SmartOClock: Workload- and Risk-Aware Overclocking in the Cloud

Jovan Stojkovic, Pulkit Misra, Íñigo Goiri, Sam Whitlock, Esha Choukse,
Mayukh Das, Chetan Bansal, Jason Lee, Zoey Sun, Haoran Qiu,
Reed Zimmermann, Savyasachi Samal, Brijesh Warrier, Ashish Raniwala, Ricardo Bianchini

Microsoft

***Abstract*—Operating server components beyond their voltage and power design limits (*i.e.*, overclocking) enables improving performance and lowering cost for cloud workloads. However, overclocking can significantly degrade component lifetime, increase power consumption, and cause power capping events, eventually diminishing the performance benefits.**

In this paper, we characterize the impact of overclocking on cloud workloads by studying their profiles from production deployments. Based on the characterization insights, we propose SmartOClock, the first distributed overclocking management platform specifically designed for cloud environments. SmartOClock is a workload-aware scheme that relies on power predictions to heterogeneously distribute the power budgets across its servers based on their needs and then enforce budget compliance locally, per-server, in a decentralized manner.

SmartOClock reduces the tail latency by 9%, application cost by 30% and total energy consumption by 10% for latency-sensitive microservices on a 36-server deployment. Simulation analysis using production traces show that SmartOClock reduces the number of power capping events by up to 95% while increasing the overclocking success rate by up to 62%. We also describe lessons from building a first-of-its-kind overclockable cluster at a cloud provider for production experiments.

I. Introduction

Motivation. Cloud services provision resources to meet their peak performance requirements [21], [25], [39], [62], [81]. For example, many services need to keep their high-percentile latency (*e.g.*, P99) below a predetermined Service-Level Objective (SLO) [24]. These services incur high operating costs to reserve enough resources for handling infrequent load spikes and leave a substantial portion underutilized or even idle for the majority of time when their load is below its peak.

As an example, Figure 1 illustrates the aggregate load pattern on a typical weekday of three services that are part of a communication and collaboration workload operated by a first-party customer of a major cloud provider. Collectively, these services use ∼1M virtual cores (across regions) to handle peaks that last for a few hours per day - between 10 am to noon for *Service A* and 5 minutes at the top and bottom of the hour for the other two services.

Emerging cloud paradigms, such as autoscaling [6], [35], [72] and serverless computing [4], [34], [45], [70], can be used to dynamically remove and add Virtual Machine (VM) instances for managing cost. However, these solutions (1) can

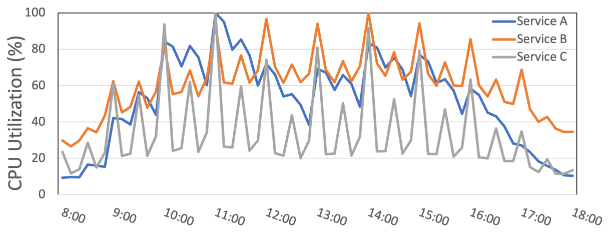

Fig. 1: Load pattern on a typical weekday in one region. Utilization normalized to the peak of each service.

increase the application's tail latency as booting up a new VM can take up to a few minutes [1], and (2) cannot be easily applied for stateful services [46], [69]. Hence, many applications still statically provision for infrequent load spikes.

On the other hand, recent advances in processing and datacenter cooling technologies have enabled component (*e.g.*, CPU, GPU) overclocking, *i.e.*, operation beyond typical voltage and power design limits [51]. Overclocking boosts a workload's performance and, thus, provides an opportunity to handle transient load spikes in a cost-efficient manner. For example, overclocking the CPU during a service's peak time can keep the tail latency below the required SLO and save cost by reducing provisioned resources.

However, overclocking is not free. If used naively, it increases power consumption and can cause frequent power capping events that diminish the performance benefits. Worse, overclocking impacts component lifetime by increasing wear-out and, thus, cannot be used indefinitely. The limited amount of overclocking needs to be used smartly as it may not benefit all workloads at all times: (1) overclocking the CPU of a memory-bound workload, or (2) overclocking a workload while experiencing a low load, will not provide much benefit.

Finally, providers need to protect workload SLOs when overclocking is unavailable. For example, a workload might have under-provisioned due to reliance on overclocking, but it would miss its SLOs under peak load if its VMs cannot be overclocked. Therefore, providers must use overclocking carefully while managing the associated risks.

Our work. For efficient use of overclocking in the cloud, we analyze cloud workloads and production traces, including the three services from Figure 1. We observe the following. First, overclocking improves the performance of popular cloud applications. However, a workload-agnostic overclocking scheme is suboptimal and often leads to missed SLOs or wasted overclocking cycles. Second, power and lifetime headroom exists to overclock most of the times without triggering power capping or compromising on reliability. Third, resource utilization history can be used to predict the availability of power and lifetime impact from overclocking. Fourth, servers' power draw within a power delivery unit (*e.g.*, a rack) is diverse, but the limit is still evenly distributed which disproportionately hurts the performance of power-hungry servers during a capping event. However, predictability in power consumption enables assigning heterogeneous power limits. Finally, a decentralized approach for local power draw enforcement during overclocking enables servers to find an efficient limit in case of initial assignment mispredictions.

We use the characterization insights to design SmartOClock, the first distributed overclocking management platform for the cloud. It enables a wide variety of cloud workloads to run with high performance at a lower cost. SmartOClock achieves its goals by introducing four novel design principles.

First, SmartOClock uses bidirectional communication between the application and the overclocking system to maximize an application's benefits from overclocking. Applications can use metrics (*e.g.*, latency, CPU utilization) or schedule-based policies to trigger overclocking, and the decisions can be made based on instance- and deployment-level monitoring. Second, it uses *admission control* to reserve power (from the available headroom) and overclocking budget for workloads. This step provides a predictable overclocking experience for workloads because SmartOClock can take corrective actions, such as scale-out, if it is unable to honor a reservation (*e.g.*, due to a change in available power for overclocking). Third, it leverages the predictability in power draw for assigning *heterogeneous* power budgets to servers. Heterogeneous assignments provide better performance while overclocking for power-hungry servers, without compromising on power safety. Finally, SmartOClock makes *decentralized* overclocking decisions for improved fault tolerance. Each server can make local decisions for granting overclocking requests based on assigned power and overclocking budgets. A server can also perform explorations to revise inefficient assignments (*e.g.*, due to incorrect or stale predictions).

We evaluate SmartOClock on a real server cluster and through simulations by using production traces. The cluster evaluation is performed on 36 overclockable servers (across 2-racks) running latency-sensitive microservices as candidates for overclocking and throughput-optimized power hungry machine learning (ML) training workloads, which are not overclocked. Our results show that SmartOClock reduces the P99 tail latency by 8.9% and application cost by 30.4% for latency-sensitive microservices, and total cluster energy consumption by 10% over state-of-the-art autoscaling solution. To validate our findings at scale, we use traces from hundreds of production racks and simulate SmartOClock. When compared to all practical policies, SmartOClock reduces the number of power capping events by up to 94.7% while increasing the overclocking success rate by up to 61.8%. We have also created a 2-rack overclockable cluster for production experimentation and share some lessons in Section VI.

Related work. There is a rich body of work on using CPU turbo-boost [16], [18], [30], [55], [74], [79], [100] and datacenter power management [37], [57], [59], [80], [84], [93], [96]. However, overclocking introduces unique challenges not addressed by prior work. First, CPU vendors design turbo to meet a cloud provider's performance and lifetime requirements. Cloud CPUs operate in performance mode [7], [36], [71], which makes cores always run at the highest turbo frequency within constraints (*e.g.*, power, thermal) [18], [82]. CPU vendors do not specify any timing limitations nor advise software-level wear leveling under their warranty terms for turbo [8], [47], and non-judicious use of turbo does not impact CPU reliability based on a recent study [76]. Generally, CPU failure is amongst the lowest types of failure in datacenter servers [65], [90]. Consequently, a provider, currently, does not need to manage reliability. Second, the oversubscription policies factor the higher power demand from turbo. Although this approach increases the total cost of ownership (TCO), it is necessary to meet the performance Service-Level Agreements (SLAs) with the customers [7], [36], [71]. In contrast, overclocking (beyond turbo) further improves performance but has a reliability impact not covered at design time by CPU vendors. Furthermore, a provider does not provision power for overclocking since turbo is sufficient to meet its performance SLAs. Therefore, overclocking is completely opportunistic - a provider needs to manage the power and reliability impact, while protecting workload SLOs when overclocking is unavailable; a problem setting not explored by prior work.

Summary. We make the following main contributions:

- We characterize the opportunities and challenges of overclocking cloud workloads, including the impact on power and component lifetime.
- We propose SmartOClock, a distributed overclocking management platform specifically designed for the cloud.
- We evaluate SmartOClock in a real system running latency-critical workloads, and using large-scale production traces.
- We share lessons from overclocking production workloads.

II. Background

Power management in cloud datacenters. The power delivery system in a cloud datacenter is organized in a hierarchy [57], [84], [93], [96]; the power budget of each parent node is split equally among its children. As providers oversubscribe power to improve datacenter utilization, the sum of the peak power draw of children nodes can exceed the budget of the parent (*e.g.*, servers in a rack) [57], [84], [93]. Under normal operation, child nodes can draw more than their even share if the cumulative power is below the limit of the parent. When it exceeds a threshold, power capping mechanisms (*e.g.*, Intel RAPL [22], prioritized capping [57], [59]) are used for safety. These mechanisms hurt performance as they reduce CPU frequency and can even throttle memory to restrict server power draw. To meet their performance SLAs, providers carefully oversubscribe to minimize/avoid capping events.

Component overclocking. Prior work shows the feasibility of overclocking in the cloud [51]. Overclocking operates components (*e.g.*, CPUs, GPUs) beyond their specifications to get frequencies even higher than turbo [10], [50].

