IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 21, 2021, accepted January 9, 2022, date of publication January 25, 2022, date of current version February 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3145955

NASCENT: A Non-Invasive Solution for Detecting Utilization of Servers in Bare-Metal Cloud

MARUTHI SESHIDHAR INUKONDA[1], (Member, IEEE), ATHARVA RAJENDRA KARPATE[2], BHEEMARJUNA REDDY TAMMA[1], (Senior Member, IEEE), SPARSH MITTAL[3,4], (Senior Member, IEEE), AND PRAVEEN TAMMANA[1]

[1]Department of Computer Science and Engineering, IIT Hyderabad, Kandi 502285, India
[2]Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA
[3]Department of Electronics and Communication Engineering, IIT Roorkee, Roorkee 247667, India
[4]Mehta Family School of Data Science and Artificial Intelligence, IIT Roorkee, Roorkee 247667, India

Corresponding author: Maruthi Seshidhar Inukonda (cs18resch01001@iith.ac.in)

ABSTRACT Physical servers are available as-a-service in bare-metal public and private cloud platforms, and their demand has been proliferating because of the high levels of privacy and security guarantees they provide to the tenants. This raises the need for efficient management of bare-metal clouds to keep operational costs low such as by reducing energy consumption. For efficiently managing the cloud infrastructure, bare-metal cloud operators need to monitor the utilization of servers. However, the privacy and security concerns prohibit the installation of third-party monitoring agents on the servers; thus, finding the server-utilization becomes a challenge. In this work, we present NASCENT, a scalable machine-learning (ML) based non-invasive solution for finding the utilization of servers without compromising the privacy and security of bare-metal cloud tenants. Our key idea is to infer utilization from various sensor readings accessible via a server's baseboard management controller (BMC) hardware. We evaluate the proposed solution with three regression based supervised ML algorithms in a Bare-metal-as-a-service (BMaaS) cloud. Our experimental evaluation shows that one of the ML algorithms employed in NASCENT infers the utilization with a root-mean-square error (RMSE) between 2.9 to 9.3 for different workloads. Also, the proposed solution uses minimal memory resources (19 KB) and can even run on BMC hardware which has very limited memory. We also propose a BMaaS cloud architecture that seamlessly integrates automated training and deployment of the ML algorithm in our solution into the life-cycle of bare-metal servers. NASCENT's codebase can be found at https://github.com/iithcandle/dhi-ojas

INDEX TERMS Bare-metal server, BMaaS cloud, embedded systems, machine learning, privacy, power consumption, server utilization.

I. INTRODUCTION

Demand for bare-metal servers in public and private cloud platforms is increasing as they give high guarantees of data privacy and security [1], especially because servers' tenants do not share them with others using virtualization techniques. Many cloud providers [2]–[9] and research institutes manage such bare-metal servers on behalf of their tenants (*e.g.*, customers, departments and labs) using a "Bare-metal-as-a-service" (BMaaS) cloud orchestration software such as Canonical's MAAS [10] and Openstack's Ironic [11]. To satisfy the diverse computing requirements of the tenants, a BMaaS cloud usually contains bare-metal servers with diverse hardware configurations.

Figure 1 shows a typical bare-metal server with two modules. The module labeled host complex contains the server's computing hardware running the host operating system. Moreover, the module labeled BMC complex (baseboard management controller [12], [13]) is an embedded hardware running BMC firmware. The BMC firmware provides power control, sensor readings, and console services, all remotely accessed through a dedicated ethernet port. Typically, BMC hardware is shipped with either a proprietary firmware [14]–[16] or an open-source firmware such as OpenBMC [17], [18]. Cloud operators or server owners cannot modify the firmware if it is proprietary. On the other hand,

The associate editor coordinating the review of this manuscript and approving it for publication was Jerry Chun-Wei Lin.

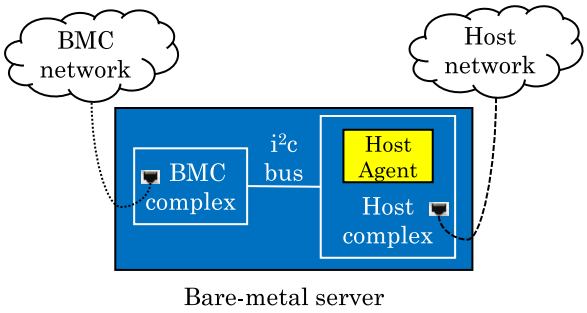

FIGURE 1. A bare-metal server used in BMaaS cloud.

OpenBMC provides flexibility to build operator-defined functionality in the BMC complex.

Efficient utilization of cloud resources in a BMaaS cloud requires optimal allocation of available compute (servers) among their tenants. To realize this goal, knowing the current CPU utilization of individual servers is crucial. To be specific, many cloud management applications benefit from CPU utilization information. Some of these applications are:

- energy management applications can turn off servers with zero or low utilization
- load-balancing applications can distribute tasks among servers based on their CPU utilization
- security applications can detect threats based on abnormally high CPU utilization
- a deadlock in parallel programming or I/O-intensive task can be detected if the CPU utilization is low
- a server's "health" or "age" can be estimated based on CPU utilization information gathered over a period of time.

Evidently, measuring the CPU utilization metric is highly important in efficient management of BMaaS cloud infrastructure.

Typically, in the Infrastructure-as-a-Service (IaaS) clouds (where virtual machines are created), host agents [19], [20] are installed in the host complex or hypervisor of the servers, and they periodically collect the CPU utilization statistics. However, no agents are installed inside the VMs to respect the privacy and security of tenants. However, BMaaS cloud operators [21] do not install any agents even in the host complex as these agents can be misused to collect sensitive user, tenant or system-related information, violating security and privacy goals promised to the tenants. Few server models, with additional licensing costs, offer BMC firmware-level functionality [22]–[25] that provides CPU utilization information without using host agents. Many other server models do not offer this firmware-level functionality, and it requires enhancements to proprietary BMC firmware. A solution for inferring CPU utilization of heterogeneous bare-metal servers in a non-invasive manner using BMC sensor readings but not requiring additional enhancements to proprietary BMC firmware is of high interest for meeting the security and privacy requirements of the tenants.

