PACS: Private and Adaptive Computational Sprinting

by

Jingyi Wu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Computational sprinting is a class of mechanisms that enables a chip to temporarily exceed its thermal limits to enhance performance. Previous approaches model the optimization of computational-sprinting strategies as a mean-field game and calculate the game's static equilibrium strategies using dynamic programming. This approach has three main limitations: (i) a requirement for a priori knowledge of all system parameters; (ii) inflexibility in equilibrium strategies, necessitating recalculation for any system parameter change; and (iii) the need for users to disclose precise characteristics of their applications without data privacy guarantees. To address these issues, we propose PACS, a private and adaptive mechanism that enables users to independently optimize their sprinting strategies. Our experiments in a simulated environment demonstrate that PACS achieves adaptability, ensures data privacy, and provides comparable performance to state-of-the-art methods. Specifically, PACS outperforms existing approaches for certain applications while incurring at most a 10% performance degradation for others, all without having prior knowledge of system parameters.

Acknowledgements

I extend my deepest gratitude to Professor Seyed Majid Zahedi for his exceptional mentorship and guidance throughout my master's journey. His steadfast support and profound expertise were pivotal to the fruition of my research endeavors. Professor Zahedi's mentorship not only shaped my academic journey but also inspired my personal and professional growth.

I am profoundly thankful to my parents for their unwavering love, encouragement, and belief in my capabilities. Their sacrifices and endless support have been the cornerstone of my achievements, providing me with the strength and motivation to pursue my aspirations.

To my best friend, Vic, whose camaraderie and support have been a constant source of comfort and motivation, I am immensely grateful. Your unwavering faith in me and your invaluable advice have been instrumental in my journey.

A special word of gratitude goes to Katherine, the treasure who has recently graced my life. Katherine, your love, support, and understanding have added a new dimension to my world, enriching my life in ways I had never imagined. Your presence has been a source of joy and inspiration, and I am deeply thankful for the love and happiness you bring into my life.

I also extend my heartfelt thanks to all members of the Multi-agent Systems Team. Your kindness, collaboration, and shared wisdom have created an enriching environment that has significantly contributed to my personal and academic growth.

In closing, I would like to express my sincere appreciation to everyone who has contributed to my journey. Your support, in its many forms, has been invaluable, and I am eternally grateful for the roles you have played in my life and studies.

Dedication

This thesis is dedicated to my parents, for their endless love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational sprinting involves operating a chip beyond its regular thermal constraints for a short time period to enhance performance. The sprinting duration can be extended by the right choice of heat sink material (i.e., phase change materials (PCMs)) [49, 55]. However, a chip in general cannot sprint uninterruptedly as the excess heat generated as a result of a sprint must be dissipated before the chip can sprint again. Computational sprinting was originally proposed for embedded and mobile devices [51, 50]. However, the concept has since been extended to datacenters for accelerating intermittent, computationally intensive workloads [56, 71, 18, 9, 8, 72].

The performance gains from a sprint depend primarily on the running workload [46, 45]. A CPU-intensive workload can be effectively accelerated by a sprint, while a sprint has little effect on the performance of a memory-intensive workload [30]. Therefore, it is important to optimize the timing of the sprints for maximizing performance gains. The power supply is crucial for data center performance. It directly impacts the ability to sustain high computational loads and operational efficiency. Constraints on power availability potentially limiting the performance of intensive workloads and the overall reliability of the data center. For a server with a dedicated power supply, this can be achieved by locally profiling the running workload and predicting its phases. However, in scenarios where multiple servers share the same power supply, the sprinting strategy on one server could indirectly impact sprinting decisions on others. This becomes particularly crucial in multi-tenant datacenters, where oversubscribed power supplies are common for optimizing performance and efficiency [6, 19]. In such power-oversubscribed datacenters, the cumulative power demand for uncoordinated simultaneous sprints could exceed the maximum capacity of the power supply, potentially causing the circuit breaker to trip [56]. Thus, the performance of datacenter is limited by both thermal dissipation and power supply.

In a recent work, Fan et al. [18] study sprinting strategies in scenarios where servers, controlled by self-interested users, share the same power supply. The authors approach the problem through a game-theoretic lens and formulate the sprinting dynamics as a "computational sprinting game." They then show that equilibrium strategies in this game are characterized by simple threshold policies. Specifically, users decide to sprint only when the expected performance gains from a sprint surpass a certain threshold. To implement this approach, the authors propose a sprinting architecture involving individual per-user agents and a centralized coordinator. Users profile their workload and report the profiled data to the coordinator. The coordinator, utilizing dynamic programming, determines static thresholds for users based on the aggregated workload profiles. Subsequently, the coordinator sends the calculated equilibrium thresholds back to the agents, who use the thresholds to make sprinting decisions based on predicted performance gains.

While the sprinting architecture proposed in [18] is elegant and efficient, it has three primary limitations. Firstly, the coordinator requires prior knowledge of all system parameters. This prerequisite implies that the coordinator must be aware of specific details about the system configuration beforehand. Secondly, the equilibrium strategies generated by the architecture are rigid. In the event of any changes in system parameters, the coordinator is compelled to recalculate new equilibrium strategies. This inflexibility hinders the system's ability to adapt dynamically. Lastly, users are obligated to disclose precise characteristics of their workload without receiving any assurances regarding data privacy. This lack of privacy guarantees might discourage users from fully sharing the details of their workloads.

To address these limitations, we propose PACS, a computational sprinting framework that leverages tools from multi-agent reinforcement learning and differential privacy to achieve adaptability and data privacy. Our contributions are outlined as follows.

- **Introduction of PACS.** In §3, we introduce our primary artifact, PACS, which is a private and adaptive computational sprinting framework. We provide an overview of its architecture, introducing a novel tax system designed to indirectly shape system-wide sprinting behavior. Additionally, we present a distributed algorithm based on model-free, multi-agent reinforcement learning to determine optimal sprinting strategies in a differentially private manner. We then provide a proof establishing that our proposed algorithm guarantees differential privacy.
- **Methodology and benchmarks.** In §4, we present three categories of benchmark applications: (a) synthetic applications based on Markov chains, (b) hybrid applications for deep-learning inference tasks rooted in queueing systems, and (3) real-world applications utilizing the Apache Spark framework [68].
- **Performance evaluation.** In §5, we evaluate our proposed framework by comparing it

with other baseline methods across synthetic, hybrid, and real-world benchmarks. Our experiments demonstrate PACS's ability to adapt to changes in system dynamics, while achieving a comparable performance to state-of-the-art methods, even outperforming them for certain applications. Additionally, we conduct a sensitivity analysis for PACS, illustrating the trade-off between data privacy and performance.

Chapter 2

Background

In this section, we first provide some background on computational sprinting. We then provide a brief discussion on the application of machine learning in systems. Finally, we present an overview of differential privacy.

2.1 Computational Sprinting

Computational sprinting is a class of mechanisms that allows a chip multiprocessor, a logic design architecture whereby multiple processing units (e.g., CPU cores) are integrated onto a single monolithic integrated circuit or onto multiple dies in a single package, to temporarily exceed its thermal limits to enhance performance. After a sprint, the excess thermal load must be removed before the chip can sprint again. This "cooling" period is required for the chip to prevent any thermal damage and maintain its long-term reliability. The maximum duration of a sprint and the minimum cooling duration depend on the choice of the heat sink material [50]. For instance, prior work has shown that the use of phase change materials (PCMs) can lead to extended sprinting durations [49, 55]

Originally proposed for mobile computing [51, 50], computational sprinting has been adopted for datacenter computing [56, 71, 9, 8, 72]. For datacenter servers with multicore processors, computational sprinting could be implemented by activating all cores and/or boosting the processor's voltage and frequency to levels that exceed the processor's sustainable thermal design power. For servers with GPU accelerators, computational sprinting can be implemented by GPU overclocking [57, 33].

The performance gain of an application from a sprint depends on the extent to which the application can utilize available resources [46, 45]. If the application is in a phase that can fully utilize resources, the sprint efficiency is high. Otherwise, a sprint increases power consumption with marginal returns. For instance, GPU overclocking can significantly improve the performance of DNN training and inference [61], whereas increasing frequency of CPU has little effect on the performance of a memory-bound application [30]. As a result, for an isolated server, maximizing sprinting efficiency requires application-specific strategies.

The "sprinting dynamics" are complex when servers are not isolated. For instance, in a multi-tenant environment, independently managed servers could share the same power source. [54] mentioned that, until 2020, power capping for low-risk power oversubscription has developed over the last 15 years into an essential enabler of data center cost reduction. In a power-oversubscribed datacenter, the cumulative power demand for uncoordinated simultaneous sprints could exceed the power supply capacity [56]. For such settings, Fan et al. [18] adopt a game-theoretic approach and propose a sprinting architecture to managing sprints based on the *(computational) sprinting game*.

2.1.1 Computational Sprinting Game

The sprinting game is a game model designed to capture system-wide sprinting dynamics. The game consists of N agents, each representing a server running an application. Time is divided into rounds. The duration of each round corresponds to the duration of a sprint. The utility that an agent derives from a sprint at any given round depends on the phase of the agent's application at that round. The objective of each agent is to maximize the long-term utility (i.e., the discounted sum of per-round utilities).

The game assigns a state to each agent. The state of an agent at any given round is one of three statuses: active, cooling, and recovery. States change according to agents' actions and system's parameters. An agent's state is active when their server operates in the normal power mode. An agent who is in the active state can decide to sprint by boosting the server's power beyond its thermal limits. Agents make this decision by comparing the utility gained from sprinting in the current round with the potential utility if they defer sprinting to a future round. The state of an agent who sprints at a round becomes cooling at the next round. The duration of the cooling status is contigent upon several factors including the chip's thermal conductance and resistence, the choice of heat sink material and cooling technology employed. This period represents the number of epochs an agent remains in the cooling state to dissipate excess heat, influenced by the environmental temperature.

When multiple agents sprint simultaneously, the total power demand could exceed the capacity, thus requiring backup power supply (e.g., UPS batteries), an event called a power emergency [22, 23]. Even with the backup power supply in place, the concern about circuit breaks still exists. It stems from the operational constrains and the potential impact on server availability and backup power supply battery longevity. When a circuit breaker trips due to an overload and resets, the power distribution shifts from a branch circuit to the backup power supply. Although lead acid batteriers in backup power supply systems can support discharge times ranging from 5 to 120 minutes, which is enough to complete a sprint, servers cannot sprint again until the backup power supply batteries be fully recharged. Additionally, frequent use of backup power supply batteries without proper recharge cycles can degrade battery life. It will affect future emergency support. Thus, managing the balance between sprinting demands and the backup power supply capabilities is crucial to maintain datacenter reliability. After a power emergency, the state of all agents changes to recovery. During a recovery period, the backup power supply recharges. The duration of each recovery period is a number of epochs, which depends on the rack's power supply and its battery capacity. Once a recovery period is over, the state of all agents changes to active.

Agents cannot sprint if their state is recovery or cooling; they can only sprint if their state is active. Sprinting greedily at every opportunity is intuitively a suboptimal strategy as it risks keeping agents in the cooling and the recovery states most of the rounds. This underscores the necessity for agents to weigh the advantages of sprinting against potential long-term losses when determining optimal strategies

Frequency of power emergencies depends on the collective sprinting behavior of all agents. Therefore, to optimize their strategies, agents need to track each other's strategies. However, this is not scalable, as the joint strategy space increases linearly with the number of agents. To address this issue, an effective approach is to utilize the framework of *mean-field* games [34, 10]. In a mean-field game, each agent reasons about all other agents in expectation, replacing them by an "average" agent. The solution concept in mean-field games is called the mean-field Nash equilibrium. In a mean-field Nash equilibrium, each agent best responds[1] to the strategy of the average agent. And the strategy of the average agent is the average of all agents' best-response strategies.