A large fraction of cloud workloads, such as search or video conferencing [21], [88], are user-facing applications with transient load spikes. These workloads collectively consume millions of virtual cores to handle peak load. For the communication and collaboration workload at the provider, although chat and conference calls occur throughout the day, the peak (maximum simultaneous calls) that governs resource provisioning lasts for a few hours each day (Figure 1). Overclocking can be used during these peaks to save costs. However, a provider needs to manage the risks from overclocking. For example, for reliability management, the peak duration needs to be within the daily overclocking budget (*e.g.*, 10% per day) that satisfies a provider's server lifetime goals. Overclocking impacts reliability [3] due to three main reasons: (1) gate oxide breakdown, (2) electro-migration, and (3) thermal cycling. These processes are time-dependent and accelerate the lifetime reduction. Prior work has showed that there is an exponential relationship between temperature, voltage, and component lifetime [27], [51], [66], [92], [95].

III. Challenges and Opportunities

A successful overclocking management scheme needs to (1) satisfy application overclocking requirements, (2) use the available power efficiently, and (3) manage the impact of overclocking on component lifetime. To design such a scheme, we answer the following questions.

Q1: When do workloads benefit from overclocking? To efficiently use overclocking, a cloud platform needs to understand the applications' needs and behavior. Treating VMs as opaque and using workload performance proxies (*e.g.*, instructions per cycle (IPC) or CPU utilization) for overclocking decision-making can be suboptimal as the relationship between proxies and target performance metric is not always clear. Without knowing a workload's performance goals, the platform may overclock prematurely (*i.e.*, when the load is low and the tail performance is not impacted) and, due to the lifetime impact, lose the ability to overclock later when the tail is impacted. Combining IPC sensitivity with CPU utilization as a proxy for load can be inefficient too because the performance of some workloads can be impacted at a moderate CPU utilization while others may show no impact even under high utilization. Finally, operators can even have *deployment-level* goals for resource provisioning (number of VMs) and overclocking based on instance-level monitoring only will be inefficient.

To illustrate these scenarios, we profile two classes of popular cloud workloads: (1) microservices from the largest open-source benchmark suite, DeathStarBench [32], and (2) a proprietary web conferencing application called WebConf.

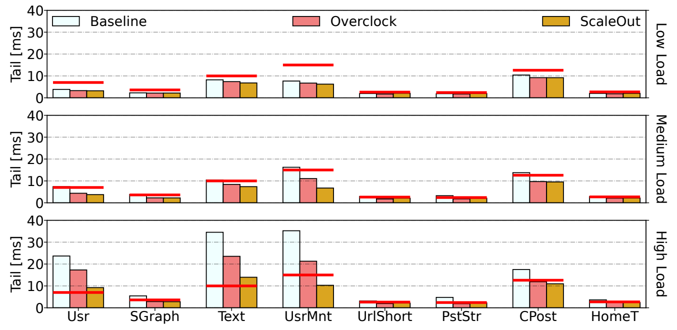

Fig. 2: Tail latency of SocialNet microservices with different loads in Baseline, Overclock, and ScaleOut environments.

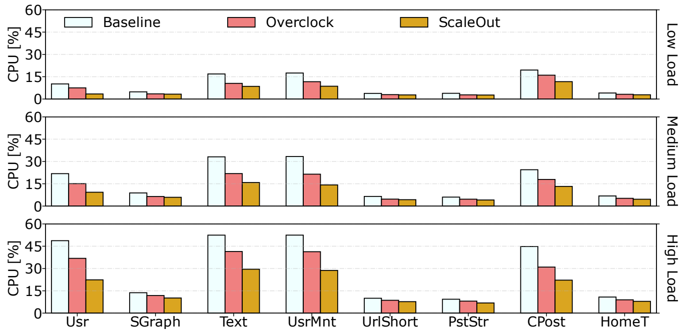

Fig. 3: CPU utilization of SocialNet microservices with different loads in Baseline, Overclock, and ScaleOut environments.

Microservices. We run eight SocialNet microservices [32] under varying loads (low, medium, and high) in three environments: *Baseline*, *Overclock*, and *ScaleOut*. *Baseline* and *Overclock* run a single VM at turbo (3.3 GHz) and overclocked (4.0 GHz) frequency. *ScaleOut* has two VMs running at turbo. Figure 2 shows the tail latency of the microservices. The red horizontal line indicates SLO, where the SLO for each service is set to be 5 times its execution time on an unloaded system [26], [60], [73]. Figure 3 shows their CPU utilization.

ScaleOut is provisioned to handle the peak load, like many services at the provider, and always operates 2 VMs that run at turbo. It achieves the best performance, but has the highest cost. On the other hand, Overclock works with a single server and can still keep the tail latency below the SLO in many cases, thereby avoiding the need to scale out. However, some services (*e.g.*, Usr) can tolerate higher CPU utilization without violating their SLO while other services (*e.g.*, UrlShort) violate their SLO even under low CPU utilization. Therefore, a workload-agnostic policy that uses CPU utilization for overclocking will make suboptimal decisions. These observations hold for any cloud workload with similar characteristics: bursty load with tail latency as the key metric. For example, ML inference servers [60], [97], serverless computing [86], and key-value stores [61] amongst others.

WebConf. The application hosts conferences in a VM. For fault-tolerance, operators provision VMs across availability zones (AZ) in a region. In an AZ, provisioning keeps the average *deployment-level* CPU utilization below 50% to handle

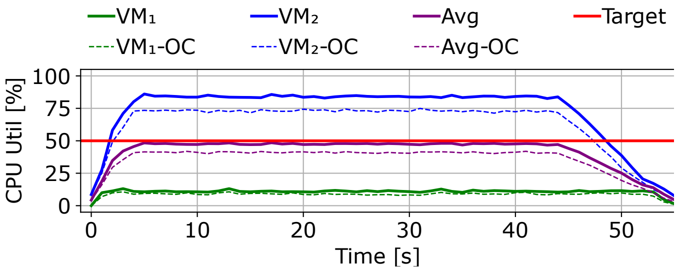

Fig. 4: CPU utilization timeline with and without overclocking for two WebConf VMs.

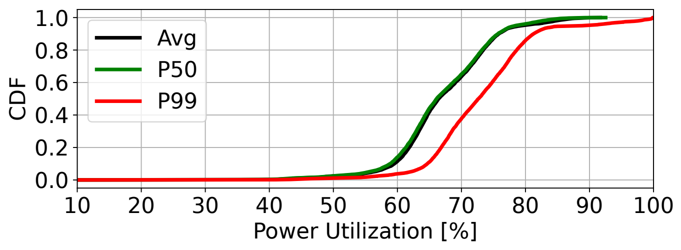

Fig. 5: Average, median (P50), and peak (P99) power utilization of 7,100 racks over 6 weeks in three regions.

load from another failed AZ. Overclocking can save cost for WebConf through deployment-level decisions. Individual VMs can have high utilization, but overclocking them is suboptimal since the deployment-level utilization may be below the target.

To illustrate, we execute WebConf on two VMs. VM_1 has low load (10% CPU utilization), while VM_2 has high (80%). Figure 4 shows VM and deployment-level average CPU utilization. Overclocking provides benefit, but is unnecessary since the baseline already meets the application-level goal.

Q2: Are there enough resources for overclocking? Since overclocking increases power consumption and component wear out, we need headroom for these resources.

Power headroom. We analyze the power consumption of a major first-party customer of the provider running distinct power-hungry services (>100), including those from Figure 1, across 7.1k dedicated racks. The racks span all major regions (*e.g.*, United States, Europe, Asia) and each rack has 24-32 servers. The analysis period is 6 weeks (April 10^{th} – May 12^{th}, 2023). Figure 5 shows the CDF of average, median (P50), and P99 rack power utilization. Half the racks have an average utilization lower than 66%. Importantly, 50% and 90% of the racks have P99 lower than 73% and 89%, respectively. We observe similar power patterns on non-dedicated racks at the provider with a mix of first- and third-party workloads.

However, naively overclocking the racks' servers can cause power capping events. To estimate the power impact from overclocking, we use the overclocking requirements of critical user-facing workloads which represent 45% of deployed cores by the first-party customer. The requirements vary – some require overclocking for several minutes per hour, while others for multiple contiguous hours during a weekday. Figure 6 shows the power consumption of a rack with and without overclocking for the week of April 24^{th}, 2023; the red line

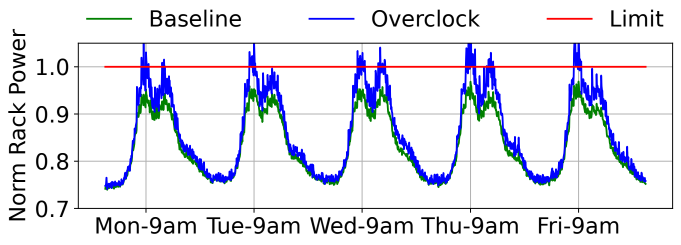

Fig. 6: Example of rack power consumption over 5 weekdays

shows the rack power limit. Each server hosts VMs of many distinct services and captures a typical datacenter environment with multi-tenant servers. The rack power draw is below the limit in the baseline case, but overclocking exceeds the limit and causes capping. More generally, overclocking the selected workloads will result in no power capping events for 85% of the time. For the remaining 15%, naive overclocking results in power capping events that cause 30-50% degradation in performance (core frequency) for workloads. There is still headroom available on these power-constrained racks, but it is insufficient to overclock to the highest frequency; the available headroom is 75% of the requisite at the 99^{th} percentile.

Therefore, most of the time (85%) racks have the needed power headroom to support overclocking. However, a power-aware policy is needed for constrained scenarios.

These findings are with the default VM scheduler that uses a set of resource-centric rules for placement [40], [89]. Providers can add power-aware scheduling policies to aid overclocking, but this exploration is future work. Nonetheless, even with optimized placement, there will still be power-constrained scenarios where overclocking has to be performed carefully.