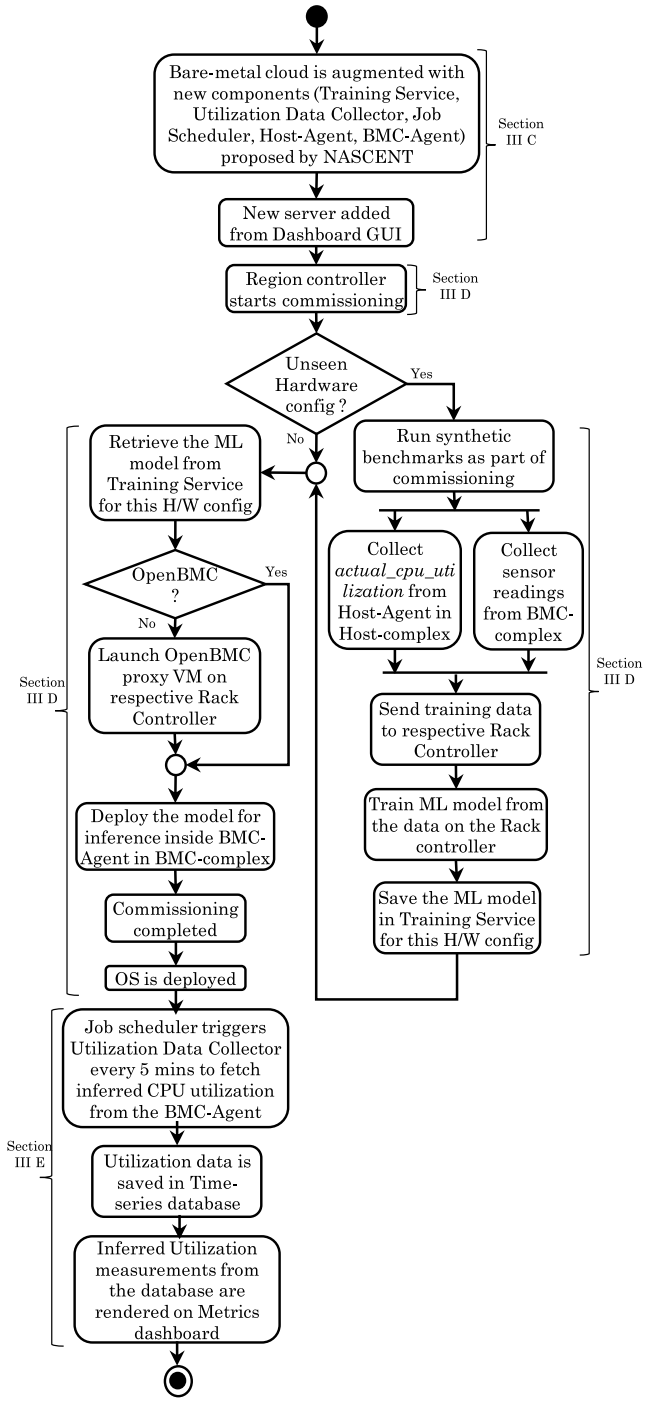

FIGURE 2. Overall flow of activities in NASCENT.

In this paper, first we have experimented with a simple power consumption-based heuristic to infer the CPU utilization of servers and found it inadequate due to the overlapping power consumption ranges. We then propose NASCENT, a non-invasive machine learning (ML) based solution, to infer a server's CPU utilization with high accuracy from its BMC's sensor readings (viz., `overall_power`, `cpu_power`, `systemboard_power`, `fan_power`, `hdd_power`, etc). We have evaluated three popular supervised machine

learning algorithms: K-Nearest Neighbor regression (KNNR), support vector regression (SVR) with polynomial function (SVRPOL), and support vector regression with radial basis function (SVRRBF) in NASCENT. NASCENT is designed to work with both OpenBMC hardware [17] and a proprietary BMC hardware.

While provisioning servers, changes can be done to the hardware configuration of bare-metal servers as per users' requests. Few minor changes to the hardware configuration can significantly change the range of power consumption sensor readings. Consequently, keeping the ML models up-to-date with frequent changes to hardware configuration at the cloud scale is challenging. In this paper, we also propose a BMaaS cloud architecture that seamlessly integrates automated training and deployment of the ML models into life-cycle of bare-metal servers. We propose augmentation of five components to BMaaS cloud architecture (viz., Training Service, Utilization Data Collector, Job Scheduler, BMC-Agents and Host-Agents). Figure 2 shows overall flow of activities in NASCENT with references to relevant sections in this paper.

The key contributions are as follows:

- **We propose NASCENT, a solution for finding server CPU utilization in a non-invasive manner.** Our approach is to train models with data of server hardware configuration and its BMC sensor readings using regression-based supervised ML algorithms, and then use the trained models to infer CPU utilization.
- We also propose a BMaaS cloud architecture for automated ML model training and deployment and did a proof-of-concept (PoC) implementation on four server in a BMaaS cloud datacenter.
- We evaluate NASCENT on both synthetically created dataset and production dataset and demonstrate its effectiveness in detecting CPU utilization with low prediction error.
- The code base of NASCENT is open sourced and available at `https://github.com/iithcandle/dhi-ojas`

The rest of the paper is organized as follows: We first discuss the related work (§II). We then present the system architecture of NASCENT, along with the machine learning aspect of NASCENT (§III). We then present the experimental results (§IV) and finally conclude the work (§V).

II. RELATED WORK

This section gives an overview of two existing techniques for utilization detection in a bare-metal cloud. It then presents the related work on OpenBMC that NASCENT embraces. The section then provides an overview of some related work in (Infrastructure-as-a-Service) IaaS clouds and power prediction from utilization. The section ends with related works in applications of utilization detection.

A. USING HOST-COMPLEX MONITORING AGENTS

One of the most popular ways to collect utilization statistics such as CPU, memory, and disk utilization is via installing agents (also known as host agents) on the host complex. Volz *et al.* [19], [20] propose installing host agents on servers and periodically collecting utilization statistics. These statistics are stored in a time-series database. However, in bare-metal clouds, the tenants may not authorize cloud operators [21] to install any agents in the host complex of the server as these agents might collect sensitive user or system-related information, violating security and privacy policies.

B. USING PROPRIETARY BMC FIRMWARE FUNCTIONALITY

In-processor firmware functionalities such as Intel's management engine (ME), AMD's platform security processor (PSP), and IBM's on-chip controller (OCC) provide hardware-level utilization in an OS-agnostic manner. Proprietary BMC firmware functionality such as compute usage per second (CUPS) [22]–[25] uses these in-processor firmware functionalities without involving host-complex operating system. Hence, CUPS meets privacy and security requirements. However, to use CUPS functionality, cloud operators have to buy firmware licenses and upgrade the existing deployments. This increases operational costs significantly and may not be suitable for the existing deployments.

Moreover, it allows measuring only the processor utilization. A similar functionality needs to be enabled by all the power-hungry hardware accelerators (*e.g.*, GPUs, smart NICs) to find the overall server utilization. This, however, further adds to the operational overheads. While designing NASCENT, we consider possible extensions to detect utilization of these accelerators.

C. OPEN COMPUTE PROJECT AND OPENBMC

Typically, BMC hardware runs a proprietary firmware that cloud operators or server owners cannot modify. To overcome vendor lock-in, OpenBMC [17], [18] has become a popular open-source alternative as it provides the necessary flexibility to build user-defined functionality in the BMC-complexes for a cloud. Today ''Open Compute Project'' [26] compliant bare-metal servers and ''OpenPower Servers'' [27] are being shipped with BMC hardware that can run OpenBMC firmware. BMCLeech [28] is a method for capturing a snapshot of bare-metal server's host complex memory from the BMC complex (running OpenBMC) in a stealthy manner for forensic investigation. The proposed NASCENT is designed to run on OpenBMC and also on proprietary BMC hardware using an OpenBMC proxy.

D. PRIVACY-PRESERVING METHODS IN IAAS CLOUD

Previous work on IaaS clouds by Borgetto *et al.* [29], Verma *et al.* [30], Lin *et al.* [31], Oh *et al.* [32], Hat *et al.* [33], and Arzani *et al.* [34] run special monitoring agents at the hypervisor level (*i.e.*, host-complex). This is used to perform

live-migration of VMs to fewer servers, power-off unused servers for energy management, or detect compromised VMs. Running monitoring agents at the hypervisor level preserves the users' privacy in the case of IaaS clouds (VMs), because monitoring agents do not run inside the VMs. However, in BMaas cloud, only tenants have complete control of the host-complex of a bare-metal server. However, BMaaS cloud operators [21] cannot install any agents even in the host complex as these agents can be misused to collect sensitive users, tenants or system-related information, violating security and privacy goals promised to the tenants. Hence, they cannot adopt these solutions to BMaaS clouds in practice.