The mean field analysis of the sprinting game corresponds to characterizing the fraction of agents who sprint when the system is not in the recovery state. In an equilibrium, this fraction becomes stationary and converges to a constant. At every non-recovery round,

[1] A best-response strategy is a strategy that maximizes an agent's utility given a strategy of other agents.

some agents are active and some are in the cooling state. Out of the active agents, some opt to sprint. In an equilibrium, the fraction of sprinting agents across rounds remains unchanged in expectation.

The authors in [18] characterize the mean-field Nash equilibrium of the sprinting game using Bellman equations. They demonstrate that equilibrium strategies are simple threshold strategies, where agents in the active state sprint only when their expected utility from a sprint surpasses a threshold. To determine threshold strategies, the authors propose a sprinting architecture that features a centralized controller. Agents profile their applications and report their application's characteristics to the controller. Given the reported application profiles and system parameters, the controller efficiently solves the game's Bellman equations using dynamic programming.

Despite its elegance and efficiency, the proposed sprinting architecture encounters three primary limitations. Firstly, it requires the controller to possess prior knowledge of all system parameters, including cooling attributes across servers, application profiles, and recovery specifics. This requirement poses practical challenges, even within moderately complex systems housing hundreds of servers. Secondly, while the statically computed equilibrium strategies are provably optimal under unchanged system dynamics, any changes in system parameters mandate a complete recalibration of all strategies, undermining the system's adaptability. Lastly, the architecture mandates comprehensive transparency from all agents regarding their application characteristics. This involves agents profiling their applications and transmitting these profiles to the controller. This requirement raises valid privacy concerns, given the architecture's lack of privacy assurances, a critical issue in competitive multi-tenant environments.

To address the limitations of the prior work, we propose PACS, a private and adaptive mechanism for computational sprinting. Our approach utilizes tools and techniques from reinforcement learning and differential privacy. In the remaider of this section, we provide background on these two areas.

2.2 Machine Learning for Systems

Machine learning (ML) techniques have been shown instrumental in intelligent decision-making for computer systems, facilitating dynamic and adaptive optimization for tasks such as caching, computation offloading, power transmission, and resource allocation [47, 64, 40, 24]. ML techniques have been also used in large-scale operations, helping optimize energy consumption datacenters, enhance cooling efficiency, and reduce datacenter carbon

footprint [36, 17, 42, 41, 13, 4]. The use of ML for systems has further promoted sustainable datacenter operations, enabling them to respond effectively to varying energy demands and environmental conditions [48]. Among ML algorithms, reinforcement learning is one of the most prominent ones used for control problems.

2.2.1 Single-agent RL

In single-agent reinforcement learning (RL), an agent learns by interacting with an environment. Model-free RL enables the agent to determine the optimal policy without prior knowledge of system dynamics or payoff functions. In each round r, the agent observes a state s_r, takes action a_r, and receives a utility u_r. The state then transitions to s_{r+1}. The agent's objective is to learn a policy, π that maximizes the expected long-term utility:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{r=0}^{\infty} \gamma^r u_r \right],$$

where γ denotes the discount factor.

With deep reinforcement learning, the policy π is represented using a neural network. This results in a parametrized policy, represented as π_θ, where θ is the set of parameters (e.g., network's weights) being learned. By integrating the robust function approximation capabilities of deep learning with the objective-centric nature of reinforcement learning, systems can derive advanced strategies that are adaptable and resilient to changing dynamics.

Given parametrized policies, several approaches have been developed to optimize the learning process. Actor-critic methods are one of such approaches [60]. These methods often consist of two network: the actor network, which is used to select actions based on policy, and the critic network, which is used to evaluate these actions using a "value function". The actor adjusts the policy parameters based on critic's feedback, enabling a balance between exploration and exploitation.

Single-agent RL frameworks, despite their advantages, often have limited application in multi-agent settings [25]. Within a datacenter, servers are managed independently, with their own set of operational requirements and constraints. Moreover, the inherent objective of each server is to maximize its individual performance rather than the collective efficiency of the entire datacenter. This independent and self-centered operation can lead to suboptimal solutions when applying a single-agent paradigm. Hence, a more holistic multi-agent framework would be better suited to manage the intricacies and interdependencies of large-scale datacenters.

2.2.2 Multi-agent RL

Multi-agent RL (MARL) studies behavior of multiple learning agents interacting in a shared environment [66]. Designing MARL methods is particularly challenging when the system comprises a large number of agents. This complexity arises from the need to model each agent, leading to prohibitive computational and memory costs. To address this, a promising approach is to utilize mean-field learning [65]. Similarly to the framework of mean-field games, mean-field learning simplifies the learning process by approximating many-body interactions with two-body interactions, essentially between an individual agent and the population average.

2.3 Differential Privacy

In today's data-driven world, data privacy has emerged as one of the most pressing challenges. Differential privacy (DP) is a robust privacy-preserving technique designed to protect individual data [15]. DP provides a strong mathematical privacy guarantee allowing data to be used without revealing sensitive information about any datapoint in a dataset. In its core, this guarantee is achieved by adding noise to the data, ensuring that individual data points cannot be distinguished.

Definition 1 (Differential privacy [15])**.** *Given adjacent datasets*[2] x *and* x'*, a mechanism* M*, refers to a mathematical function or algorithm that takes a dataset as input and produces an output, satisfies* (ϵ, δ)*-DP if for every subset of outputs* O*, it satisfies:*

$$\mathbb{P}[M(x) \in O] \leq e^{\epsilon}\, \mathbb{P}[M(x') \in O] + \delta.$$

This definition ensures that the presence or absence of any individual datapoint in the dataset does not significantly affect the outcome of a DP mechanism. The parameter ϵ determines the desired level of privacy. Typically, smaller values indicate stronger privacy assurances. However, they also necessitate higher levels of injected noise, potentially impacting the quality of the mechanism's output. When $\delta = 0$, we have the standard ϵ-DP. For $\delta > 0$, the common interpretation of (ϵ, δ)-DP is that it approximates ϵ-DP with a probability of $(1 - \delta)$.

The property of *post-processing* ensures that the DP guarantee remains unaffected even when manipulating the output of a mechanism. If M satisfies (ϵ, δ)-DP, then applying a

[2]Two datasets x and x' are adjacent if they differ only in one element.

mapping f to $M(x)$ retains the same (ϵ, δ)-DP assurance (Proposition 2.1 in [16]). Moreover, the concept of DP can be expanded and generalized by incorporating the notion of the *Rényi divergence*, defined as follows.

Definition 2 (Rényi divergence [52])**.** *For probability distributions P and Q, the Rényi divergence of order $\alpha > 1$ is:*

$$D_\alpha(P\|Q) \triangleq \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{x\sim Q} \left[\left(\frac{P(x)}{Q(x)} \right)^\alpha \right] \right),$$

where $P(x)$ $(Q(x))$ is the density of P (Q) at x.

When $\alpha = \infty$, the Rényi divergence is defined as:

$$D_\infty(P\|Q) = \sup_{x \in \text{supp } Q} \log \left(\frac{P(x)}{Q(x)} \right).$$

Here, sup means "supremum", while supp means "support". In the context of the Rényi divergence definition, when $\alpha = \infty$, the Rényi divergence $D_\infty(P\|Q)$ is defined by taking the supremum of the expression $\log \frac{P(x)}{Q(x)}$ over all x in the support of Q, denoted as supp Q. This means we are looking for the maximum value of $\log \frac{P(x)}{Q(x)}$ for all x where $Q(x)$ is non-zero. It is straightforward to show that a mechanism M is ϵ-DP if and only if $D_\infty(M(x)\|M(x')) \leq \epsilon$ for any two adjacent inputs x, x'. Here D_∞ denotes the Rényi divergence of order ∞, which is the worst-case divergence where the rarest event in $M(x)$ should not become too probable in $M(x')$. $M(x)$ and $M(x')$ represent the outputs of the mechanim M when applied to two adjacent datasets x and x', respectively. $\|$ symbolizes 'given' in the context of divergence. The statement as a whole asserts that for the mechanism M to be ϵ-differentially private, the worst-case of the output distributions for any two adjacent inputs must be bounded by ϵ. This motivates a relaxation of DP called *Rényi differential privacy (RDP).*

Definition 3 (Rényi differential privacy [44])**.** *A mechanism M is (α, ϵ)-RDP with order $\alpha > 1$ if for any two adjacent datasets x and x', it satisfies: $D_\alpha(M(x)\|M(x')) \leq \epsilon$.*

In this paper, we focus on RDP due to its ability to offer a more precise analysis of composition. Suppose that mechanisms M_1 and M_2 satisfy ϵ_1-DP and ϵ_2-DP, respectively. Then simultaneous release of the outputs of M_1 and M_2 guarantees $(\epsilon_1 + \epsilon_2)$-DP. This guarantee remains valid even when M_2 is adaptively selected based on the output of M_1. Similar guarantee holds for the composition of two RDP mechanisms, even under adaptive, sequential composition:

Lemma 4 (RDP composition [44]). *Let M_1 be (α, ϵ_1)-RDP, and let M_2 be (α, ϵ_2)-RDP. Then the mechanism $M_{1,2}$ defined as $M_{1,2}(x) \triangleq (M_1(x), M_2(x))$ is $(\alpha, \epsilon_1 + \epsilon_2)$-RDP.*

Lemma 4 mathematically expresses the intuitive concept of *privacy budget*. Here, $(M_1(x), M_2(x))$ represents the output of a composite mechanism, denoted $M_{1,2}(x)$, when it is applied to an input dataset x. The additivity property of RDP enables straightforward tracking of cumulative privacy loss throughout the iterative operation of a mechanism. This property is particularly useful in monitoring privacy loss over successive steps in a mechanism's execution.

To guarantee RDP, in this paper, we utilize the *Gaussian mechanism*. To formally define the Gaussian mechanism, we first define the sensitivity of a function as follows.

Definition 5 (Sensitivity). *Let $g : X \mapsto \mathbb{R}$ be a real-valued function. The sensitivity of g is defined as:*

$$\Delta(g) \triangleq \max_{x,x' \in X, d(x,x')=1} |g(x) - g(x')|$$

where the max is taken over all adjacent datasets x and x'.

Given a function g, the Gaussian mechanism computes the output of the function and perturbs it with noise drawn from a Gaussian distribution. The magnitude of the noise is adjusted based on the sensitivity of the function.

Definition 6 (Gaussian mechanism). *Suppose that $g : X \mapsto \mathbb{R}$ is a real-valued function with a sensitivity of $\Delta(g)$. Let $\mathcal{N}(\mu, \sigma^2)$ denote a normal distribution with mean μ and standard deviation σ. For $\alpha > 1$ and $\epsilon > 0$, the Gaussian mechanism $M^G_{g,\alpha,\epsilon}$ is defined as:*

$$M^G_{g,\alpha,\epsilon}(x) \triangleq g(x) + \mathcal{N}(0, \sigma^2)$$

Essentially, $M^G_{g,\alpha,\epsilon}(x)$ applies the function g to x and then adds noise represented by the random variable Y, which is sampled from a Gaussian distribution with mean 0 and variance σ^2, to produce the final output. This mechanism's action can be represented as $M^G_{g,\alpha,\epsilon}(x) = g(x) + Y$.

