–10, 2012. doi: 10.1109/IGCC.2012.6322253.

[6] Q. Chen, S. Xue, S. Zhao, S. Chen, Y. Wu, Y. Xu, Z. Song, T. Ma, Y. Yang, and M. Guo. Alita: Comprehensive performance isolation through bias resource management for public clouds. In *International Conference for High-Performance Computing, Networking, Storage and Analysis (SC 2020)*, pages 1–13, 2020. doi: 10.1109/SC41405.2020.00036.

[7] S. Chen, C. Delimitrou, and J. F. Martínez. PARTIES: QoS-aware resource partitioning for multiple interactive services. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)*, page 107–120, New York, NY, USA, 2019. Association for Computing Machinery.

[8] D. Cheng, J. Rao, C. Jiang, and X. Zhou. Elastic power-aware resource provisioning of heterogeneous workloads in self-sustainable datacenters. *IEEE Transactions on Computers*, 65 (2):508–521, 2016. doi: 10.1109/TC.2015.2428695.

[9] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes. Energy-efficient cloud resource management. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 386–391, 2014. doi: 10.1109/INFCOMW.2014.6849263.

[10] S. Eismann, L. Bui, J. Grohmann, C. Abad, N. Herbst, and S. Kounev. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference (Middleware 2021)*, page 248–259, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385343. doi: 10.1145/3464298.3493398. URL https://doi.org/10.1145/3464298.3493398.

[11] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup. Serverless applications: Why, when, and how? *IEEE Software*, 38(1):32–39, 2021. doi: 10.1109/MS.2020.3023302.

[12] A. S. Foundation. OpenWhisk. https://github.com/apache/openwhisk, 2023. Accessed: 2023-09-14.

[13] X. Fu, X. Wang, and C. Lefurgy. How much power oversubscription is safe and allowed in data centers? In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC 2011)*, pages 21–30, 2011.

[14] GitHub. Kepler: Kubernetes efficient power level exporter. https://github.com/sustainable-computing-io/kepler, 2023. Accessed: 2023-09-14.

[15] H. Goudarzi and M. Pedram. Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter. *IEEE Transactions on Cloud Computing*, 4(2):222–236, 2016. doi: 10.1109/TCC.2015.2474369.

[16] S. Kardani-Moghaddam, R. Buyya, and K. Ramamohanarao. ADRL: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. *IEEE Transactions on Parallel and Distributed Systems (TPDS 2020)*, 32(3):514–526, 2020.

[17] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, et al. Prediction-based power oversubscription in cloud platforms. In *2021 USENIX Annual Technical Conference (USENIX ATC 2021)*, pages 473–487, 2021.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In Y. Bengio and Y. LeCun, editors, *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*, 2016. https://arxiv.org/abs/1509.02971.

[19] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNet 2016)*, pages 50–56, New York, NY, USA, 2016. Association for Computing Machinery.

[20] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2020)*, pages 805–825, Berkeley, CA, USA, Nov. 2020. USENIX Association.

[21] H. Qiu, S. Jha, S. S. Banerjee, A. Patke, C. Wang, F. Hubertus, Z. T. Kalbarczyk, and R. K. Iyer. Is function-as-a-service a good fit for latency-critical services? In *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, page 1–8, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450391726. doi: 10.1145/3493651.3493666.

[22] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer. SIMPPO: A scalable and incremental online learning framework for serverless resource management. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC 2022)*, pages 306–322, New York, NY, USA, 2022. Association for Computing Machinery.

[23] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer. Reinforcement learning for resource management in multi-tenant serverless platforms. In *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, EuroMLSys '22, page 20–28, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392549.

[24] H. Qiu, W. Mao, C. Wang, H. Franke, A. Youssef, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer. AWARE: Automate workload autoscaling with reinforcement learning in production cloud systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 2023)*, pages 387–402, 2023.

[25] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, et al. Carbon-aware computing for datacenters. *IEEE Transactions on Power Systems*, 38(2):1270–1280, 2022.

[26] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: Coordinated multi-level power management for the data center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2008)*, page 48–59, New York, NY, USA, 2008. Association for Computing Machinery.

[27] F. Rossi, M. Nardelli, and V. Cardellini. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *Proceedings of the 12th International Conference on Cloud Computing (CLOUD 2019)*, pages 329–338, 2019.

[28] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, et al. Autopilot: Workload autoscaling at Google. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys 2020)*, pages 1–16, 2020.

[29] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 2020)*, pages 205–218, 2020.

[30] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi. ENSURE: Efficient scheduling and autonomous resource management in serverless environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS 2020)*, pages 1–10, 2020. doi: 10.1109/ACSOS49614.2020.00020.

[31] W. Tang, Y. Ke, S. Fu, H. Jiang, J. Wu, Q. Peng, and F. Gao. Demeter: QoS-aware cpu scheduling to reduce power consumption of multiple black-box workloads. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC 2022)*, page 31–46, New York, NY, USA, 2022. Association for Computing Machinery.

[32] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. Peeking behind the curtains of serverless platforms. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (ATC 2018)*, page 133–145, USA, 2018. USENIX Association. ISBN 9781931971447.

[33] Z. Wang, S. Zhu, J. Li, W. Jiang, K. K. Ramakrishnan, Y. Zheng, M. Yan, X. Zhang, and A. X. Liu. DeepScaling: Microservices autoscaling for stable cpu utilization in large scale cloud systems. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC 2022)*, pages 16–30, New York, NY, USA, 2022. Association for Computing Machinery.

[34] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *2012 41st International Conference on Parallel Processing Workshops*, pages 262–268, 2012. doi: 10.1109/ICPPW.2012.39.

[35] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song. Dynamo: Facebook's data center-wide power management system. *ACM SIGARCH Computer Architecture News*, 44(3):469–480, 2016.

[36] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz. Selecting the best VM across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC 2017)*, page 452–465, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350280. doi: 10.1145/3127479.3131614. URL https://doi.org/10.1145/3127479.3131614.

[37] Z. Yang, P. Nguyen, H. Jin, and K. Nahrstedt. MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows. In *IEEE 39th International Conference on Distributed Computing Systems (ICDCS 2019)*, pages 122–132, Washington, DC, USA, 2019. IEEE Computer Society.

[38] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou. Sinan: ML-based and QoS-aware resource management for cloud microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2021)*, page 167–181, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446693. URL https://doi.org/10.1145/3445814.3446693.