E. INFERRING POWER CONSUMPTION FROM UTILIZATION

Our work seeks to infer CPU utilization from the power consumption data, whereas many other works [35]–[37] aim to do the opposite, that is, infer power consumption from the CPU utilization. The previous research works use a linear model of the form `Power = a × Utilization+c`, where a and c are constants. However, through experiments, we observe that CPU utilization does not have a linear relationship with power consumption (refer §IV-D).

Dhiman *et al.*, [38] present a multi-variate "Gaussian mixture vector quantization" model to predict active power. Note that this is in contrast to our objective of predicting utilization. Their key idea is to build power prediction models using multiple metrics such as IPC (instructions-per-cycle), MPC (memory accesses per cycle), CTPC (cache transactions per cycle), and CPU utilization metrics. One can think of using the same metrics to detect utilization. However, these program-level metrics are not available without installing monitoring agents at the host complex.

III. NASCENT: SYSTEM ARCHITECTURE

In this section, we first define key requirements (§III-A) to be met by any solution for finding the CPU utilization in BMaaS cloud, followed by an overview of NASCENT architecture (§III-B). We then discuss the modifications to BMaaS architecture with NASCENT (§III-C) followed by the training (§III-D) and inference (§III-E) phases in detail. We then discuss feature vector (§III-F) and then discuss the ML models used (§III-G). Finally, we discuss the flow of inference and training phases (§III-H).

A. DESIGN REQUIREMENTS

The following design requirements are derived based on the practical constraints and potential use-cases of utilization detection module observed both in the research literature and in our private cloud.

1) R1: MEET SECURITY CONSTRAINTS

The solution should satisfy the privacy and security requirements of cloud tenants.

TABLE 1. Design requirements met by existing approaches and proposed NASCENT solution.

Approach	R1	R2	R3	R4
Host monitoring agents [20]	✘	✔	✔	N/A
Proprietary BMC functionality [14]–[16]	✔	✘	✔	✔
Proposed NASCENT Solution	✔	✔	✔	✔

2) R2: UTILIZATION DETECTION SHOULD BE GENERALIZED

A utilization detection technique should be generalized for heterogeneous servers in BMaaS clouds having a diverse set of CPU models.

3) R3: ABILITY TO DETECT UTILIZATION AT A FINER GRANULARITY

This ability enables a wide variety of cloud management applications (*e.g.*., load balance, energy management.)

4) R4: FAST AND SCALABLE UTILIZATION DETECTION

Utilization detection should be fast and consume minimal resources (*i.e.*, compute, storage, and network), so that it enables taking quick actions and scales well for large clouds.

In the following section, we propose NASCENT and explain how it meets all the requirements R1-R4. Table 1 summarizes the design requirements met by various solutions.

B. OVERVIEW OF NASCENT

This work focuses on designing a solution to detect CPU utilization for CPU-bound workloads. The key idea is to leverage the server's BMC sensor readings provided by the BMC-complex to meet R1-R4 requirements. Note that in the CPU-bound workloads, the usage of the CPU is much higher than that of memory, disk, and network resources. To realize this idea, we design NASCENT system that has the following steps:

- We group servers with the same hardware configuration, such as CPU processor_vendor, number of cores, and other hardware components (e.g., disk type, memory) into a single *pool*, as they show similar power consumption profile.
- Before the first server in a pool is provisioned to a user, we run synthetic CPU-bound benchmarks on the server and collect the following training data: (1) CPU utilization (`actual_cpu_util`) provided by monitoring agents running at the host-complex; and (2) BMC sensor readings provided by the BMC-complex: `overall_power`, `cpu_power`, `systemboard_power`, `fan_power`, and `hdd_power`.
- Figure 3 shows the training phase. We train a per-pool ML regression *model* with the CPU utilization data and the BMC sensor readings data collected from the first server in a pool.

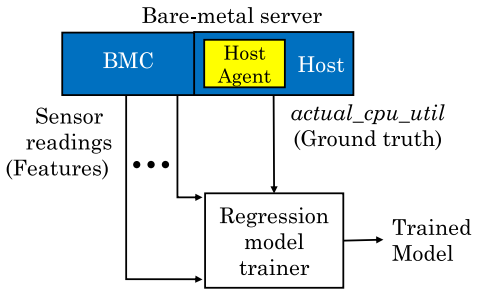

FIGURE 3. Training phase in NASCENT.

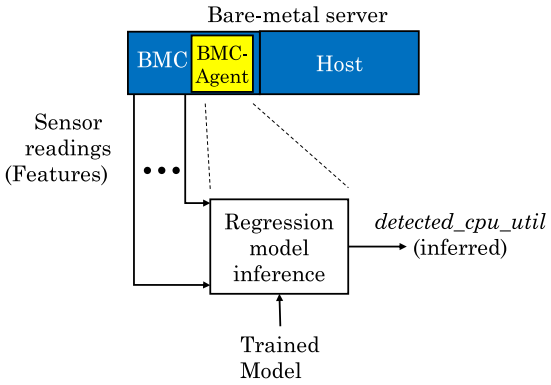

FIGURE 4. Inference phase in NASCENT.

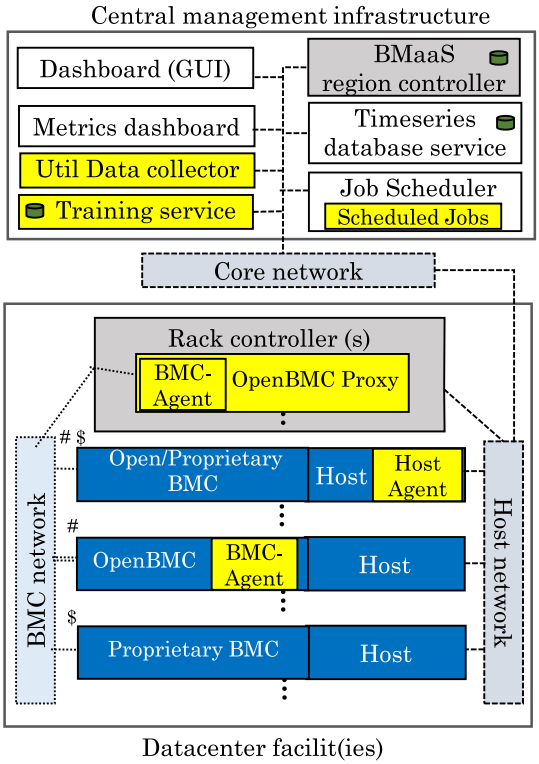

FIGURE 5. BMaaS cloud architecture using NASCENT.

- Figure 4 shows the inference phase. The trained model is deployed in BMC-complex of all servers in a pool, that is, servers with the same configuration. At runtime, a deployed server periodically reads actual BMC sensor readings and infers CPU utilization from the ML model deployed locally in BMC-complex.

By following the above steps, NASCENT detects CPU utilization without running monitoring agents in host-complex, thus satisfying **R1**. NASCENT pools servers with the same hardware configuration and trains a per-pool ML model; this generalizes utilization detection for a cloud with a diverse set of server configurations, thus satisfying **R2**. NASCENT uses regression methods as the ML models, which infers utilization at fine granularity, thus satisfying **R3**. These models require a small amount of computing and memory resources, thus easily fit in the resource-constrained BMC-complex. Moreover, the sensor readings data need not be transmitted over the network or stored for processing. Hence, NASCENT is fast and scales well for large clouds and meets **R4**.