Lemma 7 (Gaussian mechanism and RDP [44]). *Let $g : X \mapsto \mathbb{R}$ be a real-valued function with the sensitivity of $\Delta(g)$. Then $M^G_{g,\alpha,\epsilon}$ is (α, ϵ)-RDP for a specific set or range of α satisfying certain conditions..*

Finally, RDP implies DP:

Lemma 8 (From RDP to DP [44]). *Suppose that mechanism M is (α, ϵ)-RDP. Then M is $(\epsilon + \log(1/\delta)/(\alpha - 1), \delta)$-DP for any $0 < \delta < 1$.*

Chapter 3

Mechanism

In this section, we first introduce PACS, a private and adaptive computational sprinting framework, and describe its architecture. We then prove that PACS guarantees differential privacy.

3.1 Assumptions

We describe the PACS architecture considering a multi-tenant datacenter environment with N tenants and a single datacenter provider. In this work, we make the following assumptions.

- Each tenant controls a single server.
- The datacenter provider lacks direct control over servers' sprinting behavior, which is solely governed by the tenants of the servers. The datacenter provider can only indirectly influence system-wide sprinting behavior through cost measures. It is essential to note that if the datacenter provider were to have complete control over the operation of servers, the optimization of computational sprinting decisions would transition from a multi-agent game-theoretic problem to a single-agent optimization problem.
- It is physically possible for the datacenter provider to monitor (i.e., smart Power Distribution Units (PDU), anch Circuit Monitors (BCMs), Data Center Infrastructure Management (DCIM)) the power consumption of the tenants' servers. And the datacenter provider can detect whether a server sprints based on its monitored power consumption. Therefore, the exact number of servers that sprint can be accurately tracked at any given time.

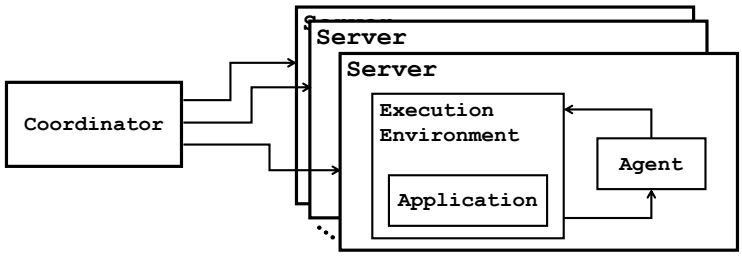


Figure 3.1: PACS architecture

- Similarly to [18], we assume that the probability that a power emergency happens is a piecewise-linear function of the number of sprinting servers. If the number of sprinting servers is less than $N_{\min}$, then this probability is 0. If the number of sprinting servers is more than $N_{\max}$, then this probability is 1. And if the number of sprinting servers is between $N_{\min}$ and $N_{\max}$, then the probability increases linearly with the number of sprinting servers.

We discuss how some of these assumptions can be relaxed at the end of this section.

3.2 PACS Architecture

Figure 3.1 shows a high-level overview of the system architecture in PACS. The datacenter provider operates the coordinator, which monitors the power consumption of all servers and indirectly influences the system-wide sprinting behavior through the implementation of a tax system in a differentially private manner. Here, the tax system is a cost function which is applied to agents to charge them based on their actions (sprint or not sprint). Each tenant manages a server. Within each server, PACS consists of two main components: an execution environment and an agent. The execution environment runs the tenant's application and executes sprinting decisions. The agent optimizes the sprinting strategy based on the application's state (e.g., number of outstanding tasks), the server state (i.e., cooling or active), and the datacenter state (i.e., the average fraction of sprinting servers and the taxes).

Remark. While seemingly similar at first glance, there exist notable distinctions between the sprinting architecture outlined in [18] and the one in PACS. Firstly, in [18], tenants are required to profile their applications and transmit the profiled data to the coordinator. The coordinator then employs these reported profiles to statically derive equilibrium strategies for the tenants. However, due to the model-free learning approach

in PACS, there is no requirement for application profiling. Furthermore, tenants compute their equilibrium strategies privately and locally in a dynamic and adaptable manner. Secondly, in [18], tenants are required to deploy a predictor to gauge utility from a sprint based on application profiles and hardware counters. In contrast, in PACS, agents base sprinting decisions on the current state of applications (for more details, refer to §3.2.2).

3.2.1 Coordinator

The coordinator monitors the number of servers that sprint. In PACS, time is divided into rounds and every L rounds form an epoch. Similarly to [18], PACS sets the length of each round based on the duration of a safe sprint. The duration of rounds and the length of epochs are configurable parameters of the system. At the end of each epoch, the coordinator calculates the average fraction of sprinting servers during the epoch.

Let n_e^r denote the number of sprinting servers at round r during epoch e. Then the average fraction of sprinters in epoch e is simply formulated as:

$$f_e = \frac{1}{N \times L} \sum_{r=1}^{L} n_e^r.$$

This is a critical statistic that serves as the mean-field parameter in PACS . This parameter enables the agents to independently optimize their sprinting strategies while considering the collective behavior of the entire server population.

To tenants, f_e is not solely a technical parameter; it holds sensitive information about their behavior and their application characteristics. Revealing precise details on the average fraction of sprinting servers per epoch could expose system vulnerabilities or strategic operations, raising significant privacy concerns. This becomes especially critical when considering potential malicious entities seeking to exploit leaked information. To counter this, the coordinator introduces calibrated noise to f_e using the Gaussian mechanism as follows:

$$\bar{f}_e = f_e + v, \tag{3.1}$$

where v is a random variable drawn from $\mathcal{N}(0, \sigma^2)$. The value of σ is determined to ensure a desired level of privacy guarantee (refer to §3.3 for further details). This strategy ensures that no single sprinting decision can be inferred with high confidence from the released data while preserving the utility of the data for system-wide decisions.

The coordinator does not have direct control over servers' sprinting behavior. Therefore, to indirectly shape the system-wide sprinting behavior, PACS implements a tax system

based on agents' sprinting decisions. This taxation entails an extra monetary cost to the tenants. In PACS, there are two types of taxes: the group tax and the individual tax. The group tax is proportional to the probability of triggering a power emergency. For epoch e with an average fraction of f_e sprinting servers, the group tax is calculated as:

$$t_e^G = \min\left(\max\left(\frac{f_e \times N - N_{\min}}{N_{\max} - N_{\min}}, 0\right), 1\right) \times T_G,$$

where, T_G is the *group taxation coefficient*, a configurable parameter. Moreover, to discourage having more than $N_{\max}$ sprinters on average, PACS imposes the individual tax. Let $n_{e,i}$ be the number of rounds at which agent i sprints during epoch e. Then the individual tax imposed on agent i at epoch e with an average fraction of f_e sprinting servers is:

$$t_{e,i}^I = \begin{cases} n_{e,i} \times T_I & \text{if } f_e \times N \geq N_{\max}, \\ 0 & \text{otherwise}, \end{cases}$$

where, T_I is the *individual taxation coefficient*, a configurable parameter in PACS . Therefore, the total tax imposed on agent i at epoch e is:

$$t_{e,i} = t_e^G + t_{e,i}^I. \tag{3.2}$$

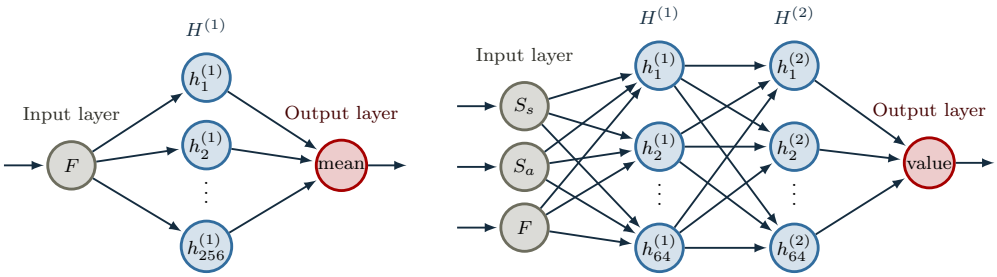
Using f_e to calculate taxes could indirectly reveal f_e to agents and violate differential privacy. To prevent this, instead of using f_e to calculate taxes, the coordinator employs $\bar{f}_e$. Since DP is preserved by post-processing, revealing $t_{e,i}$ calculated based on $\bar{f}_e$ does not impact the DP guarantee of PACS.

Algorithm 1 shows the pseudocode for the coordinator's operations. At every round, the coordinator monitors the actions of all agents. At the end of each epoch, the coordinator calculates the average fraction of sprinters, alongside the taxation, and sends this information to the agents.

The input parameters utilized by the coordinator are K, L, ϵ, δ, and α. The value of K specifies the number of epochs for which the algorithm ensures DP guarantees (further details in §3.3). The value of L corresponds to the length of each epoch. The remaining parameters, namely ϵ, δ, and α, dictate the desired level of the privacy guarantee. The value of ϵ and δ directly govern the DP assurance provided by the algorithm (refer to Theorem 9). And the value of α manages the noise variance in accordance with the Gaussian mechanism.

Algorithm 1: High-level pseudocode for the coordinator in PACS

parameters: $K, L \in \mathbb{Z}$, $\epsilon, \delta \in (0, 1)$, $\alpha > 1$;
$\epsilon' \leftarrow (1/K)(\epsilon - \log(1/\delta)/(\alpha - 1))$;
$\sigma^2 \leftarrow \alpha/2n^2\epsilon'$;
run `coordinator()`:
  foreach epoch e **do**
    for round $r = 1, \ldots, L$ **do**
      $n_e^r \leftarrow$ number of sprinting agents;
    $\bar{f}_e \leftarrow$ Equation (3.1);
    $t_{e,i} \leftarrow$ Equation (3.2) for each agent i;
    send $\bar{f}_e$ and $t_{e,i}$ to each agent i;



(a) Actor network (b) Critic network

Figure 3.2: Architecture of the actor and the critic networks

3.2.2 Agents

The primary responsibility of agents is to make sprinting decisions on behalf of their respective tenants. To make these decisions, PACS agents employ an actor-critic learning approach. The actor maintains the sprinting strategy, while the critic estimates the value function. In PACS, both the actor and the critic are implemented using neural networks. For a visual representation, refer to Figure 3.2 illustrating the architecture of the actor and critic networks in PACS.

Agents receive $\bar{f}_e$ and $t_{e,i}$ at each epoch from the coordinator. At each round r, the agent queries the server to determine the server state, $s_{S_i}^r$, which is either cooling or active. Additionally, the agent queries the execution environment to establish the application state, $s_{A_i}^r$, which could, for example, be the number of outstanding tasks, request load, or input

Algorithm 2: High-level pseudocode for agent i in PACS

parameters: $L, M \in \mathbb{Z}$;
run `agent(`i`):`
  foreach epoch e **do**
    recieve $\bar{f}_e$ and $t_{e,i}$ from coordinator;
    for round $r = 1, \ldots, L$ **do**
      $s_i^r \leftarrow (s_{S_i}^r, s_{A_i}^r, \bar{f}_e)$;
      $a_i^r \leftarrow \text{policy}(s_i^r)$;
      $u_i^r \leftarrow \text{utility}(a_i^r, t_{e,i})$;
      add (s_i^r, a_i^r, u_i^r) to buffer;
      if $0 \equiv r \pmod{M}$ **then**
        update policy;
        empty buffer;

size. The critic network takes the tuple $(s_{S_i}^r, s_{A_i}^r, \bar{f}_e)$ as input and produces an estimate of the state value as output.