Component lifetime headroom. Prior work shows that advanced cooling (*e.g.*, wax, immersion) is needed for enabling sprinting/overclocking [30], [78], [79] and not degrading expected lifetime [51]. However, there is opportunity to overclock even in air-cooled server deployments. Cloud server cooling is designed for operating components at their rated thermal design power (TDP). However, servers rarely consume their TDP due to low resource utilization in the cloud [21]. Several factors contribute to the low utilization. First, over-provisioning and diurnal workload patterns result in low VM utilization. Second, workload heterogeneity on servers results in low server utilization. Each server hosts many small VMs (2-8 cores). For resiliency, operators spread their VMs across servers and racks. Consequently, the VMs on any given server belong to different workloads. This heterogeneity results in low server utilization as the workloads have different peak times. Consequently, components are not thermally constrained for overclocking. However, advanced cooling can be used to enhance the capability (*e.g.*, duration) as lower operating temperatures reduce ageing [51]. Finally, since overclocking does not exceed TDP nor the rack limit, it will not cause additional cooling-related failures.

In fact, under-utilization enables overclocking in air. Vendors assume near-100% usage for determining frequencies/voltage (*e.g.*, turbo) that satisfy the lifetime goals. Under-

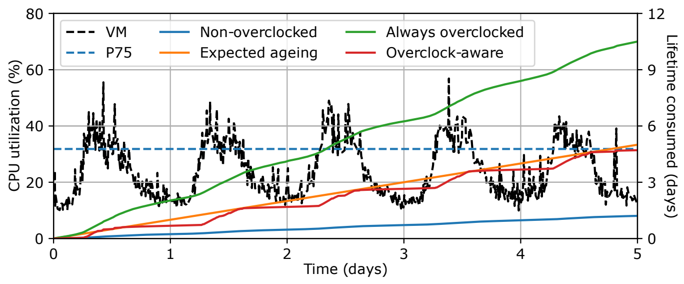

Fig. 7: CPU ageing for a VM running a workload with a diurnal pattern under multiple overclocking policies.

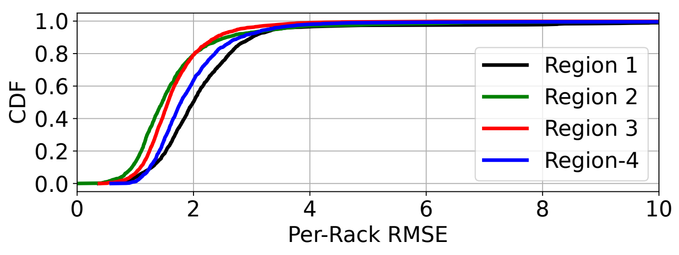

Fig. 8: CDF of RMSE with the power consumption patterns of our predictions across 7.1k racks in four regions.

utilization accumulates lifetime credits that can be consumed via overclocking. To understand the overclocking opportunity, we use a 7nm composite processor model from TSMC. The model faithfully represents CPU transistor logic and is used for design and compliance with reliability goals. It uses a complex relation between overclocking (voltage scaling), CPU utilization (time period for which cores run at the specified voltage), and aging from wear-out in the form of gate oxide failure [23], [58]. It predicts that a CPU ages by 2.5 years over a 5-year period for a conservative fleet usage. The remaining 2.5 years can be used for overclocking. But naively overclocking for 50% of the time ages the CPU by 5 years in less than a year use due to accelerated wearout. A smart system can constrain overclocking so that the part ages according to the reference (*i.e.*, 1 year ageing over a 1-year period).

Figure 7 illustrates the effect of overclocking policies. It shows the 5-day CPU utilization of a production workload with a diurnal pattern of daily midday peaks above 50% and valleys lower than 20% at night. The expectation is that the processor ages 5 days over the same period ("Expected ageing"). However, the actual ageing is less than 2-days for the "Non-overclocked" baseline. "Always overclock" ages the CPU over 10 days, indicating that, for the same CPU utilization, overclocking significantly increases wearout. On the other hand, an "Overclock-aware" policy can consume the accumulated credits by overclocking for 25% of the time and not exceed the expected ageing. Offline modeling assumes CPU utilization is unchanged while overclocking for worst-case analysis. However, overclocking's ageing impact will be less if the utilization reduces. To address this limitation, we are working with CPU vendors on "wearout counters" for online calculation of the ageing impact (see §VI).

Therefore, overclocking is enabled due to under-utilization and can be improved with advanced cooling. A system must carefully manage overclocking to comply with lifetime goals.

Q3: Can we predict the availability of resources? An efficient overclocking system must perform admission control based on available power and lifetime. We observe that a prediction-based approach can yield high accuracy.

Power predictability. A system needs to predict how much power can be used by overclocking without triggering capping. Figure 6 shows the baseline power consumption of a rack and gives us insights that historical observations of power profiles can be leveraged for prediction. The rack hosts multiple services, where each service can have a distinct power profile. However, due to statistical multiplexing, the combined power consumption of the rack with heterogeneous services shows a repeatable pattern. We analyzed the power predictability of 7.1K racks and thousands of servers in these racks that collectively run >100 services. Although the racks are dedicated for a first-party customer, this dataset accounts the fact that racks and servers on a public cloud host heterogeneous workloads. Furthermore, the dynamicity of cloud platforms (*e.g.*, VM churn according to a workload's needs) is also reflected.

Figure 8 shows the CDF of Root Mean Squared Error (RMSE) of rack power predictions in four regions of the provider. The RMSE is low even at high percentiles indicating high predictability across thousands of racks. For example, in Region 3, 50% and 99% of the racks have an RMSE lower than 1.95W and 5.11W, respectively. The findings are similar in the other regions. Furthermore, an analysis of 20K non-dedicated racks in the three most popular regions of the provider yielded similar results; these racks run a mix of first- and third-party workloads. A major reason for rack power predictability is long-lived VMs that govern resource utilization. Prior work shows that long-lived VMs (or jobs) account for >95% of allocated resources [21], [81], [85].

Component lifetime impact predictability. To remain within the overclocking lifetime budget, an overclocking management system needs to predict how much overclocked CPU cycles a given workload will consume. As a server's power draw depends on CPU utilization, predictability in power draw indicates predictability in CPU utilization. Using the aforementioned methodology for a rack's power, our analysis shows that power consumption and CPU utilization of servers are also predictable: more than 50% and 90% of the servers have an RMSE lower than 3.13W and 7.82W, respectively.

Therefore, historical observations of power draw and CPU utilization can be used to predict the available power and component lifetime headroom for overclocking.

Q4: How to assign power budgets? A server's power budget for "safe" overclocking depends on the power consumption of the other servers in the hierarchy (*e.g.*, a rack). Under fair share, the rack power budget is split equally across all servers and each server can locally ensure that its power draw stays below the limit to avoid capping while overclocking. However, this approach is inefficient since some servers may not be able to overclock even while the rack is not power-constrained.

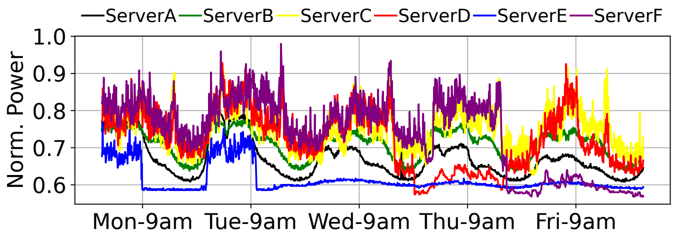

Fig. 9: Normalized power consumption over time of six randomly chosen servers within the same rack.

Figure 9 shows the normalized power consumption over time during the week of April 24th, 2023, for six randomly chosen servers in a rack. Each server is a different color. We can see that servers have very different power consumption profiles. Some servers may use even 30% less power than others. In addition, servers that consume the most power in a rack change over time. For example, at different timestamps, ServerC, ServerD, or ServerF may be the power dominant one.

Therefore, an efficient overclocking system needs to split the rack power budget *heterogeneously* across servers. Historical observations of server power demand and rack-level headroom can be used for the heterogeneous attribution.

Q5: How to efficiently use the power? The power headroom for overclocking in a rack is consumed by all servers in that rack. Thus, to grant or reject an overclocking request, each server should contact a centralized entity that has the global view of the rack's total remaining power headroom. Unfortunately, this approach is expensive and limits the system's fault-tolerance – if the centralized entity fails, then all overclocking requests would be rejected. Making local overclocking decisions using assigned server power budgets improves fault tolerance. However, overclocking requests may still be rejected due to inefficient assignments. For example, a scheme that uses power predictions for budget assignments can be suboptimal due to mispredictions.

Therefore, a high-performance and fault-tolerant overclocking system needs to be decentralized and should allow servers to explore beyond their potentially stale power limits.

IV. SmartOClock

Driven by the characterization insights, we propose *SmartOClock*: a distributed overclocking management platform for the cloud. It is readily integrated with existing platforms and enables a wide variety of workloads to run with high performance at lower cost. SmartOClock responds to the outlined questions for an efficient overclocking scheme through four novel features. First, it is *workload-intelligent* as it uses hints provided by workloads to extract the most benefits from overclocking. Second, SmartOClock performs *prediction-based admission control* of overclocking requests to avoid power capping and premature component wearout. Third, it uses predictions to split the rack power limit *heterogeneously* across servers. Finally, SmartOClock uses a *decentralized scheme* for budget enforcement while overclocking and allows controlled exploration to revise inefficient assignments.