C. BMaaS CLOUD ARCHITECTURE USING NASCENT

Figure 5 shows the high-level architecture of a BMaaS cloud using NASCENT with bare-metal servers. The BMaaS region controller contains a database of all tenants, users, and servers. A graphical user interface (GUI) based dashboard is used to commission, acquire, deploy, release, and decommission servers. Within a datacenter facility, one or more rack controllers are designated to provide near-rack network services such as pre-execution environment (PXE with DHCP, TFTP, NTP services).

We have added the following five new components to the existing BMaaS cloud architecture [39], which are highlighted in yellow in Figure 5. noitemsep

1) *Training service* provides a central web service for building ML models on the data generated using benchmarks, before or after deploying a server (when there are changes to hardware configuration). It stores trained models for each pool of homogeneous servers.
2) *Host-Agent* is a monitoring agent which runs on hosts to collect `actual_cpu_util` during training phase.
3) *Utilization data collector* receives CPU utilizations inferred by BMC-complexes of servers and stores it in a time-series database (TSDB).
4) *Job scheduler* helps in running scheduled tasks periodically.
5) *BMC-Agent* is an embedded application running either in the BMC-complex of a server or on *OpenBMC proxy* hosted by the rack controller.

NASCENT training phase and inference phase are designed to work with both OpenBMC hardware and proprietary hardware in the BMC-complex. For servers with OpenBMC compatible hardware (indicated with # in Figure 5), the inference engine module (named BMC-Agent) can be run in their respective BMC-complex. However, many private cloud datacenters have servers with proprietary BMC hardware with closed firmware running on them

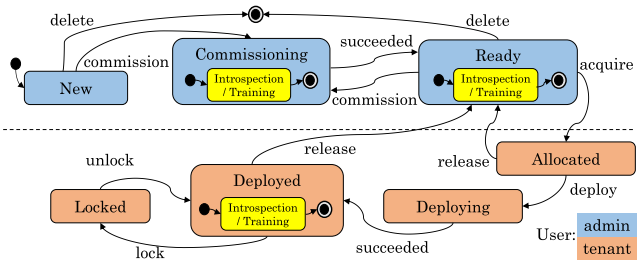

FIGURE 6. Automated model training during life-cycle of a bare-metal server in BMaaS cloud.

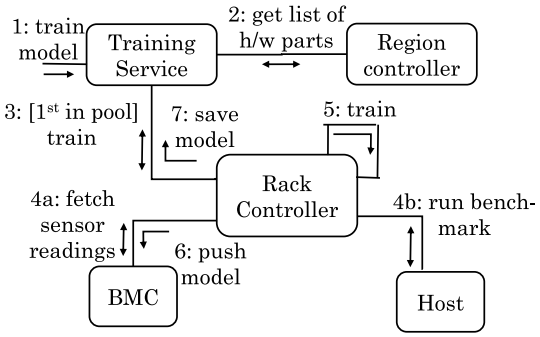

FIGURE 7. Data collection using benchmarks and ML model training service in NASCENT.

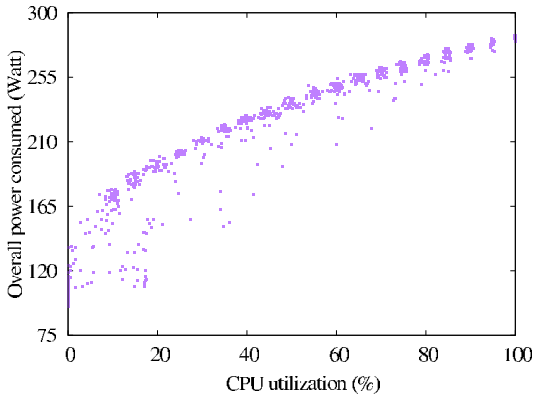

FIGURE 8. Percentage CPU utilization vs overall power consumed for benchmark data.

(indicated with $). So deploying the inference engine in their BMC-complex is not feasible as it requires firmware changes. To address this problem, for servers with proprietary BMC hardware, we propose spinning a lightweight VM on the rack controller, named as OpenBMC proxy; this VM mimics the OpenBMC hardware. We run a BMC-Agent for each server with proprietary BMC in the VM. During the training phase, servers with either OpenBMC or proprietary BMC (indicated with #$ in Figure 5) are treated in the same manner.

D. TRAINING PHASE BEFORE DEPLOYMENT

Figure 6 shows Unified Modeling Language (UML) state-machine diagram [40] for automated training and deployment of ML model during the life-cycle of a typical bare-metal server [41]. During the commissioning phase, necessary hardware introspection is performed to check the health of a few hardware parts (*e.g.* CPU, RAM, Disks). In addition, the introspection data is stored in the region controller's database. The introspection data contains details of each server's hardware parts such as product vendor, CPU model, number of cores, motherboard, CPUs, disks, NICs, accelerators, etc. This hardware introspection activity is triggered whenever hardware parts are added to or removed from an existing server.

Figure 7 shows UML interaction diagram [40] for automated training and deployment of ML model. After hardware introspection activity, the ML model training service, collects training data and trains models. More specifically, the training service gets server's hardware specifications from the region controller (steps 1-2). If the hardware specification is not seen before (step 3), the service runs synthetic CPU-bound benchmarks and collects BMC sensor readings from the BMC-complex (step 4a) and the CPU utilization data from the host-complex (step 4b). The data collection is performed in a parallel and time-synchronized manner through the respective rack controller.

The BMC sensor readings include `overall_power`, `cpu_power`, `systemboard_power`, `fan_power`, and `hdd_power`. The synthetic benchmarks perform the following steps in the host-complex: (1) sleep command mimics an idle server with CPU load of 0 for 30 minutes. (2) stress-ng [42] tool is used to run CPU loads ranging from 1% to 100% with an increment step of 5% and from 100% to 1% with a decrement step of 5%. Both the steps are run for one hour each and `actual_cpu_util` is captured for every five-second interval using `mpstat` [43] command. We choose this interval because the BMC in servers used for evaluation provides running-average readings for every five seconds. The `actual_cpu_util` from host-complex and sensor readings are subjected to inner-join at the rack controller to get benchmark data. Figure 8 shows data generated for the synthetic benchmark for a two-hour duration.

The benchmark data is used for training and validation at the rack controller (step 5), and the ML model is automatically deployed in the BMC of all servers in a pool (step 6). The generated model is sent to the training service, which can be used for other servers with the same hardware specifications in the future (step 7).