Given that equilibrium strategies in the computational sprinting game are simple threshold strategies [18], the actor network's purpose is to learn such a threshold. As such, the actor network takes $\bar{f}_e$ as input and produces an output that serves as the mean of a normal distribution to estimate the optimal threshold. It is important to note that s_{S_i} and s_{A_i} are not included in the input to the actor network. s_{S_i} is not used because the actor is only consulted when the server is in the active state, while s_{A_i} is excluded because the network's output represents a threshold on s_{A_i}. Figure 3.2 presents the architecture of the actor and the critic networks in PACS.

Outlined in Algorithm 2 is the high-level pseudocode depicting the operations of each agent. The input parameters include L, denoting the size of each epoch, and M, indicating the mini-batch size. During each round, the agent follows a sequence: constructing the system state, consulting the policy, taking an action, and receiving a utility. These (state, action, utility) tuples are stored in a buffer, facilitating mini-batch policy updates occurring every M rounds.

3.3 DP Guarantee in PACS

To analyze the DP guarantee of an iterative algorithm, it is common to individually characterize the DP guarantee of each iteration. The end-to-end DP guarantee of the algorithm is then determined using the composition rule (e.g., Lemma 4). In PACS, the coordinator is required to compute the average fraction of sprinting servers across an infinite stream of events (i.e., sprinting actions by agents at every round) and release this statistics periodically at the end of each epoch. This entails a composition of an infinite number of iterations. To ensure DP over an infinite composition of iterations, an infinite amount of noise would be required, which destroys the utility of the released data. To avoid this, PACS offers a K-epoch DP guarantee. This safeguard protects the privacy of sequences of sprinting decisions occurring within any window of K epochs.

Using the properties the Gaussian mechanism, we now show that each epoch of Algorithm 1 satisfies (α, ϵ')-RDP. Subsequently, by employing Lemma 4 and Lemma 8, we establish PACS as (ϵ, δ)-DP over any K epochs.

Theorem 9 (PACS is DP)**.** *Algorithm 1 provides (ϵ, δ)-DP over any K epochs.*

Proof. At each epoch e, the private data of agent i is the number of times agent i sprints during the epoch (i.e., $n_{e,i}$), and the publicly released data is $\bar{f}_e$. Note that $t_{e,r}$ is also released at each epoch, but it is calculated based on $\bar{f}_e$. This calculation is considered as post-processing calculation, while DP guarantees are preserved under post-processing. This means that any computation applied to the output of a differentially private mechanism does not weaken the privacy guarantee. The fundamental reason for this is that differential privacy is immune to any function that does not add new sensitive information. The calculation of $\bar{f}_e$ is a direct application of the Gaussian mechanism with the real-valued function:

$$g(n_e) = \frac{1}{N \times L} \sum_{i=1}^{N} n_{e,i}, \text{ where } n_e = (n_{e,1}, \ldots, n_{e,N}). \tag{3.3}$$

Let n_e and n'_e be two adjacent inputs that are identical except in their ith element ($n_{e,j} = n'_{e,j}$ if $j \neq i$ and $n_{e,i} \neq n'_{e,i}$). Since $n_{e,i}$ and $n'_{e,i}$ are integer numbers from 0 to L, it is easy to show that $|g(n_e) - g(n'_e)|$ is maximized when $n_{e,i} = 0$, and $n'_{e,i} = L$ (or vice versa). This means that the sensitivity of function g is $\Delta(g) = 1/N$. Therefore, it follows from Lemma 7 that each epoch e is (α, ϵ')-RDP. Consequently, according to Lemma 4, the composition of any K epochs satisfies $(\alpha, \bar{\epsilon})$-RDP, where

$$\bar{\epsilon} = \sum_{k=1}^{K} \epsilon' = K\epsilon' = \epsilon - \log(1/\delta)/(\alpha - 1).$$

Finally, it follows from Lemma 8 that Algorithm 1 provides (ϵ, δ)-DP across any K epochs. □

Remarks. We make three final remarks. First, adding noise introduces a trade-off between accuracy and privacy. Learning precise equilibrium strategies often demands extensive training epochs, implying selecting larger values for K. However, when the privacy loss per epoch is fixed, this higher number of epochs results in an increased cumulative privacy loss, which weakens the overall privacy guarantee. Conversely, aiming for a stronger privacy guarantee requires a reduction in the cumulative privacy loss. But, with a fixed number of epochs, minimizing cumulative privacy loss leads to higher noise per epoch, consequently impacting accuracy adversely.

Second, in PACS, the variance of the added noise at each epoch is quantified as:

$$\sigma^2 = \frac{K\alpha}{2n^2(\epsilon - \log(1/\delta)/(\alpha - 1))}. \tag{3.4}$$

For a constant K, the variance of added noise at each iteration of the algorithm converges asymptotically to zero as n grows large if we set $\epsilon = \Theta(1/\log(n))$, $\delta = \Theta(1/n)$, and $\alpha = \log(1/\delta)/(\alpha - 1) = 0.5\epsilon$.

Third, we note that some of the assumptions made at the beginning of this section can be relaxed. For instance, the initial assumption that each tenant controls a single server can be modified to assume that each tenant has control over at most C servers, where C is a constant. This adjustment directly changes the sensitivity of the function g (defined in (3.3)) from $1/N$ to C/N. Consequently, additional noise needs to be introduced, as explained in the analysis in the proof of Theorem 9. The magnitude of added noise is inversely proportional to the value of C, meaning that lower values of C result in reduced additional noise as the formula:

$$g(n_e) = \frac{1}{C \times L} \sum_{i=1}^{N} n_{e,i}$$

Chapter 4

Experimental Methodology

In this section, we begin by presenting the three categories of applications utilized to assess PACS: (1) synthetic applications, (2) hybrid applications, and (3) real-world applications. Subsequently, we introduce three baseline mechanisms employed for comparison against PACS. Finally, we discuss the fine-tuning process to optimize the parameters of PACS. The source code for our implementation of PACS alongside other baselines and the benchmarks is available at https://anonymous.4open.science/r/PACS-D590.

App.	$\mathbf{P_{ij}}$
M_1	0.5 if $i = j \pm 1$; 0 otherwise
M_2	0.4 if $i = j - 1$; 0.6 if $i = j + 1$; 0 otherwise
M_3	0.6 if $i = j - 1$; 0.4 if $i = j + 1$; 0 otherwise
M_4	$\propto \lvert j - i \rvert + 1$
M_5	$\propto 1/(\lvert j - i \rvert + 1)$
M_6	$\propto 1/(j + 1)$
M_7	$\propto (j + 1)^2$
M_8	0.1

Table 4.1: Transition probabilities for synthetic applications.

App.	Model	Baseline TPS	Sprinting TPS
Q_1	`ResNet`	41	86
Q_2	`MobileNet`	44	112
Q_3	`SqueezeNet`	25	75
Q_4	`AlexNet`	11	32

Table 4.2: Throughput parameters for hybrid benchmarks

4.1 Synthetic Benchmarks

To evaluate PACS, we examine eight synthetic *Markov* applications. Each Markov application M encompasses a finite state space S_M. At each state $i \in S_M$, the utility gain from sprinting is denoted as u_i. The application's state transitions from state i to state j with a probability represented by $P_{i,j}$. The specifics of $P_{i,j}$ for various applications and states are summarized in Table 4.1. This Markov model generalizes the utility model outlined in [18], where $P_{i,j} = f(j)$ for some probability density function f over all states.

In our experiments, each Markov application consists of 10 states. At state i, the performance gain from sprinting is assumed to be i ($u_i = i$). If an agent sprints, the total utility is the corresponding performance gain minus the tax (see §3.2.1). For each tenant, we initialize each application in a randomly selected state.

Among the eight applications, M_1 to M_3 follow birth-death Markov models, where state transitions include only two types: "births," increasing the state variable by one, and "deaths," decreasing the state variable by one. Conversely, the remaining applications–M_4 through M_8–feature a more intricate state transitioning structure, allowing transitions to any state from the current one. In M_4 and M_5, transition probabilities rely on the distance between states (direct dependence in M_4 and inverse dependence in M_5). In M_6 and M_7, transition probabilities are contingent upon the destination (direct dependence in M_6 and inverse dependence in M_7). Moreover, in M_8, the current state transitions to any other state with equal probability.

4.2 Hybrid Benchmarks

For our hybrid benchmarks, we focus on deep neural network (DNN) inference workloads. Tenants deploy DNN inference models on servers equipped with DNN accelerators. Each

server receives inference tasks at a fixed rate. These tasks are queued in the server while waiting to be executed. To assess the performance gain from a sprint, we profile real-world DNN inference tasks on the RocketChip SoC [5] equipped with a Gemmini-generated DNN accelerator [20], using the cycle-accurate FireSim simulator [35].

4.2.1 DNN Inference Tasks

We profile DNN inference tasks from four models: `ResNet` [28], `MobileNet` [29], `SqueezeNet` [31], and `AlexNet` [37]:

- `ResNet` is a DNN used for image recognition.
- `MobileNet` is a DNN designed for mobile and embedded devices.
- `SqueezeNet` is a resource-efficient DNN for image classification.
- `AlexNet` is a DNN for image recognition and classification. We use a pretrained version of the network trained on more than a million images from the ImageNet database.

We measure the throughput of the inference tasks on models that are pre-trained using ImageNet [53].

4.2.2 Testbed

We use FireSim [35] to simulate our testbed. FireSim is an FPGA-accelerated cycle-accurate simulator for full-system hardware simulations. Our testbed comprises a Rocket Chip SoC [5] featuring an in-order Rocket Core paired with a Gemmini-generated DNN accelerator [20]. Each SoC is configured with 4GiB off-chip DDR3 DRAM.

The DNN accelerator is set to operate as a weight-stationary 16×16 systolic array. In the baseline power mode, the CPU core and the accelerator run at 1GHz. To represent the effect of a sprint on application performance, we elevate the SoC clock frequency to 3 GHz. This higher frequency is applied across all SoC components, including the core, accelerator, and buses. However, the off-chip memory components (DDR3 DRAM) maintain the baseline frequency.

DNN inference models are executed using the `ONNX` [1] runtime framework, specifically ported to our simulated environment. The framework utilizes Gemmini-generated DNN accelerators for matrix-multiplication (`matmul`), convolution (`conv`), and activation operations, with the Rocket core handling operations like serialization. Measured throughput in terms of tasks per second (TPS) for the inference models on the baseline and power-boosted SoC is presented in Table 4.2.

App.	Algorithm	Category	Dataset	Data size
S_1	`ALS`	Collaborative Filtering	movielens2015 [27]	891M
S_2	`Kmeans`	Clustering	uscensus1990 [38]	345M
S_3	`Linear Regression`	Classification	kddb2010 [59]	782M
S_4	`PageRank`	Graph Processing	wdc2012 [43]	3.4G
S_5	`SVM`	Classification	kdda2010 [59]	522M

Table 4.3: Spark Workloads

4.2.3 Queueing Model

Tenants receive inference tasks at a fixed arrival rate, with the number of tasks per round following a Poisson distribution. These tasks, whether during sprinting or normal operation, are presumed to have service times that fit an Exponential distribution. This assumption is due to the Exponential distribution's memorylessness property, meaning the time to complete a task is independent of how long it's been processed. This simplifies mathematical modeling and mirrors many real-world processes where the remaining task time doesn't depend on the elapsed processing time. The service rate is determined by the throughput measured from our cycle-accurate simulations. Upon arrival, inference tasks are queued for execution. The utilities of tenants are determined based on the length of their queues. Specifically, at each round, tenants incur a cost–reflecting a form of delay or latency–proportional to their queue lengths. Consequently, the longer the queue, the higher the incurred cost.