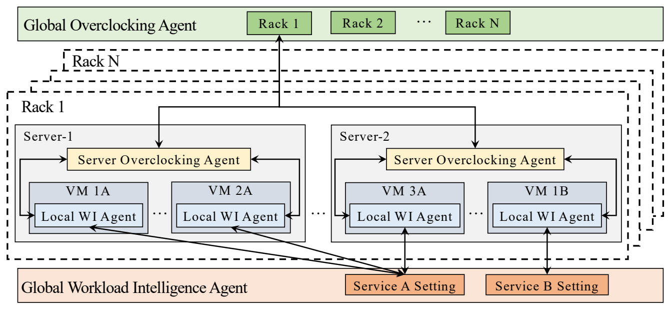

Fig. 10: Overview of the SmartOClock overclocking system.

Architecture. Figure 10 shows the architecture of SmartOClock. The system is organized hierarchically where each controller manages the components on its level and communicates with the controllers from the upper and lower levels. First, when deploying their services, the workload owners configure the *Global Workload Intelligence Agent* for their service. They specify the conditions under which the workload needs to be overclocked. As workloads are composed of one or more VMs, each VM is deployed with its own *Local Workload Intelligence Agent*. Like conventional auto-scaling, the local agent collects the metrics of interest from the VM and sends them to the service's global agent. Thus, this setup does not introduce new security or privacy challenges. The global agent decides if any VM needs to be overclocked using the metrics *aggregated at a service-level* and sends signals to the local agents of such VMs. On receiving a signal, a local agent sends an overclocking request to the *Server Overclocking Agent* (sOA). The request can be submitted via a local interface, such as a hypervisor-specific shared memory implementation [68], [83], [94] or locally-terminated network endpoint [5], [33], [67]. The sOA predicts if there are enough resources to satisfy the request. Based on the prediction, the sOA grants or rejects the request. If the request is rejected, the local agents inform the global agent which takes the corrective actions (*e.g.*, request scaling-out or redistribute the load towards the overclocked VMs). In the background, each sOA monitors the power and overclocking needs, and creates the server's profile to be periodically sent to the *Global Overclocking Agent* (gOA). The gOA uses the profiles to split its power budget into efficient per-server budgets. The sOA uses the assigned budget for the admission control until the budget gets updated.

A. Workload-Aware Overclocking

Overview. SmartOClock extends the existing autoscaling interface with overclocking. A workload specifies the scale-up (start) and scale-down (stop) thresholds for overclocking. The overclocking hints can be inserted by developers after profiling or they can be automated using the existing tools for automatic instance scaling [11], [31], [64], [87], [98]. Like conventional autoscaling, the overclocking thresholds can be: metrics-based or schedule-based. Under *metrics-based* overclocking, workloads can use application metrics (*e.g.*, tail latency, queue length) or resource utilization (*e.g.*, CPU, network) to trigger overclocking. The granularity of application hints can be per-function in the case of tail latency or per-VM in the case of resource utilization. These metrics can then be monitored per- and across-VM instances for specified time intervals to meet an application's goals. Additionally, workloads that have predictable times for high traffic (*e.g.*, 9-10 AM local time) can use *schedule-based* thresholds. Finally, workloads can also use a combination of metrics- and schedule-based. Importantly, extending the autoscaling interface for overclocking enables using scaling out (creating new VMs) as a fallback mechanisms for when overclocking is not possible. The scale-out signal can also be triggered proactively by SmartOClock using predictions for the ability to overclock (see Section IV-D).

Adopting WI by cloud users. Although workload owners already carefully tune the metrics and thresholds for horizontal scaling, there is overhead in repeating the process for vertical scaling (overclocking). To ease adoption, SmartOClock can be extended to infer the overclocking thresholds. It can leverage workload historical data to determine scale-up values. The lifetime impact of overclocking can be factored in this analysis. For example, use P90 of historical value if overclocking can be performed for 10% of the time only to comply with lifetime goals. The overclocking impact needs to be estimated to determine the scale-down value. An inaccurate estimate can either cause dithering if it is too close to the scale-up threshold or waste precious overclocking time if the estimate is too low. Performance models using low-level architectural counters can be used for the estimation. Workload owners can also leverage the inferred thresholds as an initial estimation.

B. Overclocking Admission Control

Overview. Naively granting all overclocking requests (1) increases the chance of power capping events deteriorating performance of all VMs, and (2) wears out the server's components causing premature server decommission. Instead, SmartOClock performs admission control for the overclocking requests based on *power and component lifetime predictions*. It (1) predicts the rack's power consumption and assesses if an overclocking request will result in power capping, and (2) predicts the CPU utilization of cores requesting overclocking and assesses if cores will exceed the allowed overclocking lifetime budget. Based on these predictions, SmartOClock decides (1) if the requested power and overclocking budgets can be reserved for overclocking a schedule-based workload, or (2) for how long a given VM with metrics-based overclocking can be overclocked before taking the corrective actions. Note that the power reservation is *soft*, the power can be taken by workloads outside of the system that do not need overclocking and SmartOClock needs to adjust.

Managing power. As observed in Section III, power draw of racks and servers in those racks is highly predictable. Hence, the Global and Server Overclocking Agents in SmartOClock continuously monitor the server and rack power consumption and use the data gathered during monitoring to periodically (*e.g.*, weekly) recompute the per-rack and per-server power templates. The templates are used to predict if the additional power of overclocking will trigger a power capping event.

SmartOClock creates a power template using *per-day aggregation* of power draws across all weekdays in the prior week. The template represents a single day and the same template is used for predictions for all days in the following week. For example, the template's value at 9AM is the median of rack's power consumption at 9AM across all five weekdays. A separate template is used for weekends. The intuition for this approach is that (1) using a coarse-grained measurement (*e.g.*, the maximum over a week) is too conservative (*i.e.*, it unnecessarily rejects many overclocking request) and (2) using fine-grained measurements (*i.e.*, all power measurements from the prior week) is insufficiently robust to outliers (*e.g.*, holidays during the prior week). Section V-B compares the accuracy of several template-creation strategies.

Managing lifetime impact from overclocking. A max time to overclock a component is obtained through an offline analysis with the vendors (*e.g.*, 10% over a 5-year period). This analysis uses realistic, yet conservative, utilization of cloud components to determine the opportunity. The duration of individual overclockings can vary, but SmartOClock needs to honor the *total* overclocking time assumption to comply with component lifetime goals. This requirement is the same as for using turbo-boost on non-overclockable CPUs.

To get uniform overclocking over a component's expected lifetime, SmartOClock divides the overall budget into epochs. The definition of an epoch is configurable (*e.g.*, a day, week). Using a longer epoch, such as a week, enables assigning unused budgets from the weekend to the weekdays. Hence, SmartOClock defines an epoch to be a week and calculates per-weekday max overclocking time.

Each sOA ensures that the overclocked time-in-state of a component (*e.g.*, per-core of a CPU) does not exceed limit. Tracking and enforcement is per-server; an sOA uses mechanisms like Intel PMT [48] for the time-in-state tracking and denies overclocking requests if the budget is exhausted. Due to hardware heterogeneity, vendor-specific APIs are needed for tracking, calling such APIs is already supported by operating systems (*e.g.*, Intel PMT [49] and AMD HSMP [9] on Linux), and enforcement is via standard interfaces (*e.g.*, CPPC [2] for CPU cores). For a predictable overclocking experience, an sOA reserves overclocking budgets for scheduled requests. Unused budgets can be used by unscheduled (metrics-based) overclocking and also carried over to the next epoch.

C. Heterogeneous Power Budgets

Overview. SmartOClock splits the rack power budget *heterogeneously* amongst servers. The sOAs collect their server's power draw and overclocking needs over time to create power and overclocking *templates*. The power template specifies a server's draw at a given timestamp. The overclock template specifies the number of cores that *requested* and were *granted* overclocking. The sOAs periodically (*e.g.*, weekly) exchange their templates with the gOA. The gOA combines power and overclocking templates of all sOAs and computes individual power budgets. It grants power credits to servers for periods when VMs are overclocked, per the reported template.

Power budget computation. The power budget computation happens in three phases. First, the gOA uses its power model to separate the server's power into the regular and overclock power; the number of cores from the server's overclocking template enable the gOA to discriminate the two portions. Second, the gOA assigns to each sOA the initial power budget that is equal to the server's regular power consumption. Finally, the gOA splits the remaining power headroom based on the overclocking requirements, *i.e.*, servers with more overclocked cores in the past get larger extra power budgets for the future.

For example, a rack has two servers (X and Y) and 1.3kW power limit. Typical power consumption without overclocking of Server-X and Server-Y at 9AM is 400W and 300W, respectively. Thus, the unused power is 600W. In addition, at 9AM, Server-X and Server-Y typically need to overclock 5 cores (extra 50W) and 10 cores (extra 100W), respectively. Based on this history, the gOA computes the power budgets for 9AM for the two servers: for Server-X $400W + \frac{50\times600}{50+100}W = 600W$, and for Server-Y $300W + \frac{100\times600}{50+100}W = 700W$.

D. Decentralized Budgets Enforcement

Overview. SmartOClock takes *decentralized* overclocking decisions. It allows servers to locally process overclocking requests from their VMs. An sOA uses the server's power profile to predict if overclocking will exceed the server's power budget. As the budget computations rely on predictions, they may become suboptimal. Thus, SmartOClock allows sOAs to explore beyond their initial assignments. Similarly, an sOA tracks the overclocking time budget of VMs and predicts if a VM will run out.Then, to avoid missed SLOs, the sOA informs the WI agent of the inability to overclock; the global WI agent can take corrective actions using the configured scale-out policies. Enabling local decisions is key for reactively handling activity bursts under metrics-based overclocking. The overclocking trigger by a WI agent is conveyed to the (local) sOA that can start/stop overclocking in order of a few milliseconds. Furthermore, if the assigned power budget is insufficient (*e.g.*, due to change in load), then the sOA can independently explore a higher budget to maximize overclocking.