E. INFERENCE PHASE AFTER DEPLOYMENT

UML interaction diagram in Figure 9 shows the workflow of the inference phase of NASCENT. In all servers, the deployed BMC agents periodically (every 5 seconds) fetch sensor readings and give the readings as input to the ML model to infer CPU utilization viz., `detected_cpu_util`. To check the accuracy of the inference after deployment, from few servers where cloud administrator has access to the host-complex, CPU utilization information `actual_cpu_util` from the host-complex is collected and compared with the corresponding `detected_cpu_util`. As shown in Figure 9,

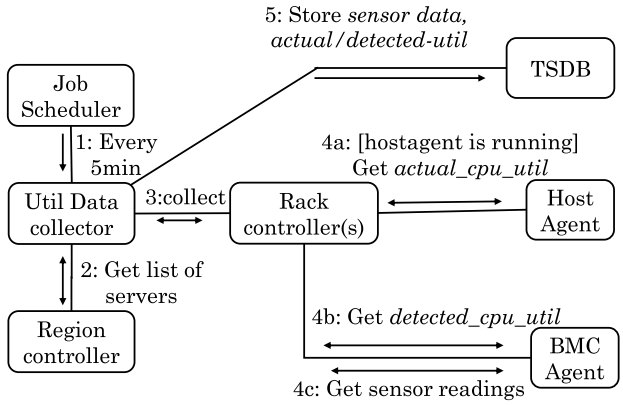

FIGURE 9. Utilization detection of servers during inference phase of NASCENT.

a data collection job is periodically (every five minutes) triggered by the job scheduler, and the job fetches CPU utilization data and stores the data in the TSDB. In addition, `detected_cpu_util` samples in that 5-minute duration are fetched from the respective BMC and stored in the TSDB. Using a dashboard we implemented (see Figure 5), the cloud administrator can view both present and past samples of `actual_cpu_util` and `detected_cpu_util` stored in the TSDB.

1) APPLICATIONS OF UTILIZATION DETECTION

E-mail alerts are sent periodically (*e.g.*, every day) to respective users of the servers if `detected_cpu_util` or `actual_cpu_util` of servers show near-zero utilization for the entire period (*e.g.*, one day). Such servers could either be released or re-purposed to get cost and power savings. Alerts are also sent to users when abnormal utilization of CPU is detected to let users check possible anomalous usage (*e.g.* crypto-jacking [44]–[47] attacks). Further, depending on the cloud management applications, other custom alerts can also be generated.

F. FEATURE VECTOR IN ML MODELS

The ML model in NASCENT is trained with sensor readings as features. The ground truth is the utilization of servers, which is the `actual_cpu_util` collected from the host-complex during benchmarking. As all the features have numeric values with little different ranges, we used the normalization values of the features.

To understand which features (sensor readings collected through BMC) have a high correlation with CPU utilization, we study the relationship between them using two correlation methods: Pearson correlation coefficient (α) and Spearman correlation coefficient (β) [48]. α evaluates the linear relationship between two variables, whereas β evaluates the monotonic relationship: α is linear when the change in one variable leads to a proportional change in another variable; β measures the change in a variable, but the relationship is not necessarily linear. Both α and β have a value between $+1$ and -1, where 1 is a total positive correlation, 0 is no correlation and -1 is a total negative correlation.

TABLE 2. Correlation of the features with `actual_cpu_util` metric.

Feature	Pearson coeff (α)	Spearman coeff (β)
`overall_power`	+0.914	+0.922
`cpu_power`	+0.877	+0.803
`systemboard_power`	+0.811	+0.782
`fan_power`	-0.058	-0.069
`hdd_power`	-0.131	-0.115

Table 2 shows α and β in a sample dataset for a CPU-bound workload described in section §IV-B. We can see three features that are positively correlated with `actual_cpu_util` metric: `overall_power`, `cpu_power`, and `systemboard_power`. This motivates us to use these sensor readings as a feature vector for training ML models. Please note that `fan_power` and `hdd_power` are also correlated but negatively, probably because the specific dataset is CPU-bound.

G. ML MODELS

For detecting fine granular CPU utilization (R3 in §III-A) we use regression methods. Regression is a method that is used to learn a relationship between one dependent variable and other independent variables. $y = w^T x$ where w is a coefficient vector, x is feature vector, y is a target value. From Figure 8, we observe that CPU utilization does not have a linear relationship with power consumption. This is because the power consumption depends on two components: static and dynamic. The static power consumption is constant across all workloads, but the dynamic power does not vary linearly with utilization. We trained models using three non-linear ML regression algorithms: (a) K-Nearest Neighbor regression (KNNR), (b) Support vector regression (SVR) - Radial Basis Function (RBF), and (c) Support vector regression - Polynomial.

1) K-NEAREST NEIGHBOR REGRESSION

For a given test sample, the KNNR model [49], [50] measures the Euclidean distance of the sample from all the training samples and selects the K nearest samples. Then, it takes the average of the respective K training samples' target values, and the average result is the predicted value of the given test sample. Note that this regression model does not learn the relation between the features and the target. The KNNR model consists of all the training data samples and K value, also called hyper-parameter.

2) SUPPORT VECTOR REGRESSION

In SVR [51], [52], we minimize the L2-norm of the coefficient vector and also impose the constraint that the absolute error should be less than a specific threshold ϵ. The ϵ (epsilon-tube) is a hyper-parameter that can be tuned during training to get the best model. It helps in avoiding over-fitting or under-fitting of the model. Also, it helps in getting a more

generalized model with respect to the training data. Here, we seek to minimize:

$$J(\theta, K) = \ min\frac{1}{2}\|w\|_2^2 \ K(x_i, x_j) \text{ such that } \ | \ y_i - w_i^T x_i \ | < \epsilon$$

The SVR method uses a kernel K to best fit the model. Depending on the relationship between the dependent and the independent variables, a suitable kernel such as linear, polynomial, or radial basis function (RBF), can be used in SVR. Linear kernel and polynomial kernel are similar, but the function is defined differently, i.e., the order in the linear kernel is one, while the order in the polynomial is arbitrary. SVR with linear kernel learns a linear function, while SVR with polynomial kernel learns a non-linear function. The RBF kernel uses normal curves around the data points and calculates the Euclidean distance between the data points, which best fits the model. In general, the RBF kernel provides the lowest error, but its inference latency is higher than that of linear and polynomial kernels.

NASCENT assorts servers with the same hardware configuration into a pool. A per-pool model is trained using data collected from the first server in that pool and infer the `cpu_util` of all the servers (including the first server) in the pool.

H. FLOW OF INFERENCE AND TRAINING PHASES

To give insights, we now discuss the pseudo-code of SVR training and inference. Let $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ be a data matrix of the sensor readings where n is the number of examples, and m is the number of features, and $\boldsymbol{y}$ is a vector of CPU utilization values. For every instance of $\boldsymbol{x}_j \in \mathbb{R}^m$ in the data matrix, there is a CPU utilization value y_j. The instances in the dataset y_j vary from 0 to 100, which is the CPU utilization percentage.

Algorithm 1 shows pseudo-code for model training. Training of this model expects feature vector X and target vector y as the input. It then passes the feature value to kernel K, which transforms it into a higher dimension space and finds the best linear model in that space by optimizing the cost function $J(\theta, K)$. Loop (step 5-9) keeps iterating until the desired ϵ is achieved.