4.3 Real-world Benchmarks

For our real-world benchmarks, we focus on Apache Spark workloads [68], which are also used in [18]. We run five machine learning (ML) applications from the Spark machine learning library (a.k.a. `MLlib`): alternating least squares (`ALS`), `K-means`, linear regression (`LR`), `PageRank`, and support vector machine (`SVM`). Table 4.3 summarizes the datasets used to run each of the five ML applications.

We run the Spark workloads on an AMD Ryzen Threadripper PRO 3945WX 12-Cores server. We use the same setting as [18] that in the baseline power mode, workloads execute on three cores running at 2.2GHz cores, while in the sprinting mode, workloads execute on all twelve cores running at 4.2GHz. Spark is configured to run locally using different

number of threads by setting `--master local [k]`, where `k` is 12 in the sprinting mode and 3 in the baseline power mode.

We collect executor log files and measure the TPS for each application in both baseline and sprinting power modes. We track TPS throughout the end-to-end execution of an application in both modes. Each application is defined with a set of jobs, and each job is further divided into tasks that run in parallel. While jobs are completed sequentially, tasks can finish in any order. The total number of tasks in a job remains constant and is independent of the available hardware resources. Consequently, TPS serves as a performance metric for a fixed amount of work. Given that the execution times in sprinting mode are shorter than the baseline power mode, we extend the traces of the sprinting mode for comparison with baseline traces. Linear interpolation is used to extend our sprinting traces because it provides a quick and straightforward method for estimating the missing data points, allowing for a consistent comparison with the longer-duration baseline traces.

We calculate performance gains from sprinting as the difference in throughput between the baseline and sprinting power modes. If an agent sprints, the total utility is equal to the corresponding performance gain minus the tax (see §3.2.1). Agents are assigned random starting points within the trace. Once an application reaches the end of a trace, it recommences from the beginning and continues until completion.

4.4 System Parameters

We use the same parameters as those employed to evaluate the computational sprinting game in [18] and described in §2.1. Specifically, a server in the cooling state remains in this state for $\Delta t_{\text{cooling}}$ rounds, where $\Delta t_{\text{cooling}}$ follows a geometric distribution with parameter p_c. Here, p_c represents the probability of a server in the cooling state remaining in that state, while the probability of transitioning back to the active state is $(1 - p_c)$. We set p_c to 0.5. Consequently, if agents sprint whenever their servers are active, servers spend an average of 2/3 of the rounds in the cooling state.

4.5 Baselines

We compare PACS against three other baseline methods: dynamic programming (representing the computational sprinting game [18]), cooperative thresholds, and PACS w/o noise:

Parameter	Value
Discount factor (γ)	0.9999
Mini-batch size (M)	5
Learning rates for M_1–M_8	1×10^{-4}, ..., 9×10^{-4}
Learning rates for Q_1–Q_4	1×10^{-3}, 2×10^{-3}
Actor learning rates for S_1–S_5	1×10^{-4}, ..., 4×10^{-4}
Critic learning rates for S_1–S_5	1×10^{-3}, 2×10^{-3}, 3×10^{-3}
Number of hidden layers for actor	1
Number of hidden layers for critic	2

Table 4.4: Learning parameters in PACS

- **Dynamic programming (DyProg).** The core of the dynamic programming approach lies in the value iteration method, an iterative algorithm to solve Bellman equations [7]. At each iteration, the algorithm calculates the expected utility of sprinting versus not sprinting, factoring in both the immediate utility and the discounted future utilities. It then updates the policy to choose the action that maximizes utility. This process continues until the policy stabilizes, indicating an optimal or near-optimal solution, which is the best threshold value. This is the method proposed in [18].
- **Cooperative threshold (Thr).** The cooperative threshold approach involves agents collaboratively deploying globally optimal thresholds for sprinting. These threshold values are determined through a brute-force search aimed at maximizing the total system performance. However, these thresholds do not constitute an equilibrium, because they do not correspond to agents' best responses to the system dynamics. Consequently, their enforcement relies on the datacenter's intervention. The cooperative threshold's performance serves as an upper bound on the achievable performance under any static threshold method.
- **PACS w/o noise.** The primary distinction between PACS and PACS w/o noise lies in PACS's assurance of differential privacy through the addition of noise, whereas PACS w/o noise does not incorporate DP. In both approaches, the coordinator collects actions from all agents and computes the fraction of sprinters and associated taxes. In PACS, the coordinator introduces noise to the fraction of sprinters, while this step is omitted in PACS w/o noise.

4.6 Fine-tuning Process

Table 4.4 summarizes the learning parameters used in PACS. Since neural networks are often sensitive to their parameters, for our experiments, we employ simple hyperparameter tuning via `Optuna`, a library designed for precise hyperparameter selection [3]. The objective function is aimed at maximizing the cumulative average utility, with the learning rates of the actor and the critic as the parameters we tune.

Chapter 5

Evaluation

In this section, we assess the performance of PACS by comparing it with other baseline methods for synthetic, hybrid, and real-world applications. We further illustrate PACS's adaptability when changes occur in system dynamics. Finally, we provide a sensitivity analysis for PACS to show the trade-off between data privacy and performance.

5.1 Performance

In this subsection, we empirically analyze the performance of PACS. Each experiment is conducted 10 times, and we report the average performance along with its 95% confidence level. The 95% confidence interval (CI) for the average performance is computed by the mean and standard deviation (std) as:

$$CI = \bar{x} \pm (t \times \frac{1}{\sqrt{N}} \times \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}})$$

, where the t-score for a two-tailed test with a 95% confidence level is approximately 2.262. To facilitate comparison, we normalize the performances of all methods by the performance of the cooperative threshold method.

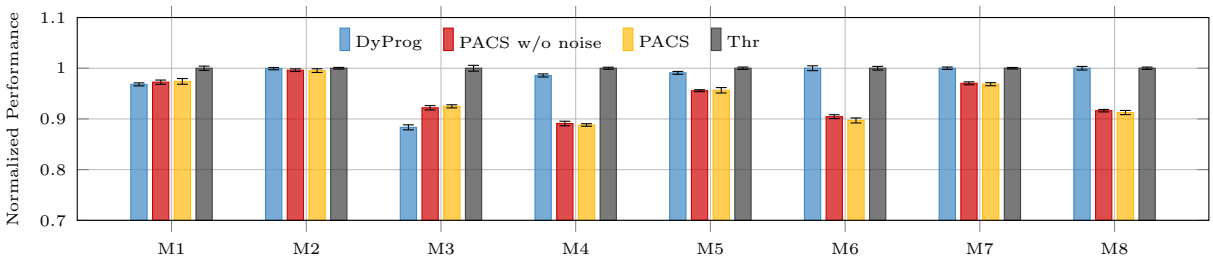


Figure 5.1: Performance of different methods for synthetic applications.

5.1.1 Synthetic Applications

Figure 5.1 illustrates the performance of different methods for synthetic applications. The cooperative threshold method achieves the highest performance for all applications. As cana be seen, PACS and PACS w/o noise achieve similar average performances (less than a 2% difference). However, the variance in performance is higher under PACS due to the added random noise.

PACS and PACS w/o noise perform as well as dynamic programming for M1-M3. However, dynamic programming outperforms PACS and PACS w/o noise for M4-M8 by up to 10%. The ratio between the performance of PACS w/o noise and dynamic programming for M4-M8 is approximately 90%, 96%, 90%, 97%, and 91%, respectively.

PACS and PACS w/o noise perform better for M1-M3 primarily because PACS is deliberately designed with a simple neural network architecture to minimize runtime overheads. This simple architecture can easily represent the straightforward application dynamics of M1-M3 (i.e., the birth-death Markov model). However, it performs suboptimally when tasked with representing more complex application models such as those in M4-M8 (refer to Table 4.1). Additionally, to protect data privacy, the privacy window is set to 120 epochs. This short training period may not be sufficient to train a neural network to fully represent a complex Markov environment.

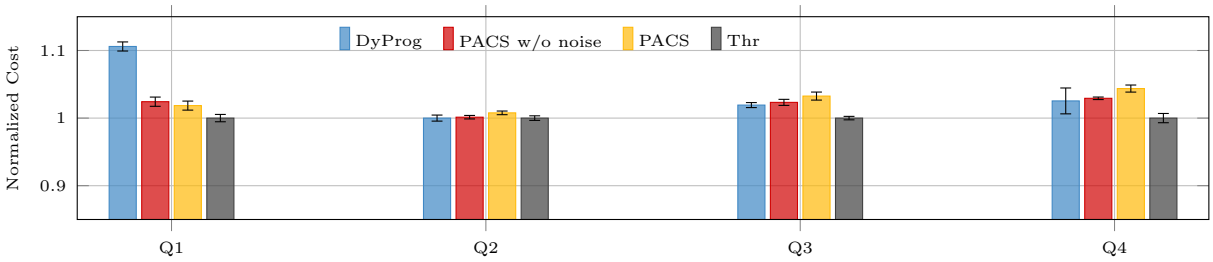


Figure 5.2: Cost of different methods for synthetic applications.

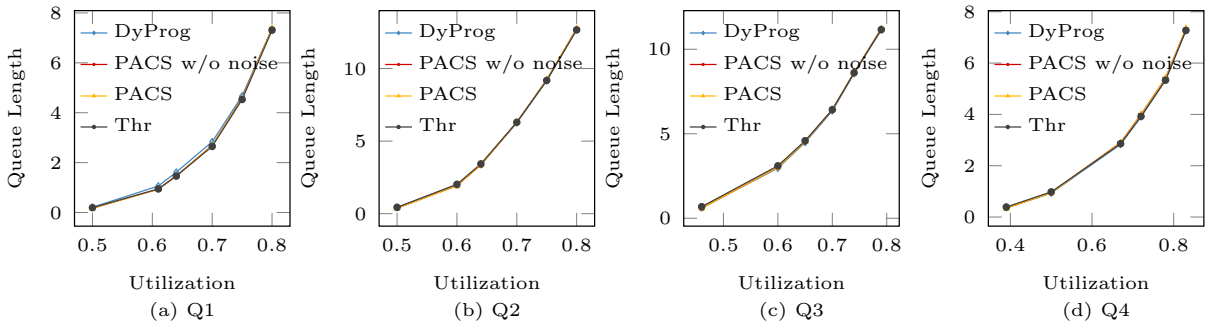


Figure 5.3: Queue lengths for different utilization under different methods.

5.1.2 Hybrid Applications

For hybrid applications, tenants incur costs proportional to the length of their inference queues (refer to §4.2.3). Tenants are additionally taxed based on their cumulative power consumption, as discussed in §3.2.1. Therefore, for hybrid applications, lower costs correspond to superior performance.

Figure 5.2 shows the average cost incurred by tenants under different baselines for hybrid applications. Here, the cost is $-\max(0, L_{curr} + R_{arr} - R_{dep})$, where L_{curr} is the current queue length, R_{arr} is the arrival rate and the R_{dep} is the departure rate which depends on sprint or not. For these experiments, the utilization (i.e., the ratio of the arrival rate to the departure rate) for all hybrid applications is set to about 65%. The cooperative threshold method achieves the highest performance (lowest costs). PACS w/o noise outperforms the dynamic programming method for Q1. Q1 has the lowest ratio between sprinting throughput and baseline throughput compared to the other applications.

PACS w/o noise performs nearly as well as dynamic programming for Q2 to Q4 (the difference is less than 1%).