Power budget enforcement. The gOA periodically sends the heterogeneously assigned power budgets to each sOA. Then, each sOA uses a prioritized feedback loop to control the power draw while overclocking. For example, scheduled overclocking VMs can be of higher priority compared to unscheduled (metrics-based) overclocking requests. In the feedback loop, an sOA changes the frequency of the overclocked VMs per priority in discrete steps (*e.g.*, 100 MHz). Based on the impact of the last frequency change on the power draw, the sOA (1) maintains the VMs at the current frequency (if $threshold \leq draw < limit$, where *threshold* = *limit* - *buffer*), (2) increases frequency by step size (if $draw < threshold$), or (3) reduces frequency by step size (if $draw > limit$). Prioritization enables overclocking the more important VMs to the maximum extent before less important VMs are overclocked.

Exploring beyond the local budgets. Due to occasional mispredictions, the initial power allotment may become suboptimal—some servers consume less than predicted while other servers are limited by their budget and cannot overclock VMs to the maximum extent. Thus, SmartOClock allows sOAs to explore beyond their allocated power budgets. Specifically, on constrained servers, the sOA tries to gradually exceed the limit through two phases: *exploration* and *exploitation*.

Exploration. The sOA *conditionally* increases budget by a step size (*e.g.*, 20W) that causes the feedback loop to start increasing the frequency of the overclocked VMs. If within a short timespan (*e.g.*, 30 seconds), the sOA does not receive any *warning* messages from the rack power capping system (run in the rack manager on each rack), then it further increases the budget. The sOA stops when all VMs are overclocked to the highest frequency or when it receives a warning message. The rack manager sends a warning message to all sOAs when the rack's power draw reaches a *warning threshold* (*e.g.*, 95% of the rack's power limit). An sOA ignores the message if it is not exploring. Otherwise, it reduces its budget by the step size and uses *exponential back-off* for the next exploration phase.

Exploitation. After establishing a safe power budget (*i.e.*, no warning messages), the sOA stops exploring and enters the *exploitation* phase. In this phase, the sOA uses the new power budget to grant the overclocking requests until either the *time to exploit* expires or upon receiving a *power capping event*. When the time to exploit expires, the sOA starts a new exploration phase if needed. On a power capping event, the sOA goes back to its initial power budget.

Similarly, an sOA can explore beyond the local per-core overclocking budget. If a VM requires overclocking for longer than its assigned cores can sustain, an sOA can still start overclocking a VM on those cores until their budget is exhausted. Then, the sOA explores if any other cores on a server have enough budget to support the VM's overclocking. In that case, the sOA reschedules the VM on those cores.

Managing resource exhaustion. When an overclocking request is rejected, the global WI agent takes corrective actions per an operator-chosen policy. A simple policy is to scale out while factoring the number of VMs that cannot be overclocked across a deployment (*e.g.*, create *x* new if *y* existing VMs cannot be overclocked). Figure 11 shows the operations performed by SmartOClock for managing power exhaustion. First, sOA predicts when it will run out of power for overclocking. For this check, it first predicts the extra power from overclocking a given VM (at a given core frequency and worst-case CPU utilization). Next, via the template, it finds the time when the predicted extra power exceeds the server's budget. It then sends a signal to the global WI agent if the time to exhaustion is within a configurable window (*e.g.*, 15 minutes). To minimize performance impact from lack of overclocking, the length of the window should be greater than the time to scale out, so that overclocking is still available for the time it takes to scale

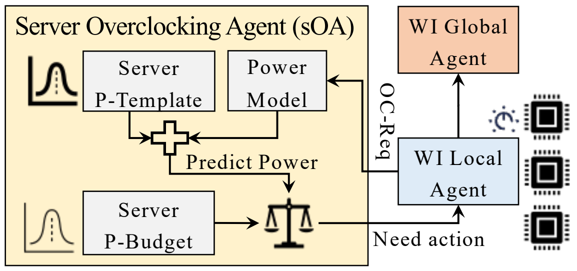

Fig. 11: Server Overclocking Agent in SmartOClock.

out. Finally, this operation can be performed ahead of time for scheduled overclocking requests to protect workload SLOs. For metrics-based overclocking, the scale-up (overclocking) threshold can be set before scale-out, where SLOs would be missed if resources are inadequately provisioned after the scale-out threshold is exceeded. Setting an earlier scale-up threshold allows using overclocking to handle load spikes and enables reverting to scale-out if overclocking is not possible. Like power, an sOA also predicts the time to exhaustion of the overclocking budget and informs the global WI agent.

V. Evaluation

To evaluate SmartOClock, we perform real-system experiments running cloud applications in an overclockable server cluster, and large-scale analysis using production-level traces.

A. *Cluster-Level Experiments*

Methodology. We implement SmartOClock and conduct the experiments on 36 overclockable servers (all 28 from one rack, and 8 from another during scale-out). Each server has an AMD CPU with 64 cores (128 threads). The CPU is configured to operate in performance mode [82]. Its max turbo and overclocking frequency are 3.3 and 4.0GHz. The active cores steadily run at these frequencies while TDP-unconstrained.

To set the load for each server, we take an example production rack that has the same hardware and servers per rack as our cluster. Based on the power traces of these production servers, we select which application to run in each individual server to mimic the *same* power profile. We run VMs hosting two open-source applications: (1) the latency-critical social network microservices (*SocialNet*) from DeathStarBench [32] and, (2) the throughput-optimized machine learning training (*MLTrain*) from FunctionBench [54]. In the power traces, 14 of the servers show constant high power while the other 14 show a diurnal pattern. For the first 14 servers, we use MLTrain and SocialNet for the other 14. The load for each benchmark instance is configured to mimic the power consumption of the corresponding production server.

We define the per-server load in our experiments based on the production traces. As the profiled production servers run mostly different, independent, workloads, each server in the experiments run independent set of SocialNet microservice instances. Thus, there is no correlation in the power consumption or loads across servers (*i.e.*, the peak load on one server does not affect the load on others). Auto-scaling is set for

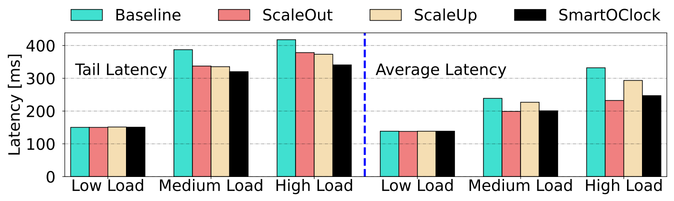

Fig. 12: P99 tail and average latencies of SocialNet services.

SocialNet based on its tail latency (initial count is 14). As in Section III, we set the SLO of each microservice to be 5 times its execution time on an unloaded system [26], [60], [73]. We will release and open-source the setup to deploy workloads in public cloud VMs.

We compare SmartOClock with a *Baseline* system that does not scale neither horizontally (number of instances) nor vertically (core's frequency), and *ScaleOut* and *ScaleUp* systems that only scale out/in and up/down, respectively, the number of SocialNet instances based on the observed tail. In the evaluation we use a metric-based overclocking policy, which is less predictable; experiments with the schedule-based policy show slightly better results due to better predictability.

Application performance. Figure 12 shows the P99 tail and average latency of SocialNet microservices in four environments. We group the 14 instances into three classes based on their load: Low, Medium, and High Load. Bars in the figure are the average across all instances with the same load level.

All systems perform equally well under low load. The impact on tail latency becomes prominent with increased load. In high load, SmartOClock reduces the tail latency of Baseline, ScaleOut, and ScaleUp by 19.0%, 10.5%, and 8.9%.

The average latency of SmartOClock is lower than Baseline and ScaleUp, but slightly higher than ScaleOut. The reason is that, to reduce the application's cost and prevent scaling out, SmartOClock operates for a longer time with higher latencies that are still below the SLO. However, SmartOClock significantly reduces the number of missed SLOs. The total number of missed SLOs at high load is reduced by 26×, 4.8×, and 2.3× over Baseline, ScaleOut, and ScaleUp, respectively. These results show that overclocking (via ScaleUp or SmartOClock) reduces missed SLOs compared to ScaleOut. However, overclocking alone is insufficient at higher loads as evidenced by the greater missed SLOs with ScaleUp, despite it overclocking for 5x longer. A combination of ScaleUp and ScaleOut via SmartOClock provides the best performance. Finally, SmartOClock reacts fast to sudden workload shifts and keeps application performance within its SLO: even on servers that triggered overclocking for more than 140 times within 5 minutes, SmartOClock did not miss any deadlines.

Cost. Performance improvements from SmartOClock result in cost savings for the users as they need to pay for fewer VMs. Figure 13 shows the average number of concurrently active VM instances for each environment over the entire run. Under high load, SmartOClock saves substantial cost by reducing the number of required instances by 30.4% over ScaleOut.

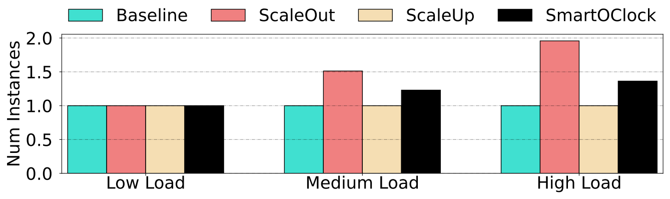

Fig. 13: Average number of SocialNet VMs varying load.

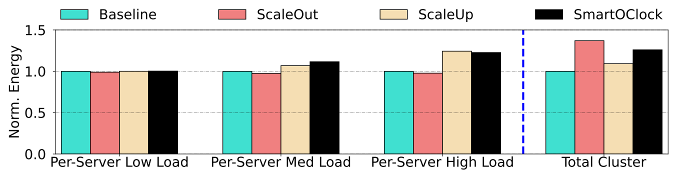

Fig. 14: Normalized per-single-server energy.