Algorithm 1 Support Vector Regression Training

Input: $\{X_j\}_{j=1}^n$ ▷ Sensor readings
Input: $\{y_j\}_{j=1}^n$ ▷ Actual CPU utilization
Input: ϵ ▷ Epsilon-tube (a hyper parameter)
Output: θ ▷ SVR Model

1: **Initialize:** $\alpha \leftarrow 1.0$ ▷ Learning rate
2: **Initialize:** $\theta \leftarrow \{\theta_i\}_{i=1}^m$ ▷ Random initialization
3: $K \leftarrow$ SVRPOL or SVRRBF ▷ Kernel to be used
4: **Declare:** $J(\theta, K)$ ▷ Cost function
5: **repeat**
6: **for** $\theta_i \in \theta$ **do**
7: $\theta_i \leftarrow \theta_i - \alpha \times \frac{\partial}{\partial \theta_i} J(\theta, K)$ ▷ Optimize
8: **end for**
9: **until** $\| \frac{\partial}{\partial \theta_i} J(\theta, K) \| \leq \epsilon$ ▷ Convergence
10: **return** θ

Algorithm 2 Support Vector Regression Inference

Input: θ ▷ SVR Model
Input: $\{X_j\}_{j=1}^k$ ▷ Sensor readings
Output: $\{\hat{y}_j\}_{j=1}^k$ ▷ Inferred CPU utilization

1: $K \leftarrow$ SVRPOL or SVRRBF ▷ Kernel to be used
2: $\hat{y} \leftarrow (\theta^T K(X))$ ▷ Dot product
3: **return** $\hat{y}$

Algorithm 2 shows pseudo-code for the inference phase. During inference, the model (parameters vector and the kernel) and data matrix $\boldsymbol{X} \in \mathbb{R}^{k \times m}$ having k sensor readings are passed as the input to get inference results as a vector $\hat{y}$. In step 2 of the algorithm, the data matrix is processed by the kernel and then subjected to dot-product with transposed parameters vector to get a vector of scalar values $\hat{y}_j$. These $\hat{y}_j$ are k inferred CPU utilization values corresponding to the given $\boldsymbol{X}$.

In summary, we give a data matrix of sensor readings as well as the ground truth of the CPU utilization and try to find the optimal weights for a support vector regression model. The model is used to predict the unknown y for a given $\boldsymbol{x}$. The optimal weights are also tested on data that the model has not been exposed to before.

IV. EXPERIMENTATION AND RESULTS

In this section, we first explain the experimental setup (§IV-A), workloads considered for the evaluation (§IV-B), and evaluation parameters and metrics (§IV-C). We then present experiments performed with static threshold based heuristics (§IV-D) and NASCENT (§IV-E). We then present results from ablation study of the proposed ML models (§IV-F). We compare the results provided by the best ML model in NASCENT with that provided by the existing proprietary BMC firmware functionality, namely CUPS (§IV-G). We finally present the salient features and applications of NASCENT (§IV-I).

1) SETUP

We conduct the experiments by implementing NASCENT in a private bare-metal cloud commissioned as part of Dhi-Ojas project [39]. It has a region controller and two rack controllers managing 105 bare-metal servers, belonging to 12 tenants. All the servers run Linux OS. All of these servers have proprietary BMC firmware.

2) CLOUD TECHNOLOGIES

We use open-source cloud technologies such as MAAS [10] for cloud management, InfluxDB [53] for the time-series database service, Grafana [54] dashboard to display summary information for monitoring purposes, and Jenkins [55] for scheduling jobs.

3) DEVELOPMENT ENVIRONMENT

We have used a pool having four Fujitsu RX2540M2 servers (2 processors with 28 cores each and 64GB RAM) to evaluate NASCENT, as these servers have sensors for measuring the overall power consumption (*i.e.*`overall_power`) and the power consumption of individual components, namely `cpu_power`, `systemboard_power`, `fan_power` and `hdd_power`. The RX2540M2 BMC firmware maintains a running average of `overall_power` for a five-second window. The RX2540M2 BMC firmware reports instantaneous readings for the power consumption of individual components. We also evaluated NASCENT on two other different pools, viz., DellEMC PowerEdge R630, which has four servers and HPE Proliant DL380G9, which has eleven servers. The servers in the two pools have a running average of `overall_power` of 60 seconds and 5 minutes, respectively. All these servers have proprietary BMC firmware. Note that OpenBMC based servers provide the flexibility to adjust the BMC firmware to report running-average readings of desired sensors for custom running-average windows and run user-defined applications.

A. EXPERIMENTAL SETUP

As these four servers have proprietary BMC, we run BMC-Agent in OpenBMC proxy (a QEMU VM instance) that emulates the OpenBMC based BMC hardware. One proxy VM is created for each proprietary BMC, and it operates inside the rack controller, as shown in Figure 5. Note that we do not need this VM for OpenBMC compatible hardware servers; since the BMC-Agent developed works under the BMC hardware resource constraints, it can be easily ported to OpenBMC compatible hardware servers. In the future, we plan to port the BMC-Agent to OpenBMC compatible servers and also customize OpenBMC firmware to compute running-average readings to embrace all the individual power consumption sensors. This would allow us to apply NASCENT for other production workloads such as I/O bound, memory-bound, network bound, accelerator bound.

The BMC and thus, the proxy VM is highly resource-constrained, for example, it has 1 ARM vcpu, 256 MB RAM and 140 MB flash and allows only C/C++ execution environment. This constrains us to develop the inference engine (BMC-Agent) in C++ using `libsvm` library [56]. The training code, which runs in the NASCENT training service, is developed in Python and `sklearn`. The host monitoring agent (Host-Agent) is a lightweight Python script that returns the aggregate percentage of CPUs used (ranging from 0 to 100), averaged over the last five-second interval. Utilization-data collector is developed in Python and executed by a Jenkins periodic-job to retrieve and save results in an InfluxDB database.

B. WORKLOADS

1) TRAINING (BENCHMARK) DATASET

We train one model for every pool of servers with identical hardware configurations. The training dataset comprises of `actual_cpu_util` from Host-Agent and corresponding `overall_power` sensor reading from the BMC-Agent. The model is trained by running synthetic benchmark workloads on one of the servers from each pool before provisioning. The benchmarking scripts run for about 2.5 hours to collect 1574 training data points. These data points are used to train machine learning models.

2) INFERENCE (PRODUCTION) DATASET

To check the accuracy of NASCENT, we have collected production datasets with `actual_cpu_util` and respective `overall_power` readings for the following workloads: (1) Build server workload, (2) DenseNet [57] Training workload, (3) Deep learning recommendation model (DLRM) [58] from MLPerf suite. Note that we took permission from the tenants to collect these statistics from the host complex. The reasoning for the choice of these workloads is as follows:

(1) Build server workload: A build server is used for building (compiling and linking) the source code of the software. These builds form typical CPU-extensive workloads and hence, are chosen for evaluation.

(2) DenseNet training workload: DenseNet is a convolutional neural network architecture primarily used for computer vision applications. This workload runs on multiple Linux containers simultaneously inside the bare-metal server.

(3) DLRM training workload: MLPerf training suite of benchmarks measure how fast a system can train ML models to a target quality metric. Of this, the DLRM [59] recommendation training workload is run on a bare-metal server. Despite the emergence of accelerators, CPU remains a widely-used platform for running ML services in large-scale datacenters such as those managed by Facebook and Microsoft [60]. DenseNet and DLRM training emulate such ML workloads.

Kindly note that these ML workloads are entirely different from the ML models used to make predictions in NASCENT. We have included both non-ML (build server) and ML (DenseNet and DLRM) workloads to show that NASCENT works well for all types of workloads, including traditional and emerging workloads.

C. EVALUATION PARAMETERS AND METRICS

We now define four parameters used for the evaluation of NASCENT. (1) Training Time, (2) Model Size, (3) Inference Time, and (4) Model Performance.

1) TRAINING TIME

Average-time taken (in ms) for generating regression model after training data is available.

2) INFERENCE TIME

Average-time taken (in ms) for real-time inference of each sample on OpenBMC proxy VM.