It is essential to highlight that dynamic programming finds the static equilibrium thresholds, whereas PACS determines dynamic thresholds. In other words, the dynamic programming method outputs a single threshold, while the actor network in PACS could output different thresholds for different system states. PACS and PACS w/o noise incur similar costs, with a difference of less than 2%, while, as expected, the variance is lower under PACS w/o noise.

Figure 5.3 shows the average queue length of all tenants under different baselines for a variety of utilization ratios, ranging from 38% to 83%. It is easy to see that the average queue lengths are quite similar under different methods for different utilization ratios. This shows that PACS achieve similar performance compared to the dynamic programming method while providing adaptability and guaranteeing data privacy for all tenants.

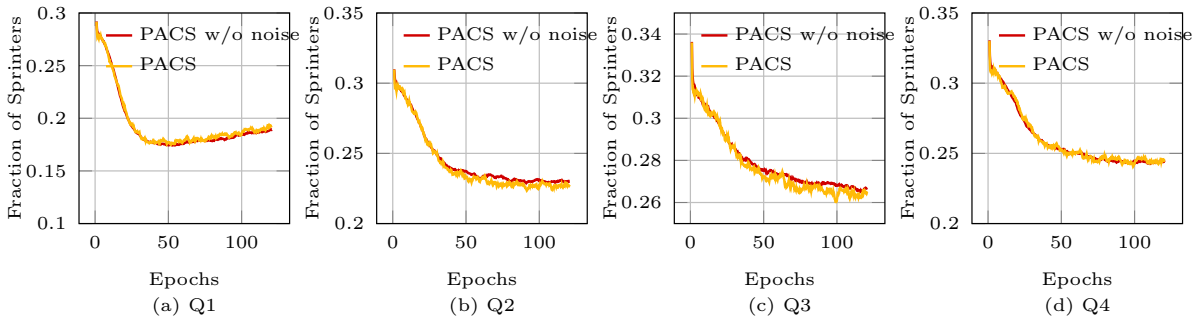


Figure 5.4: Fraction of sprinters under PACS and PACS w/o noise for hybrid applications.

Figure 5.4 shows the fraction of sprinters over time under PACS and PACS w/o noise. As can be seen, the differences between the two curves is negligible.

5.1.3 Real-world Applications

Figure 5.5 compares the performance of different methods for real-world Spark applications. As observed previously, the cooperative threshold method achieves the highest performance. PACS w/o noise outperforms dynamic programming for `PageRank` and `kmeans` and performs nearly as well for the other applications (a difference of less than 2%). Our results further indicate that PACS and PACS w/o noise perform comparably, with the performance difference between them being less than 10% across all applications.

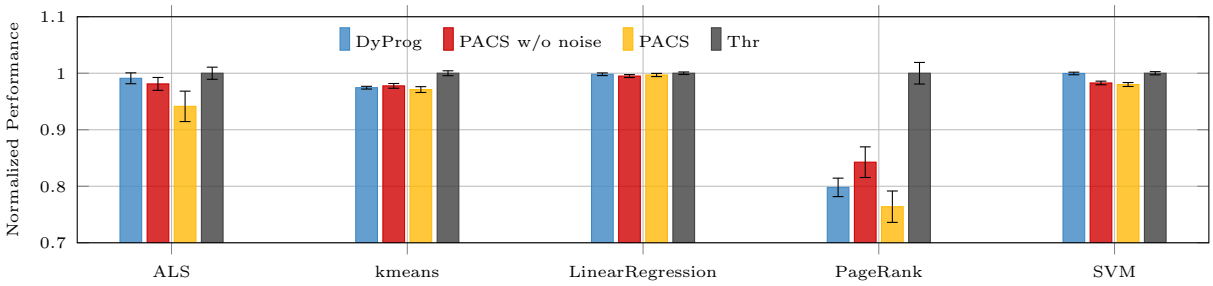


Figure 5.5: Performances of different methods for real-world applications.

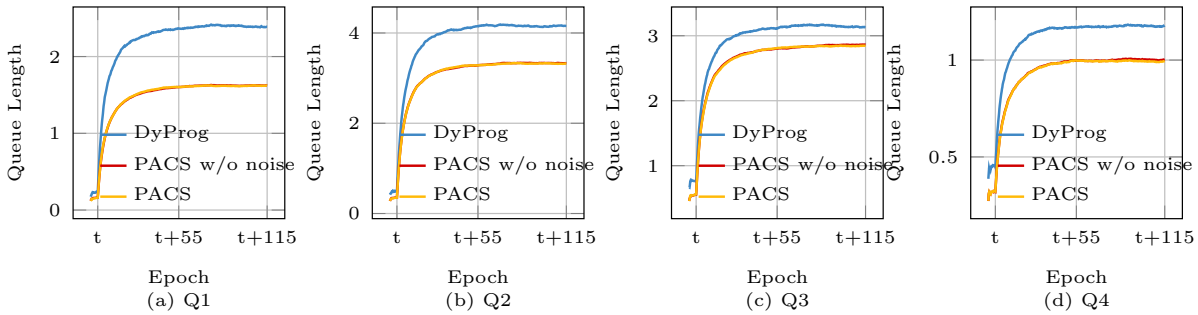


Figure 5.6: Change in queue length when arrival rate increases by 30% at time t.

5.2 Adaptability Analysis

Next, we empirically study the adaptability of PACS when changes occur in system dynamics. Specifically, we use hybrid applications and modify the arrival rate and service rate of inference tasks at some point during the execution of the applications.

5.2.1 Increase in Arrival Rate

We begin by examining the change in the inference queue length when the arrival rates increase. Figure 5.6 compares the average queue length of tenants under PACS w/o noise, PACS, and dynamic programming when arrival rates increase by 30% at time t. As shown, the average queue length increases under all methods immediately following time t. PACS and PACS w/o noise adaptively react to this change in the environment by adjusting their optimal thresholds, achieving a lower average queue length compared to dynamic programming. Once converged, the average queue length under PACS is less than that under dynamic programming by 32%, 20%, 9%, and 14% for Q1-Q4, respectively.

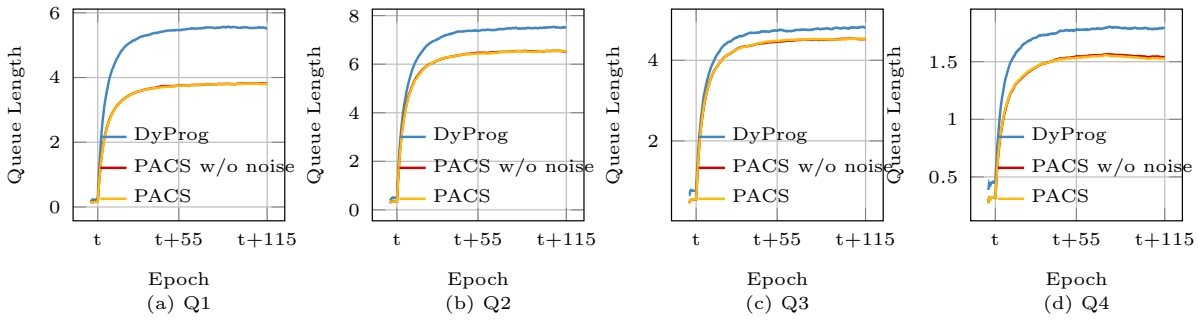


Figure 5.7: Change in queue length when service rate decreases by 33% at time t.

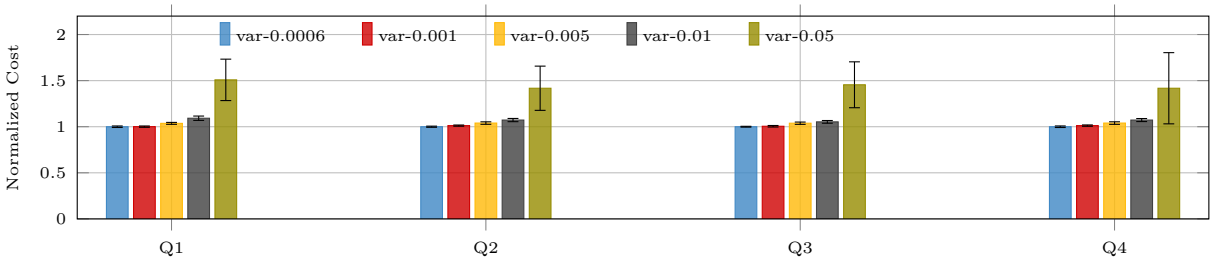


Figure 5.8: Average cost under PACS with varying noise variances.

5.2.2 Decrease in Service Rate

We then explore the change in the average queue length when service rates decrease. Figure 5.7 shows that the average inference queue length increases at time t under all methods when tenants' service rates decrease to two-thirds of their original values. Similar to the previous scenario, PACS adapts to this change and maintains a lower average queue length compared to dynamic programming. Specifically, the average queue length under PACS is 31%, 13%, 6%, and 14% of that under dynamic programming for Q1-Q4, respectively.

5.3 Sensitivity Analysis

Beyond adaptability, we conducted a sensitivity analysis for PACS concerning the magnitude (i.e., variance) of the added noise, where higher variances correspond to higher privacy guarantees. Figure 5.8 compares the performance of hybrid applications under PACS with varying noise variances, ranging from 0.0006 to 0.05. The costs are normalized based on

the cost with a 0.0006 variance. With a variance of 0.05, the cost is about 1.4 times higher compared to the case with a variance of 0.005.

This comparison highlights a clear trade-off between performance and privacy: as the magnitude of variance increases, the performance of PACS decreases and the confidence interval widens. The increase in noise leads to high variations in the values of the fractions of sprinters and the values of incurred taxes from one epoch to another. As the information agents receive becomes less stable, the policies they employ become less reliable for the environment in which they operate. In essence, this results in decision-making that is not optimally aligned with the actual environmental conditions, leading to suboptimal outcomes.

Remark. The results presented in this section highlight the substantial advancements achieved by PACS in comparison to the dynamic programming method proposed in [18]. PACS exhibits significant improvements in scalability, adaptability, system stability, and data privacy. These enhancements contribute to the robustness and confidentiality of multi-tenant interactions, crucial aspects in real-world applications of distributed computational sprinting.

Chapter 6

Related Work

6.1 Computational sprinting for datacenters.

Computational sprinting has been studied for datacenter applications running on CPUs and GPUs. Cai et al. [8] propose a system designed to manage QoS for applications that are sensitive to latency in data centers. This system, underpinned by a robust feedback control mechanism, enables computational sprinting. It does this through precisely scheduling of processor core numbers, setting their operating frequencies, and determining the duration for high-intensity computing. This allows for the efficient handling of workloads with bursty workloads while adhering to QoS standards and operating within thermal limits. It is particularly adept at forecasting upcoming load intensities and then dynamically adjusting computing resources to optimize power usage.

In [72], the authors propose a system designed to manage computational sprinting in data center servers both effectively and with control. It is composed of two distinct power controllers and a power load allocator. This allocator is tasked with deciding the distribution of power loads across various sources. One of the controllers focuses on managing the server's power, particularly adjusting the CPU cores dedicated to batch processing tasks, aiming to enhance efficiency in computing, energy usage, and cost. The other controller, designed for the UPS (Uninterruptible Power Supply), dynamically modulates the rate at which the UPS's energy storage is discharged. This adjustment is crucial for meeting the fluctuating power requirements of interactive workloads while also maintaining power stability and safety.