Energy consumption. Figure 14 shows normalized (1) per-single-server energy consumption under low, medium, and high load, and (2) total energy consumption of the system. Note that ScaleOut and SmartOClock are the only systems that meet SLOs. As the load increases, SmartOClock frequently overclocks cores, which increases the per-server energy consumption. However, as it uses fewer instances, the total energy consumption is reduced by 10% on average over ScaleOut. The savings are larger if we only consider servers running latency-critical microservices — 23% on average over ScaleOut.

Power-constrained environments. We evaluate SmartOClock's overclocking admission control and heterogeneous power budgeting under constraints. We reduce the rack's limit and measure the performance in two systems: NaiveOClock and SmartOClock. NaiveOClock grants all overclocking requests and on a power capping event splits the rack's budget equally among the servers. SmartOClock reduces the SocialNet tail latency by 6.7% and 8.4% for medium and high loads, respectively, and improves the MLTrain throughput by 10.4%.

Overclocking-constrained environments. To evaluate SmartOClock's proactive scale-out, we restrict the overclocking budget and measure the number of missed SLOs with and without proactive scaling. As we reduce the budget to 75%, 50%, and 25% of its initial value, reactive scale-out misses the SLO for 5.0%, 6.1%, and 7.2% of time, while SmartOClock's proactive approach eliminates all SLO violations.

B. Large-Scale Simulations

Methodology. We use production traces of the same first-party customer of the provider (see Section III) from multiple datacenters.Datacenters are composed of hundreds of racks and a few thousand servers with either Intel or AMD CPUs. Each workload's VMs are spread across servers and racks. The traces include rack and server power, and VM-level CPU utilization. All data is collected for 6 weeks (April 10^{th} - May 12^{th}, 2023), at a 5-minute granularity. Overclocking requirements (*e.g.*, time of day) are obtained from the operators.

We develop a discrete event simulator to evaluate SmartOClock. Models are used to estimate the power impact of overclocking; CPU utilization and core frequency are the input. We validate the model for each server generation.

TABLE I: Comparison of SmartOClock to different baselines.

System	Norm. # of Power Caps	Successful OClock Reqs	Penalty on Power Cap	Norm. Performance
High-Power Clusters				
Central	1.0	92%	21%	1.186
NaiveOClock	118.6	55%	34%	0.963
NoFeedback	5.5	72%	22%	1.122
NoWarning	27.4	81%	23%	1.081
SmartOClock	6.3	89%	22%	1.164
Medium-Power Clusters				
Central	1.0	96%	11%	1.195
NaiveOClock	36.6	79%	19%	1.022
NoFeedback	3.4	83%	11%	1.163
NoWarning	7.2	87%	12%	1.160
SmartOClock	3.9	93%	11%	1.185
Low-Power Clusters				
Central	1.0	99%	1%	1.208
NaiveOClock	14.0	99%	5%	1.172
NoFeedback	1.0	98%	1%	1.205
NoWarning	1.1	99%	2%	1.205
SmartOClock	1.0	99%	1%	1.208

We compare SmartOClock to *(1) Central* – an oracle with a global view of power draw that can precisely decide if an overclocking request will result in capping, *(2) NaiveOClock* – a system that grants all overclocking requests, *(3) NoFeedback* – a system that adheres to the per-server power budgets with no exploration beyond, and *(4) NoWarning* – a system that allows exploring but with no warnings. The servers go back to their initial power budget on a capping event.

Overclocking success and power capping. Table I shows the results: (1) number of power capping events in each system normalized to Central, (2) percentage of successful overclocking requests, (3) performance penalty of capping on non-overclocked VMs, and (4) normalized performance over Baseline. We define the performance penalty and improvement as reduction and increase in VM frequency compared to the Baseline (max turbo), respectively. Clusters are split into three groups based on power draw: *High*, *Medium*, and *Low-Power*.

First, naively granting overclocking requests causes many power capping events. NaiveOClock causes 118.6×, 36.6×, and 14.0× more events than Central in High, Medium, and Low-Power clusters, respectively. In contrast, SmartOClock lowers the events by 18.9× in High-Power clusters via prediction for admission control. Adding the warning messages efficiently controls overclocking beyond a server's budget: it reduces the number of events over NoWarning by up to 4.3×.

Second, SmartOClock successfully grants majority of overclocking requests. It is within 4%, 3%, and 1% of the success rate of an oracle Central system in High, Medium, and Low-Power clusters, respectively. The feedback-loop for exploring beyond the per-server budget is important: SmartOClock has up to 1.24× higher success rate than NoFeedback approach.

Finally, heterogeneous power distribution by SmartOClock reduces the performance penalty from power capping events. All systems bar NaiveOClock employ this optimization. The heterogeneous power budgets reduce the performance penalty due to power capping events over NaiveOClock by 1.62× and

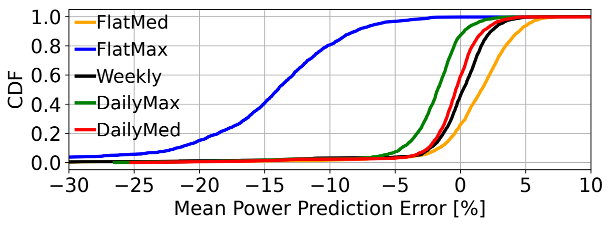

Fig. 15: CDF of mean power prediction for each technique.

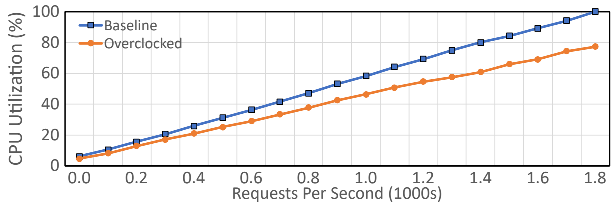

Fig. 16: Impact of overclocking Service B.

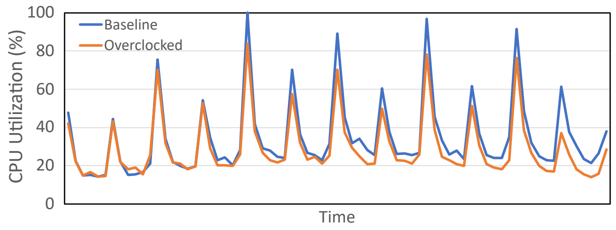

Fig. 17: Impact of overclocking Service C.

1.72× in High and Medium-Power clusters, respectively.

Power predictions accuracy. Figure 15 shows the CDF of prediction accuracy for computing the power templates. *FlatMed* and *FlatMax* use a constant prediction: a median or maximum of all prior measurements. FlatMed is opportunistic and underpredicts power, leading to high P99 prediction errors. Whereas, FlatMax is conservative and overpredicts power, resulting in negative prediction errors at low percentiles.

Weekly uses a time series of power measurements from the previous week for predictions in the following week. It is impacted by outliers since it treats each day separately: a few hours may behave differently due to the unexpected events. Thus, at high percentiles, its prediction error can be significant.

Finally, *DailyMed* and *DailyMax*, aggregate the power measurements across a week to represent a single, typical, day. The templates are time series of median or maximum values. DailyMed, used in SmartOClock, has the highest accuracy.

C. Experiments with Production Services

We evaluate overclocking *Service B* and *C* under production load. Each service consumes hundreds of virtual cores across tens of VMs. The deployment resource usage is similar to Figure 1 and the SLOs are consistent with each service's goals.

Figure 16 shows the average CPU utilization of *Service B*'s VMs for different request rates (bucketized by 0.1 due to live load variability). Overclocking reduces CPU utilization of VMs by 23% at a peak of 1.8k requests per second (RPS); the baseline operates at turbo (3.3 GHz). Alternatively, for the same CPU utilization, baseline can service 1.4k RPS vs 1.8k (28% higher) with overclocking. Figure 17 shows that overclocking reduces *Service C*'s 5-minute peaks over a weekday by 16%. The deployment load is similar on both days. Both results show the opportunity to down-provision while meeting the performance SLOs. Finally, overclocking enables servicing 25% additional load by *Service A* VMs under synthetic traffic; production experiments are being setup.

VI. Lessons from Production Deployment

We built a first-of-its-kind 2-rack (56 servers) overclockable cluster at a provider for CPU overclocking in production. Our deployment does not yet include cluster-wide coordination.

Motivation for building a cluster. Although CPU overclocking can provide substantial performance and cost benefits, a comprehensive analysis (*e.g.*, TCO reduction, revenue increase) is needed for new hardware features at scale. Projecting improvements is challenging due to workload-specific variations, as previous work shows [51]. Further, evaluating in a lab environment is not possible for even first-party workloads due to software dependencies (*e.g.*, deployment framework) and security concerns that prevent experimentation with production traffic. Thus, building an overclockable cluster was imperative.

Using experimental hardware in production datacenters. The provider has a rigid process for ensuring stability (*e.g.*, thermal limitations), reliability (*e.g.*, firmware errors), and performance for hardware deployment at scale that adds overhead for limited-scale experimentation. To address, we retrofitted by installing overclockable CPUs and firmware updates (*e.g.*, BIOS) on already-deployed servers. We also bypassed software checks that remove servers with unexpected configuration. A drawback is that the platform (*e.g.*, motherboard, cooling) is not optimized for overclocking, leading to thermal and max current throttling under heavy loads. Additionally, we provisioned adequate power to avoid capping; the limits are lowered for power management evaluations.

Experiments with first-party workloads. Since the cluster contains experimental hardware, we enforce strict admission control. However, this policy led to atypical workload placements that impact overclocking. Typically servers house VMs from various workloads due to dynamic cloud environments and scheduler efforts to optimize resource utilization [40], [89], but our policy caused VMs from the same workload to occupy entire servers. Although this impacts overclocking efficiency, it is useful for conservative benefit estimation.