3) MODEL SIZE

Size (in KB) of regression model generated after training.

4) MODEL PERFORMANCE

The efficiency of regression models during inference on unseen data defined in terms of various metrics. We discuss the following three metrics used for the evaluation of different regression models experimented in this paper.

5) ROOT MEAN SQUARED ERROR (RMSE)

RMSE is the most widely used metric for regression tasks. It is computed as the square root of the average squared differences between the model's predictions and the actual observations.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2}$$

where, y_i - actual value, p_i - predicted value.

6) MEAN ABSOLUTE ERROR (MAE)

It is the absolute difference between the actual target value and the value predicted by the model. Since we do not take the square of differences, and all the individual differences are weighted equally, it is more robust to outliers.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - p_i|$$

7) COEFFICIENT OF DETERMINATION (R^2)

The ''coefficient of determination'' metric compares the current model with a constant baseline and evaluates the current model's superiority. The value of R^2 generally varies from 0 to 1 as the model improves. A higher R^2 value shows a better model. However, if the model does not follow the trend of training data, the metric's value is negative.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$
$$SS_{res} = \sum_{i=1}^{n}(y_i - p_i)^2$$
$$SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

where, y_i - actual value, p_i - predicted value, $\bar{y}$ - mean of actual values, SS_{res} - sum of squares of difference between actual and predicted value, SS_{tot} - sum of squares of difference between actual and mean value.

8) ADJUSTED R^2 (R^2_a)

R^2 suffers from the problem that the scores improve on increasing terms even though the model is not improving. *Adjusted* R^2 adjusts for the increasing predictors and shows improvement only if there is a real improvement in the model. Value of *Adjusted* R^2 also varies from 0 to 1 as the model improves where a higher value refers to a better model.

$$R^2_a = 1 - \left[\left(\frac{n-1}{n-k-1}\right) * (1 - R^2)\right]$$

where, R^2_a - *Adjusted* R^2, n - number of samples in dataset, k - number of independent variables, R^2 - coefficient of determination.

TABLE 3. Overall power consumption ranges (in watts) at various Percentage CPU utilization levels of a Fujitsu Primergy RX2540M2 used as build server.

% Utilization range [Min, Max]	Power consumption range [Min, Max], Median
[0,20]	[95, 237], 104
(20,40]	[187, 242], 217
(40,60]	[214, 277], 251
(60,80]	[227, 303], 273
(80,100]	[235, 316], 289

D. EXPERIMENTS WITH STATIC THRESHOLD BASED HEURISTICS

One may use a naive approach of measuring the static minimum and maximum thresholds on power consumed for every percentage CPU utilization level by running different workloads. These thresholds can be deployed in BMC to infer percentage CPU utilization at run-time. Since the thresholds can be derived before provisioning a server, this method is non-invasive, and the logic is simple enough to run in the resource-constrained BMC-complex. Hence, this solution meets **R1** and **R4**, respectively. This method is also generic as we can obtain thresholds for every CPU model; therefore, it meets **R2**. However, as we show below, the predictions provided by this method are inaccurate, thus violating **R3**.

Table 3 shows the minimum and maximum thresholds obtained from a production build server, which is a CPU-bound workload. From the table, we can observe a considerable overlap in power consumed at different percentage CPU utilization levels, causing ambiguity in determining the percentage CPU utilization from a given power consumption value. For instance, the power consumption of 200 watts corresponds to two ranges of CPU-utilization, viz., [0, 20] and (20, 40].

More specifically, Figure 10 shows the correlation between the CPU load and the overall power consumed from a production server. From this figure also, we can observe high variance in determining power consumed (y-axis) by the server at different percentage CPU utilization levels (x-axis). As shown in Figure 10, we also observed similar high variance behavior for CPU-bound production workloads. The data for production workload is collected from a server already deployed after taking its owner's permission. To summarize, given such a large interval size and variance in the power consumption, the server utilization cannot be accurately detected using the static thresholds.

We also observe that the power consumption at different utilization levels varies significantly across different servers. This is because our private cloud has bare-metal servers with a wide variety of hardware configurations provisioned to users based on their requirements. Few minor changes in the count or type of peripheral devices significantly change the power consumption ranges. The main factors that influence power consumption are heterogeneity in count and type of each server component such as processor, processor-frequency,

TABLE 4. Model performance with different regression methods on various datasets using `overall_power` as the feature and $\epsilon = 0.1$ (For RMSE and MAE, lower is better. For $AdjR^2$, higher is better).

Dataset	Data Subset	Subset Size	Utilization [Min,Max]	RMSE			MAE			$AdjR^2$		
				KNNR	SVRPOL	SVRRBF	KNNR	SVRPOL	SVRRBF	KNNR	SVRPOL	SVRRBF
Synthetic benchmark	Train	1575	0, 100	3.34	6.86	4.19	1.48	4.82	2.02	0.98	0.95	0.98
Build server	Test	39369	0, 100	5.14	8.96	2.89	2.95	6.37	1.49	0.85	0.56	0.95
DenseNet training	Test	7386	0, 58.05	4.18	6.58	3.40	5.14	2.91	2.41	0.89	0.73	0.93
DLRM training	Test	3903	0, 100	10.65	15.94	8.73	10.19	10.76	8.32	0.64	0.21	0.76

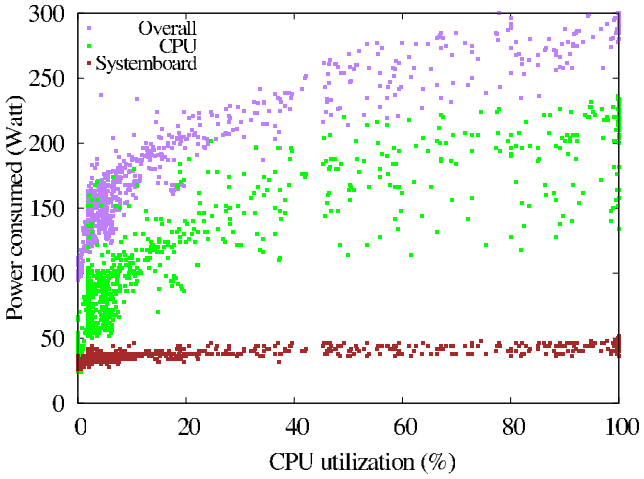

FIGURE 10. CPU utilization vs `overall_power`, `cpu_power`, `systemboard_power` consumed for a production build server.

core(s), memory, disk(s), OS, and a variety of hardware devices (*e.g.*, GPUs, FPGAs) connected to a server.

Therefore, inferring the current CPU utilization based on power consumption-based heuristic alone is inadequate.

E. EXPERIMENTS WITH THE PROPOSED NASCENT SOLUTION

From Figure 10, we observed that the relationship between overall power and `actual_cpu_util` is not exactly linear. We evaluated the models trained using three non-linear ML regression algorithms discussed in section §III-G: (a) K-Nearest Neighbor regression (KNNR), (b) Support vector regression - Radial Basis Function (SVRRBF), and (c) Support vector regression - Polynomial (SVRPOL). As for the hyper-parameter for SVRRBF and SVRPOL, we set $\epsilon = 0.1$. Table 4 shows the summary of results with prediction errors for three regression models. We have shown results with three metrics viz., root mean squared error (*RMSE*), mean absolute error (*MAE*), and adjusted R^2 ($AdjR^2$). Across all the production workloads, the lowest error is observed for SVRRBF, followed by KNNR and SVRPOL.