Ilager et al. [32] address the challenge of high energy consumption in GPUs, which are increasingly utilized in modern computing paradigms like cloud computing for AI/ML

and deep learning tasks. The research focuses on optimizing Dynamic Voltage Frequency Scaling (DVFS), a technique used to reduce GPU power consumption. Recognizing the complexity of establishing optimal clock frequencies—owing to the nonlinear interplay between an application's runtime performance, energy usage, and execution time, and the varied responses of different applications to identical clock settings—the study proposes a novel, data-driven frequency scaling approach. This approach involves collecting performance and energy consumption data from application profiling and then employing this data to train predictive models. These models are designed to accurately forecast power usage and execution time across various clock settings. The flexibility of this solution is highlighted by its applicability to a wide range of workloads and GPU architectures. Additionally, the study introduces a deadline-aware application scheduling algorithm, which leverages the predictive models to minimize energy consumption while adhering to application deadlines.

Morris et al. [45] propose an innovative approach to optimizing computational sprinting in cloud computing environments. It is a model-driven methodology designed to effectively choose the most suitable sprinting policies, with a focus on minimizing response times for query executions. This approach is particularly novel as it navigates the complex interaction between sprinting, queuing, and processing times, which can significantly impact the efficiency of query execution. The authors have developed a comprehensive system that accurately predicts the impact of various sprinting strategies on response times by offline profiling, machine learning, and first-principles simulation. This methodology has been rigorously tested across multiple scenarios, demonstrating its robustness and accuracy.

In this paper, we consider a dynamic method to manage computational sprinting decisions while providing data privacy guarantees.

6.2 Game theory and machine learning for datacenter management.

6.2.1 Game theory for datacenter management.

Game theory has been proven effective in datacenter resource management. [11] proposed a novel architecture for resource management in internet hosting centers, with a specific focus on energy efficiency for large server clusters. This architecture is adeptly designed to enable hosting centers to automatically adapt their server resources to fluctuating loads, enhance energy efficiency by dynamically adjusting the active server set, and respond effectively to

power disruptions or thermal events in line with Service Level Agreements (SLAs). Central to their system is an innovative economic approach, where services 'bid' for resources based on the performance delivered, essentially linking resource allocation to service performance. The system is engineered to continuously monitor load and strategically plan resource distribution, using a resource allocation algorithm that adjusts resource prices to maintain a balance between supply and demand, thereby optimizing resource utilization. Additionally, the architecture includes a reconfigurable server switching infrastructure that efficiently directs request traffic to designated servers. This study presents a significant leap forward in resource management strategies for hosting centers, prioritizing energy efficiency and service adaptability.

[21] address the complex problem of fair resource allocation within systems that encompass diverse resource types and varying user demands. To effectively address this challenge, the authors introduce an innovative concept named Dominant Resource Fairness (DRF), which is an extension of the max-min fairness principle, adapted for environments with multiple types of resources. The distinctiveness of DRF lies in its adherence to several crucial and desirable properties. Firstly, it encourages resource sharing among users by guaranteeing that no user benefits disproportionately from an equal resource partition. Secondly, DRF is designed to be strategy-proof, preventing users from manipulating their resource allocation by misrepresenting their actual requirements. Thirdly, the system ensures that it is envy-free, meaning no user would feel the need to exchange her allocation with another, indicating a sense of equitable distribution. Lastly, the allocations under DRF are Pareto efficient, ensuring that any improvement in a user's allocation doesn't come at the cost of another's. The practical applicability of DRF is demonstrated through its implementation in the Mesos cluster resource manager. This study presents a significant advancement in the field of resource allocation, offering a fair, efficient, and user-centric approach.

[58] propose a new strategy for survivable virtual network mapping within a Cloud's backbone. It aims to enhance the Cloud Provider's revenue while effectively managing physical failures of routers and links. To navigate around the exponential complexity typically associated with network mapping, the authors introduce a novel reliable embedding strategy known as CG-VNE, which is grounded in the coordination game framework. This strategy involves formulating the problem as two complex coordination games. The first game focuses on the mapping of virtual routers, considering the interdependencies of each router's actions on the mapping of its connected virtual links. Consequently, the second game is initiated to embed these virtual links, with both games engaging fictitious players who collaborate to achieve a Nash Equilibrium. This equilibrium not only exists but also aligns with a social optimum. The ultimate goal of CG-VNE is to maximize the Cloud

provider's revenue by simultaneously increasing the acceptance rate of client requests and minimizing the rate of virtual network failures or outages which are due to issues in the underlying physical network infrastructure. This study presents a significant breakthrough in cloud networking, offering a practical and efficient solution for enhancing network survivability and provider profitability.

[69] focuses on the strategic manipulation of resource allocation within these data centers. The authors observed that even in a company as large as Google, employees engaged in a strategic manipulation of resource requests. Some of them inflating their needs to minimize sharing, while others deflated their resource requests to pretend that their tasks could easily fit within any computer. Once their tasks were loaded onto a computer, these operations would then consume all of the machine's available resources, leaving no room for the tasks of their colleagues. Highlighting the seriousness of this issue, the authors point out the significant energy consumption of data centers globally. The study sheds light on the problem of wasted energy, especially considering that idle servers can dissipate up to 50% of the power they consume at peak capacity, a critical concern given that a typical user's task utilizes only 20 to 30 percent of a server's potential.

6.2.2 Machine learning for datacenter management.

Machine learning techniques are shown to be effective tools for bolstering energy efficiency. [26] introduce a strategy for dynamic virtual machine (VM) consolidation for reducing energy consumption in large-scale datacenters. It combines centralized and distributed approaches to optimize power usage. This strategy employs a distributed, multi-agent ML technique to determine the most efficient power mode and operating frequency for each server in real-time. Concurrently, it utilizes a centralized heuristic algorithm for the strategic migration of VMs to the most appropriate hosts.

Due to the rapid expansion of cloud computing, [62] propose a solution to address the pressing issue of increasing brown energy consumption and carbon emissions in cloud datacenters. It aims to explore strategies for efficiently matching renewable energy generators with datacenters from various cloud providers to reduce carbon emissions, minimize costs, and adhere to SLOs despite the challenges of renewable energy shortage. This task is complicated because of the competition of energy requests among datacenters, the unpredictability of renewable energy generation, and the need for rapid decision-making. The authors first evaluate several ML techniques for their long-term prediction accuracy regarding the real-world data of renewable energy generation and datacenter energy demand, and find SARIMA model has the best performance. Then, they propose a MARL strategy for

individual datacenters to optimize their requests for renewable energy from specific generators. Moreover, they introduce a deadline-guaranteed job postponement method to manage non-urgent tasks during periods of insufficient renewable energy supply.

[12] introduces a model-free deep RL based approach for joint optimization, designed to improve the cooperation between IT operations and cooling mechanims. To address the problem with high-dimensional state space and the extensive hybrid discrete-continuous action space, the authors propose a hybrid AC-DDPG multi-agent structure. This structure is stabilized through the introduction of a scheduling baseline comparison method. They design an asynchronous control optimization algorithm to address the disparities in response times between IT and cooling systems.

Moreover, machine learning techniques are also shown to be effective tools for facilitating effective workload balancing. [67] propose a MARL approach to address the network load balancing problem, which is a task characterized by its complexity and real-world applicability challenges. They identify traditional heuristic solution as being inflexible to changes in workload distributions and arrival rates. So, they frame the network load balancing issue as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) and introduce MARL as a solution. They train and test their methods on an emulation system that simulates realistic, moderate-to-large-scale network environments. The results highlight the superiority of the MARL in achieving effective load balancing across different scenarios, comparing to traditional strategies.

[70] propose a MARL framework for the efficient scheduling of distributed DL jobs across large GPU clusters. The author find that there is a challenge of server sharing and the interference it causes among co-located DL jobs. Their approach aims to improve resource utilization and minimizing job completion time (JCT) at the same time. To navigate the complexities of job placement in such large-scale environments, the framework incorporates hierarchical graph neural networks. These networks are designed to encode the datacenter topology and server architecture, facilitating topology-aware job placements that are sensitive to the nuances of the physical and network infrastructure of the datacenter. Moreover, the authors introduce a job interference model to address the limited precise reward samples for different job placements. Unlike traditional methods that rely on either explicit interference modeling or black-box schedulers using RL, this framework leverages multiple schedulers to manage the workload in vast clusters containing thousands of GPU servers.

[39] propose an approach to computation offloading in IoT edge computing networks to address the multiple users competing for limited resources. The authors introduce a mechanism that formulates the offloading process as a stochastic game, where each user

operates as learning agent. They design a MARL framework to solve this game. Within this framework, they introduce the Independent Learners based Milti-Agent Q-Learning (IL-based MA-Q) algorithm. This algorithm enables efficient and energy-saving computation offloading decisions without need for extra channel estimation efforts at the centralized gateway.

[2] propose a fully decentralized load-balacing framework to manage the distribution of workloads across servers in dense racks for microsecond-scale workloads. The authors design a fully decentralized load-balancing framework where servers collectively balance the load in the system. It has been modeled as a cooperative stochastic game, and the authors implement a decentralized algorithm based on multi-agent-learning theory to find the Nash Equilibrium.

In this paper, we utilized a multi-agent reinforcement learning technique to design an adaptive mechanism that can make optimal sprinting decisions without requiring any prior information about the system dynamics.

6.3 Data Privacy for datacenters.

Data privacy is a notable problem in multi-tenant environments, such as cloud computing and datacenter networks. [63] propose a cloud-based framework designed to deploy DNNs on mobile devices efficiently while addressing the privacy concerns asociated with the cloud computing. It intelligently partitions DNN tasks between mobile devices and cloud datacenters. The framework performs simple data transformations on the device side and offloads the computationally intensive training and complex inference tasks to the cloud. Also, it integrates a lightweight, privacy-preserving mechanism that employs arbitrary data removement and random noise addition, offering a robust privacy gaurantee through a comprehensive privacy budget analysis. The authors introduce a noisy training method to enhance the resilience of the cloud-side network to perturbed data, such that both privacy and inference performance are ensured.

[73] propose a method for optimizing process mapping in geo-distributed cloud datacenters. It is used to address the challenge posed by non-uniform communication costs, multi-level data privacy requirements, heterogeneous network performance, and the potential for process failures. The authors introduce an optimization problem formulation that incorporates special privacy requirements specific to geo-distributed data centers. Their method aims to efficiently identify optimal or near-optimal solutions for mapping parallel processes to physical nodes, taking into account the complex constraints of these environments.

[14] propose a differential privacy-based query model specifically designed for sustainable fog computing-supported datacenters. The privacy challenges in fog computing is from device heterogeneity and the need for low-latency, secure data processing. The authors introduce a way to qualify and ensure privacy preservation through rigirous mathematical proofs to solve the privacy challenges in fog computing. The core of their approach involves capturing the structural information of datacenters supported by sustainable fog computing and mapping the datasets involved in query results to real vectors. To protect data privacy, they implement differential privacy by injecting Laplace noise into the data.

In this paper, we implement differential privacy in PACS with Gaussian mechanism as well. We sample noise from Gaussian distribution and add it into the fraction of sprinters to avoid data leaking. For a given privacy budget, Gaussian noise can result in more accurate query results, especially when the function/query has low sensitivity or when a (ϵ, δ)-differential privacy level is acceptable. Additionally, the Gaussian mechanism's adaptability to (ϵ, δ)-differential privacy can be advantageous when a slightly relaxed privacy guarantee is acceptable for a significant gain in accuracy.