Finer-grained overclocking. SmartOClock can overclock individual VMs but first-party operators want finer-grained overclocking (*e.g.*, containers in VMs). Although overclocking VMs still works, it is inefficient because of the higher power and reliability impact. Since containers are scheduled inside a guest VM without host visibility, we need guest participation for finer-grained overclocking. However, unsupervised control of frequency by guests can compromise reliability and power management. We are exploring a safe and efficient solution.

Hardware support for overclocking. Overclocking lifetime budgets can be improved with *wear-out counters* that indicate how a component's (*e.g.*, CPU core) lifetime is impacted by utilization (voltage) and operating temperatures. SmartOClock can use wearout counters to upgrade from a conservative offline model to a *per-part* online calculation for safety.

Furthermore, the prioritized feedback loop for managing power while overclocking can be offloaded to the hardware for efficiency. We are extending the ACPI [2] CPPC interface to configure VM priority while scheduling (no affinitization) on CPU cores.The firmware can use these priorities to assign per-core performance (frequency) while managing power.

Vendor engagements to enable overclocking. As overclocking is enabled by under-utilization (Section III), instead of overclocking, vendors (*e.g.*, Intel, AMD) inquire about designing a CPU with revised time-in-state assumptions for offline certification. However, this is still inefficient as it does not leverage the impact of utilization variability from workload demands (with and without overclocking) and temperature fluctuations on ageing at cloud scale. Using wear-out counters to track usage impact on ageing lacks these limitations.

Furthermore, we are working with vendors to ensure all cores can hit a minimum-desired overclocking frequency (*e.g.*, 15-20% beyond max turbo). Some cores can run faster, but this variability is not exposed on server CPUs (even for turbo); we are exploring bringing mechanisms from client CPUs (*e.g.*, ACPI CPPC preferred cores [2]) to leverage this variability.

Overclocking beyond CPUs. SmartOClock is a general framework and its principles can be easily applied for overclocking any server component. Our initial focus was CPU since it provides the highest benefits, but we have started exploring overclocking of other components (*e.g.*, GPU).

Silent data corruption (SDC). Recent work shows risks from SDC at scale [28], [41], [91]. Although overclocking can aggravate error rates, our extensive lab and production experiments do not show an increase in (un)correctable errors, with frequencies $\sim$20% beyond max turbo; this is inline with prior work [51]. Nonetheless, for safety, we work with vendors to define max overclocking frequency. Furthermore, techniques from the recent SDC work can be used for added safeguarding.

VII. Related Work

Computational sprinting. Extensive research [15], [16], [18], [30], [55], [63], [74], [78], [79], [99], [100] has explored computational sprinting (*i.e.*, boosting CPU frequency for short periods). Mechanisms like game theory [30], formal control [77], and performance modeling [74] have been proposed to manage sprinting. Researchers have also investigated efficiency factors like resource interference [63], power availability [16], processor design [38], and cooling [51]. However, none of these works holistically address the overclocking challenges in the cloud. They either focus on single-server setup, assume transparent-box knowledge of applications, or overlook multi-tenancy on a server or rack. The closest related work is Computational Sprinting Game (CSG) [30]. There are two major differences between CSG and SmartOClock. First, CSG leverages turbo and is constrained by thermal/power limits. In contrast, overclocking also affects reliability whose time scales are orders of magnitude (months/years) more than for power/thermal (minutes). It is nontrivial to add reliability under CSG when evaluating sprint utility. SmartOClockuses epochs to divide overclocking budget across coarse-grain time scales (days) that local agents enforce. Second, lack of sprinting/overclocking (of even a few VMs) can impact SLO of workloads that under-provision while relying on sprinting to handle peaks. Therefore, a mitigation mechanism to protect performance is needed when sprinting is unavailable, a problem not addressed by CSG. Section V presents the impact of proactive scaleout by SmartOClock to protect workload SLOs.

Undervolting. Prior work has proposed decreasing the voltage for a frequency below its safe marginal value for reducing power [12], [13], [17], [29], [52], [75]. However, undervolting can introduce instability and pipeline (*i.e.*, timing) errors, thereby necessitating hardware designers to add mechanisms for fault tolerance. For example, Razor [29] uses additional latches that run on a delayed clock in vulnerable paths to detect/recover from errors. This body of work is complementary and can create additional power and component lifetime headroom (reduced wearout from lower voltage) for overclocking.

Datacenter power management. Prior work has proposed oversubscription through leveraging statistical properties of concurrent power usage across servers [14], [37], [42], [56], [57], [59], [80], [84], [93] to improve datacenter power utilization and save costs. These works are complementary and influence our non-overclocked baseline. The provider leverages policies based on these prior works to oversubscribe power. The policies factor the power demand from turbo to meet the provider's performance SLAs [7], [36], [71] and prioritized throttling [57], [59] is used to protect (turbo) performance of critical workloads under rare power capping events.

Naively adding overclocking to the baseline power utilization increases the probability of power capping events. Increasing provisioned power cannot be used to address this problem due to the TCO impact, especially when turbo is sufficient to meet a provider's performance SLAs. Consequently, an overclocking system can only leverage unutilized power while meeting workload SLOs when overclocking is not possible; problems not addressed in prior power management work.

Workload intelligence. Research has leveraged workload awareness to optimize performance, energy consumption, and cost [19]–[21], [44], [53], [98], [101]. Sinan [98] uses ML models to allocate resources per microservice tier, minimizing cost while maintaining end-to-end tail latency targets. Re-Tail [19], Rubik [53], Adrenaline [43], and Gemini [101] use application-specific features to predict optimal per-request frequencies, reducing power consumption while meeting SLOs. Resource Central [21] gathers VM telemetry, learns VM behaviors offline, and provides online predictions for various resource managers. We propose a clean interface for cloud workloads to provide the necessary signals for overclocking without compromising their opaque-box implementations.

VIII. Conclusion

In this paper we proposed SmartOClock, the first distributed overclocking management platform for cloud environments. SmartOClock enables cloud workloads to run with high performance at a lower cost by introducing four novel features: workload-aware overclocking, prediction-based overclocking admission control, heterogeneous power budgeting and decentralized budget enforcement. SmartOClock reduces the tail latency by 8.9% and the application cost by 30.4%. Additionally, it minimizes the number of power capping events while increasing the overclocking success rate by up to 62%.

References

[1] S. I. Abrita, M. Sarker, F. Abrar, and M. A. Adnan, "Benchmarking vm startup time in the cloud," in *Benchmarking, Measuring, and Optimizing*, C. Zheng and J. Zhan, Eds., 2019.

[2] ACPI Specification Revision Committee, "Advanced configuration and power interface specification," 2022. [Online]. Available: https://uefi.org/specifications

[3] P. Alcorn, "CPU Overclocking Impact on Lifespan and Reliability," https://www.tomshardware.com/how-to/how-to-overclock-a-cpu#section-cpu-overclocking-impact-on-lifespan-and-reliability, 2023.

[4] Amazon AWS, "AWS Lambda," https://aws.amazon.com/lambda/.

[5] Amazon AWS, "Instance metadata and user data," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html.

[6] Amazon AWS, "AWS Auto Scaling," https://aws.amazon.com/autoscaling/, 2023.

[7] Amazon Web Services, "Processor state control for your EC2 instance," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html.

[8] AMD, "CPU warranty terms," https://www.amd.com/system/files/documents/processor-warranty-update.pdf.

[9] AMD, "Host System Management Port (HSMP)," https://github.com/amd/amd_hsmp.

[10] AMD, "Turbo Core Technology," https://www.amd.com/en/technologies/turbo-core.

[11] A. F. Baarzi and G. Kesidis, "SHOWAR: Right-Sizing And Efficient Scheduling of Microservices," in *SoCC*, 2021.

[12] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors," in *ISCA*, 2013.

[13] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage Noise in Multi-Core Processors: Empirical Characterization and Optimization Opportunities," in *MICRO*, 2014.

[14] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *IGCC*, 2012.

[15] H. Cai, Q. Cao, F. Sheng, Y. Yang, C. Xie, and L. Xiao, "ESprint: QoS-Aware Management for Effective Computational Sprinting in Data Centers," in *CCGRID*, 2019.

[16] H. Cai, X. Zhou, Q. Cao, H. Jiang, F. Sheng, X. Qi, J. Yao, C. Xie, L. Xiao, and L. Gu, "GreenSprint: Effective Computational Sprinting in Green Data Centers," in *IPDPS*, 2018.

[17] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," *Proceedings of the ACM on Measurements and Analysis of Computer Systems*, 2017.

[18] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova, "Evaluation of the Intel® Core™ i7 Turbo Boost feature," in *IISWC*, 2009.

[19] S. Chen, A. Jin, C. Delimitrou, and J. F. Martínez, "ReTail: Opting for Learning Simplicity to Enable QoS-Aware Power Management in the Cloud," in *HPCA*, 2022.

[20] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards Accurate Prediction for High-Dimensional and Highly-Variable Cloud Workloads with Deep Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, 2020.

[21] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in *SOSP*, 2017.

[22] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *ISLPED*, 2010.

[23] W. R. Davis, C. Shaw, and A. R. Hassan, "How to write a compact reliability model with the open model interface (omi)," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, 2020, pp. 1–2.

[24] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013.

[25] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *ASPLOS*, 2014.

[26] C. Delimitrou and C. Kozyrakis, "Amdahl's Law for Tail Latency," *Commun. ACM*, vol. 61, no. 8, jul 2018.

[27] D. DiMaria and J. Stathis, "Non-arrhenius temperature dependence of reliability in ultrathin silicon dioxide films," *Applied Physics Letters*, 1999.

[28] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent Data Corruptions at Scale," *CoRR*, vol. abs/2102.11245, 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[29] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *MICRO*, 2003.

[30] S. Fan, S. M. Zahedi, and B. C. Lee, "The Computational Sprinting Game," in *ASPLOS*, 2016.

[31] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ML-driven performance debugging in microservices," in *ASPLOS*, 2021.

[32] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *ASPLOS*, 2019.

[33] Google Cloud, "About VM metadata," https://cloud.google.com/compute/docs/metadata/overview.

[34] Google Cloud, "Google Cloud Functions," https://cloud.google.com/functions.

[35] Google Cloud, "Autoscaling groups of instances," https://cloud.google.com/compute/docs/autoscaler/, 2023.

[36] Google Compute Platform, "CPU platforms," https://cloud.google.com/compute/docs/cpu-platforms.

[37] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," 2009.

[38] B. Greskamp and J. Torrellas, "Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking," in *PACT*, 2007.

[39] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who Limits the Resource Efficiency of My Datacenter: An Analysis of Alibaba Datacenter Traces," in *IWQoS*, 2019.

[40] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM Allocation Service at Scale," in *OSDI*, 2020.

[41] P. H. Hochschild, P. J. Turner, J. C. Mogul, R. K. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *HotOS*, 2021.

[42] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang, "SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-Scale Datacenters," in *ASPLOS*, 2018.

[43] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *HPCA*, 2015.

[44] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters," in *SC*, 2021.

[45] IBM Cloud, "IBM Cloud Functions," https://cloud.ibm.com/functions/.

[46] IBM Cloud, ""Scaling stateful and stateless services"," https://www.ibm.com/docs/en/cloud-app-management/2019.3.0?topic=sizing-scaling-stateless-stateful-services.

[47] Intel, "CPU warranty terms," https://www.intel.com/content/dam/support/us/en/documents/processors/Limited_Warranty_8.5x11_for_Web_English.pdf.

[48] Intel, "Intel Platform Analysis Technology," https://www.intel.com/content/www/us/en/developer/topic-technology/platform-analysis-technology/overview.html.

[49] Intel, "Platform Monitoring Technology Telemetry (PMT)," https://github.com/intel/Intel-PMT.

[50] Intel, "What Is Intel® Turbo Boost Technology?" https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html.

[51] M. Jalili, I. Manousakis, I. Goiri, P. A. Misra, A. Raniwala, H. Alissa, B. Ramakrishnan, P. Tuma, C. Belady, M. Fontoura, and R. Bianchini, "Cost-Efficient Overclocking in Immersion-Cooled Datacenters," in *ISCA*, 2021.

[52] M. Kaliorakis, A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions," *IEEE Computer Architecture Letters*, 2018.

[53] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *MICRO*, 2015.

[54] J. Kim and K. Lee, "FunctionBench: A Suite of Workloads for Serverless Cloud Function Service," in *CLOUD*, 2019.

[55] S. Kondguli and M. Huang, "A Case for a More Effective, Power-Efficient Turbo Boosting," *ACM Transactions on Architecture and Code Optimization*, vol. 15, no. 1, 2018.

[56] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing, "Managing Distributed Ups Energy for Effective Power Capping in Data Centers," in *ISCA*, 2012.

[57] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, and R. Bianchini, "Prediction-Based Power Oversubscription in Cloud Platforms," in *USENIX ATC*, 2021.

[58] Y.-H. Lee, N. R. Mielke, W. McMahon, Y.-L. R. Lu, and S. Pae, "Thin-gate-oxide breakdown and cpu failure-rate estimation," *IEEE Transactions on Device and Materials Reliability*, vol. 7, no. 1, pp. 74–83, 2007.

[59] S. Li, X. Wang, X. Zhang, V. Kontorinis, S. Kodakara, D. Lo, and P. Ranganathan, "Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale," in *OSDI*, 2020.

[60] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez, and I. Stoica, "AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving," in *OSDI*, 2023.

[61] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-Memory Key-Value Storage," in *NSDI*, 2014.

[62] Q. Liu and Z. Yu, "The Elasticity and Plasticity in Semi-Containerized Co-Locating Cloud Workload: A View from Alibaba Trace," in *SoCC*, 2018.

[63] D. Lo and C. Kozyrakis, "Dynamic management of TurboMode in modern multi-core chips," in *HPCA*, 2014.

[64] S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu, "The power of prediction: microservice auto scaling via workload learning," in *SoCC*, 2022.

[65] J. Lyu, M. You, C. Irvene, M. Jung, T. Narmore, J. Shapiro, L. Marshall, S. Samal, I. Manousakis, L. Hsu, P. Subbarayalu, A. Raniwala, B. Warrier, R. Bianchini, B. Schroeder, and D. S. Berger, "Hyrax: Fail-in-Place server operation in cloud platforms," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 287–304. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/lyu

[66] D. Marcon, T. Kauerauf, F. Medjdoub, J. Das, M. Van Hove, P. Srivastava, K. Cheng, M. Leys, R. Mertens, S. Decoutere, G. Meneghesso, E. Zanoni, and G. Borghs, "A comprehensive reliability investigation of the voltage-, temperature- and device geometry-dependence of the gate degradation on state-of-the-art GaN-on-Si HEMTs," in *Proceedings of the 2010 International Electron Devices Meeting*, 2010.

[67] Microsoft Azure, "Azure Instance Metadata Service," https://learn.microsoft.com/en-us/azure/virtual-machines/instance-metadata-service.

[68] Microsoft Azure, "Data Exchange: Using key-value pairs to share information between the host and guest on Hyper-V," https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn798287(v=ws.11).

[69] Microsoft Azure, "Introduction to Auto Scaling," https://learn.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-resource-manager-autoscaling.

[70] Microsoft Azure, "Microsoft Azure Functions," https://azure.microsoft.com/en-gb/services/functions/.

[71] Microsoft Azure, "Virtual Machine series," https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series.

[72] Microsoft Azure, "Overview of autoscale in Azure," https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview, 2023.

[73] A. Mirhosseini and T. Wenisch, "μSteal: A Theory-Backed Framework for Preemptive Work and Resource Stealing in Mixed-Criticality Microservices," in *ICS*, 2021.

[74] N. Morris, C. Stewart, L. Chen, R. Birke, and J. Kelley, "Model-Driven Computational Sprinting," in *EuroSys*, 2018.

[75] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs," in *MICRO*, 2017.

[76] L. Piga, I. Narayanan, A. Sundarrajan, M. Skach, Q. Deng, M. C. B. Maity, A. Huang, A. Dhanotia, and P. Malani, "Expanding Datacenter Capacity with DVFS Boosting: A Safe and Scalable Deployment Experience," in *ASPLOS*, 2024.

[77] R. P. Pothukuchi, J. L. Greathouse, K. Rao, C. Erb, L. Piga, P. G. Voulgaris, and J. Torrellas, "Tangram: Integrated Control of Heterogeneous Computers," in *MICRO*, 2019.

[78] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, "Computational Sprinting on a Hardware/Software Testbed," in *ASPLOS*, 2013.

[79] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Computational Sprinting," in *HPCA*, 2012.

[80] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level Power Management for Dense Blade Servers," in *ISCA*, 2006.

[81] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in *SoCC*, 2012.

[82] H. Rui, "amd-pstate CPU Performance Scaling Driver," https://docs.kernel.org/admin-guide/pm/amd-pstate.html, 2023.

[83] R. Russell, "Virtio: Towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 95–103, jul 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400108

[84] V. Sakalkar, V. Kontorinis, D. Landhuis, S. Li, D. De Ronde, T. Blooming, A. Ramesh, J. Kennedy, C. Malone, J. Clidaras, and P. Ranganathan, "Data Center Power Oversubscription with a Medium Voltage Power Plane and Priority-Aware Capping," in *ASPLOS*, 2020.

[85] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in *EuroSys*, 2013.

[86] 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 *USENIX ATC*, 2020.

[87] J. Stojkovic, T. Xu, H. Franke, and J. Torrellas, "MXFaaS: Resource Sharing in Serverless Environments for Parallelism and Efficiency," in *ISCA*, 2023.

[88] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: The next Generation," in *EuroSys*, 2020.

[89] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-Scale Cluster Management at Google with Borg," in *EuroSys*, 2015.

[90] G. Wang, L. Zhang, and W. Xu, "What Can We Learn from Four Years of Data Center Hardware Failures?" in *DSN*, 2017.

[91] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding Silent Data Corruptions in a Large Production CPU Population," in *SOSP*, 2023.

[92] E. Wu, J. Sune, W. Lai, E. Nowak, J. McKenna, A. Vayshenker, and D. Harmon, "Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides," *Solid-State Electronics*, vol. 46, no. 11, 2002.

[93] 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," in *ISCA*, 2016.

[94] Xen Project, "XenStore," https://wiki.xenproject.org/wiki/XenStore.

[95] A. Yassine, H. Nariman, M. McBride, M. Uzer, and K. Olasupo, "Time dependent breakdown of ultrathin gate oxide," *IEEE Transactions on Electron Devices*, vol. 47, no. 7, 2000.

[96] C. Zhang, A. G. Kumbhare, I. Manousakis, D. Zhang, P. A. Misra, R. Assis, K. Woolcock, N. Mahalingam, B. Warrier, D. Gauthier, L. Kunnath, S. Solomon, O. Morales, M. Fontoura, and R. Bianchini, "Flex: High-Availability Datacenters with Zero Reserved Power," in *ISCA*, 2021.

[97] J. Zhang, S. Elnikety, S. Zarar, A. Gupta, and S. Garg, "Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems," in *HotCloud*, 2020.

[98] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices," in *ASPLOS*, 2021.

[99] W. Zheng and X. Wang, "Data Center Sprinting: Enabling Computational Sprinting at the Data Center Level," in *ICDCS*, 2015.

[100] W. Zheng, X. Wang, Y. Ma, C. Li, H. Lin, B. Yao, J. Zhang, and M. Guo, "SprintCon: Controllable and Efficient Computational Sprinting for Data Center Servers," in *IPDPS*, 2019.

[101] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, "Gemini: Learning to Manage CPU Power for Latency-Critical Search Engines," in *MICRO*, 2020.