Chapter 7

Conclusion

In this paper, we introduced PACS, a computational-sprinting framework designed to address the limitations of existing approaches through adaptability and enhanced data privacy, all without requiring prior knowledge of system dynamics. Within PACS, the datacenter provider plays a critical role in monitoring tenants' power consumption, implementing a system where the cost of sprinting actions, "taxes", is based on the cumulative power usage of the tenants. This mechanism incentivizes tenants to utilize a multi-agent reinforcement learning algorithm to fine-tune their sprinting strategy, taking into account both system dynamics and the costs associated with increased power consumption during sprinting periods.

Experimental results demonstrate that PACS achieves comparable performance to state-of-the-art methods, outperforming them for some applications while incurring at most a 10% performance degradation for others. Interpreting these results, it's clear that the core of PACS's efficiency lies in its innovative approach to managing the cost of sprinting actions. The multi-agent reinforcement learning algorithm deployed by tenants doesn't merely aim to optimize power usage; it strategically navigates the cost-benefit landscape of sprinting actions. These costs serve a similar role by applying a "charge" to the computational resources used during sprinting. This encourages a more judicious use of sprinting, ensuring that tenants optimize their use of resources in a manner that balances immediate computational demands with the overarching goal of minimizing the associated costs.

Furthermore, the slight performance degradation noted in some scenarios illuminates the algorithm's capacity to prioritize long-term efficiency and cost-effectiveness over short-term performance peaks. This trade-off underscores the sophistication of PACS's adaptive strategies. It shows how dynamically the system responds to computational sprinting chal-

lenges. Importantly, it emphasizes the need to consider the cost implications of sprinting actions. These considerations are crucial within the broader context of system performance and resource management.

In summary, PACS emerges as a forward-thinking framework that adeptly balances performance, privacy, and the strategic management of sprinting costs. Its nuanced approach to incorporating the costs of sprinting actions into the operational decision-making process represents a significant advancement in the field of computational sprinting, promising avenues for further research and application.

References

[1] ONNX: Open neural network exchange. https://github.com/onnx/onnx.

[2] Ali Hossein Abbasi Abyaneh, Maizi Liao, and Seyed Majid Zahedi. Malcolm: Multi-agent learning for cooperative load management at rack scale. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, 6(3):1–25, 2022.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 2623–2631, 2019.

[4] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

[5] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The Rocket Chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, 2016.

[6] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019.

[7] Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[8] Haoran Cai, Qiang Cao, Feng Sheng, Yang Yang, Changsheng Xie, and Liang Xiao. ESprint: QoS-aware management for effective computational sprinting in data centers. In *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 420–429, 2019.

[9] Haoran Cai, Xu Zhou, Qiang Cao, Hong Jiang, Feng Sheng, Xiandong Qi, Jie Yao, Changsheng Xie, Liang Xiao, and Liang Gu. Greensprint: Effective computational sprinting in green data centers. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 690–699, 2018.

[10] Peter E Caines, Minyi Huang, and Roland P Malhamé. Large population stochastic dynamic games: Closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle. *Communications in Information and Systems*, 6(3):221–252, 2006.

[11] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, 2001.

[12] Ce Chi, Kaixuan Ji, Avinab Marahatta, Penglei Song, Fa Zhang, and Zhiyong Liu. Jointly optimizing the IT and cooling systems for data center energy efficiency based on multi-agent deep reinforcement learning. In *Proceedings of the 11th ACM International Conference on Future Energy Systems (e-Energy)*, pages 489–495, 2020.

[13] Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425, 2020.

[14] Miao Du, Kun Wang, Xiulong Liu, Song Guo, and Yan Zhang. A differential privacy-based query model for sustainable fog data centers. *IEEE Transactions on Sustainable computing*, 4(2):145–155, 2017.

[15] Cynthia Dwork. Differential privacy. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1–12, 2006.

[16] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2014.

[17] Richard Evans and Jim Gao. DeepMind AI reduces Google data centre cooling bill by 40%. https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/, 2016.

[18] Songchun Fan, Seyed Majid Zahedi, and Benjamin C. Lee. The computational sprinting game. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, page 561–575, 2016.

[19] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, pages 13–23, 2007.

[20] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774, 2021.

[21] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *In proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[22] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *Proceedings of the 38th annual international symposium on Computer architecture*, pages 341–352, 2011.

[23] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 75–86, 2012.

[24] Yinghao Guo, Rui Zhao, Shiwei Lai, Lisheng Fan, Xianfu Lei, and George K Karagiannidis. Distributed machine learning for multiuser mobile edge computing systems. *IEEE Journal of Selected Topics in Signal Processing*, 16(3):460–473, 2022.

[25] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Proceedings of the Autonomous Agents and Multiagent Systems (AAMAS)*, pages 66–83, 2017.

[26] Kawsar Haghshenas, Ali Pahlevan, Marina Zapater, Siamak Mohammadi, and David Atienza. Magnetic: Multi-agent machine learning-based approach for energy efficient dynamic consolidation in data centers. *IEEE Transactions on Services Computing*, 15(1):30–44, 2019.

[27] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4):1–19, 2015.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[30] Ziqiang Huang, José A Joao, Alejandro Rico, Andrew D Hilton, and Benjamin C Lee. DynaSprint: Microarchitectural sprints with dynamic utility and thermal management. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 426–439, 2019.

[31] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[32] Shashikant Ilager, Rajeev Muralidhar, Kotagiri Rammohanrao, and Rajkumar Buyya. A data-driven frequency scaling approach for deadline-aware energy efficient scheduling on graphics processing units (GPUs). In *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 579–588, 2020.

[33] Majid Jalili, Ioannis Manousakis, Íñigo Goiri, Pulkit A. Misra, Ashish Raniwala, Husam Alissa, Bharath Ramakrishnan, Phillip Tuma, Christian Belady, Marcus Fontoura, and Ricardo Bianchini. Cost-efficient overclocking in immersion-cooled datacenters. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 623–636, 2021.

[34] Lasry Jean-Michel and Lions Pierre-Louis. Mean field games. *Japanese Journal of Mathematics*, 2(1):229–260, 2007.

[35] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42, 2018.

[36] Joe Kava. Better data centers through machine learning. https://blog.google/inside-google/infrastructure/better-data-centers-through-machine/, 2014.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[38] Moshe Lichman et al. UCI machine learning repository, 2013.

[39] Xiaolan Liu, Jiadong Yu, Zhiyong Feng, and Yue Gao. Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing. *China Communications*, 17(9):220–236, 2020.

[40] Xiaolan Liu, Jiadong Yu, Jian Wang, and Yue Gao. Resource allocation with edge computing in IoT networks via machine learning. *IEEE Internet of Things Journal*, 7(4):3415–3426, 2020.

[41] Diptyaroop Maji, Prashant Shenoy, and Ramesh K Sitaraman. CarbonCast: multi-day forecasting of grid carbon intensity. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 198–207, 2022.

[42] Diptyaroop Maji, Ramesh K Sitaraman, and Prashant Shenoy. DACF: Day-ahead carbon intensity forecasting of power grids using machine learning. In *Proceedings of the Thirteenth ACM International Conference on Future Energy Systems*, pages 188–192, 2022.

[43] Robert Meusel, Oliver Lehmberg, Christian Bizer, and Sebastiano Vigna. Web Data Commons - Hyperlink Graphs. http://webdatacommons.org/hyperlinkgraph, 2012. Online; accessed: 12-29-2016.

[44] Ilya Mironov. Rényi differential privacy. In *Proceedings of the 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017.

[45] Nathaniel Morris, Christopher Stewart, Lydia Chen, Robert Birke, and Jaimie Kelley. Model-driven computational sprinting. In *Proceedings of the 13th EuroSys Conference*, page 13, 2018.

[46] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. BatchSizer: Power-performance trade-off for DNN inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 819–824, 2021.

[47] Samrat Nath and Jingxian Wu. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems. *Intelligent and Converged Networks*, 1(2):181–198, 2020.

[48] Ana Radovanovic. Our data centers now work harder when the sun shines and wind blows. https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows/, 2020.

[49] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin Pipe, Thomas Wenisch, and Milo Martin. Utilizing dark silicon to save energy with computational sprinting. *IEEE Micro*, 33(5):20–28, 2013.

[50] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P Pipe, Thomas F Wenisch, and Milo MK Martin. Computational sprinting on a hardware/software testbed. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 155–166, 2013.

[51] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. Computational sprinting. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2012.

[52] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–562, 1961.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[54] Varun Sakalkar, Vasileios Kontorinis, David Landhuis, Shaohong Li, Darren De Ronde, Thomas Blooming, Anand Ramesh, James Kennedy, Christopher Malone, Jimmy Clidaras, et al. Data center power oversubscription with a medium voltage power plane and priority-aware capping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 497–511, 2020.

[55] Lei Shao, Arun Raghavan, Laurel Emurian, Marios C Papaefthymiou, Thomas F Wenisch, Milo MK Martin, and Kevin P Pipe. On-chip phase change heat sinks

designed for computational sprinting. In *Proceedings of the IEEE Symposium on Semiconductor Thermal Measurement and Management (SEMI-THERM)*, pages 29–34, 2014.

[56] Matt Skach, Manish Arora, Chang-Hong Hsu, Qi Li, Dean Tullsen, Lingjia Tang, and Jason Mars. Thermal time shifting: Leveraging phase change materials to reduce cooling costs in warehouse-scale computers. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 439–449, 2015.

[57] Matthew Skach. *Thermal Energy Storage for Datacenters with Phase Change Materials*. PhD thesis, 2018.

[58] Oussama Soualah, Ilhem Fajjari, Nadjib Aitsaadi, and Abdelhamid Mellouk. A reliable virtual network embedding algorithm based on game theory within cloud's backbone. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 2975–2981, 2014.

[59] J Stamper, A Niculescu-Mizil, S Ritter, GJ Gordon, and KR Koedinger. Algebra I 2008-2009. Challenge data set from KDD Cup 2010 educational data mining challenge, 2010.

[60] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[61] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *Proceedings of the 10th ACM International Conference on Future Energy Systems (e-Energy)*, pages 315–325, 2019.

[62] Haoyu Wang, Haiying Shen, Jiechao Gao, Kevin Zheng, and Xiaoying Li. Multi-agent reinforcement learning based distributed renewable energy matching for datacenters. In *Proceedings of the 50th International Conference on Parallel Processing (ICPP)*, pages 1–10, 2021.

[63] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2407–2416, 2018.

[64] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. Resource allocation based on deep reinforcement learning in IoT edge computing. *IEEE Journal on Selected Areas in Communications*, 38(6):1133–1146, 2020.

[65] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5571–5580, 2018.

[66] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

[67] Zhiyuan Yao, Zihan Ding, and Thomas Clausen. Multi-agent reinforcement learning for network load balancing in data center. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, pages 3594–3603, 2022.

[68] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

[69] Seyed Majid Zahedi and Benjamin C. Lee. A win for game theory in the data center. *IEEE Spectrum*, 57(4):40–44, 2020.

[70] Xiaoyang Zhao and Chuan Wu. Large-scale machine learning cluster scheduling via multi-agent graph reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(4):4962–4974, 2022.

[71] Wenli Zheng and Xiaorui Wang. Data center sprinting: Enabling computational sprinting at the data center level. In *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 175–184, 2015.

[72] Wenli Zheng, Xiaorui Wang, Yue Ma, Chao Li, Hao Lin, Bin Yao, Jianfeng Zhang, and Minyi Guo. SprintCon: Controllable and efficient computational sprinting for data center servers. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 815–824, 2019.

[73] Amelie Chi Zhou, Yao Xiao, Yifan Gong, Bingsheng He, Jidong Zhai, and Rui Mao. Privacy regulation aware process mapping in geo-distributed cloud data centers. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1872–1888, 2019.