1) BENCHMARK RESULT

Figure 11(a) shows the ground truth (GT) *i.e.*`actual_cpu_util`, and `detected_cpu_util` using KNNR, SVRPOL and SVRRBF ML models on the benchmark dataset (training subset). SVRRBF provides the best fit amongst the three ML models; KNNR overfits to the given benchmark dataset, which is evident from the two humps between 100 and 175 `overall_power` range. Due to this overfitting, KNNR performs better on training data but is relatively inaccurate on the test data.

2) PRODUCTION RESULTS

(1) Build server workload: Figure 11(b) shows the ground-truth vs. inferred (predicted) values for the build server workload. As reflected from the benchmark results, SVRRBF provides the best fit and the lowest error across all metrics (Table 4) with an average RMSE of up to 3.

(2) DenseNet Training workload: As shown in Figure 11(c), for DenseNet training also, we observe a similar trend as observed for the build server. Specifically, the KNNR over-fits and the SVRPOL shows the highest error amongst the three ML models.

(3) DLRM training workload: Figure 11(d) shows the results for DLRM training workload. Among the three workloads, the prediction error is highest for the DLRM training workload. This is because the DLRM training workload is resource-intensive and runs for a longer time. This increases the fan speeds to upwards of 1000RPM, which is more than the fan speeds seen with the synthetic benchmark. Hence, the RMSE metric rises to 8.7 for SVRRBF and up to 15.9 for other models.

We observed that on the production dataset, K-nearest neighbor regression achieves RMSE values in the range 4.18 to 10.65, SVRPOL achieves RMSE values between 8.96 and 15.94 and SVRRBF achieves RMSE values between 2.89 and 8.73.

3) TIME AND SPACE OVERHEAD

Table 5 shows training time, inference time, and model size for KNNR, SVRPOL and SVRRBF ML models. KNNR model takes the least time for training, but it has the largest model size (42 KB) amongst all the ML models as KNNR keeps all the training samples in the model. Due to its large size, we do not recommend using KNNR as the final solution. Similarly, SVRRBF has the longest training and inference time amongst all the models with smallest model size (19*KB*). Although SVRRBF has a long training time (65 millisec), it is still negligible compared to the time incurred in data-collection using synthetic benchmarks (2.5 hours).

The inference application running in OpenBMC Proxy VM has a peak memory demand of 4 MB. The total time

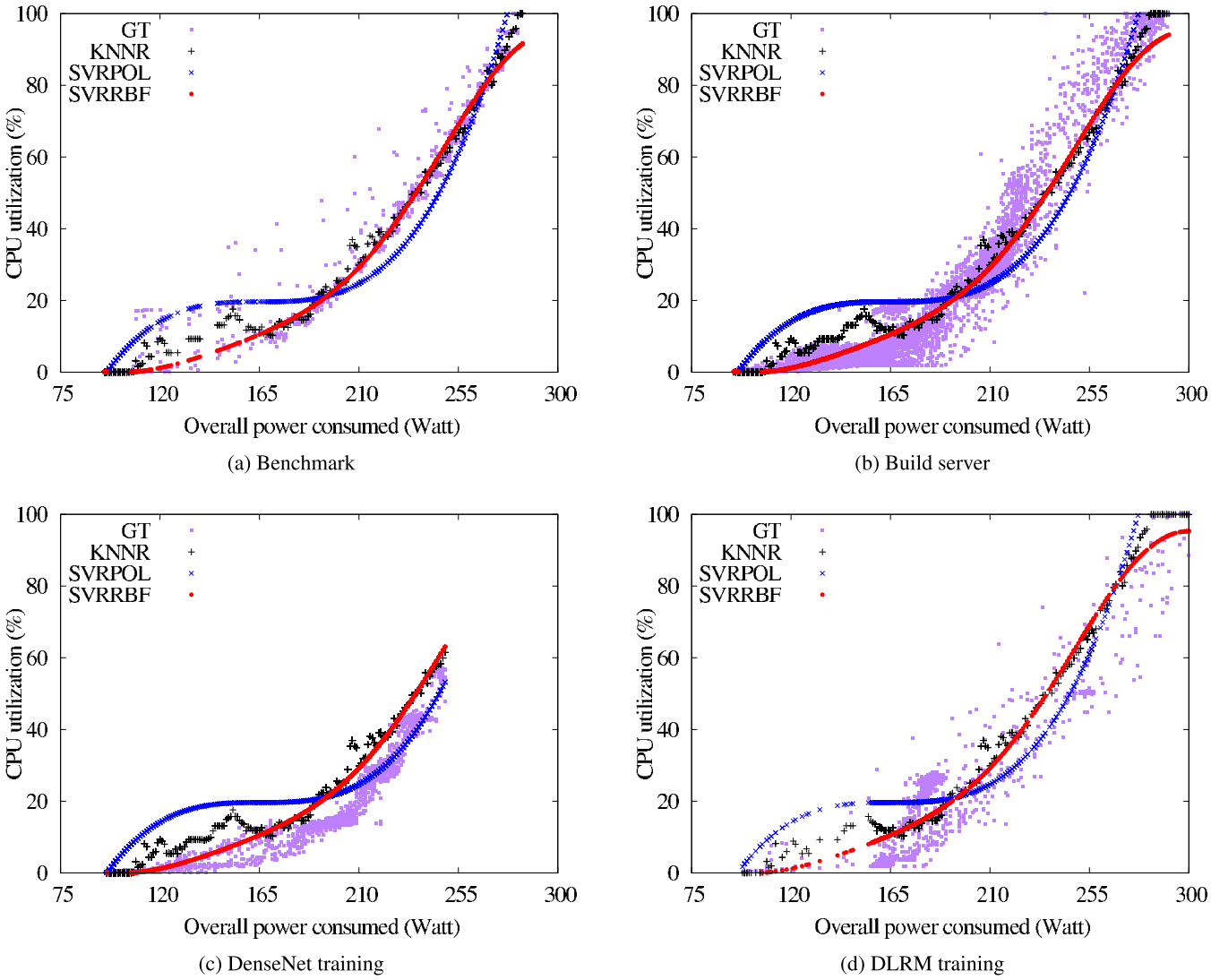

FIGURE 11. Different regression models' `detected_cpu_util` and the `actual_cpu_util` (GT) for benchmark and production (build server, DenseNet training, DLRM training) data.

TABLE 5. Model sizes and inference time for regression models.

Parameter	KNNR	SVRPOL	SVRRBF
Training Time (ms)	2	17	65
Inference Time (ms)	0.015	0.082	0.186
Model Size (KB)	42	33	19

taken to gather sensor readings and perform inference for a single sample is about 120 ms. Out of this 120 ms, more than 119 ms is spent on reading the sensors from the BMC-complex over the BMC-network. Consequently, the actual inference time stands trivial (at 186 μs) compared to the time taken to read sensors (at 119 ms). When the inference is done near the sensors (i.e., inside OpenBMC hardware), the time taken to read all sensors would be relatively lower (approximately 40 ms) since there would be no network overhead.

Overall, from Table 4 and Table 5, we conclude that SVRRBF provides the least error with negligible time and space overheads across multiple test workloads. As such, we recommend using SVRRBF as the final solution.

F. ABLATION STUDY

Our main results presented in Table 4 correlated `actual_cpu_util` with `overall_power`. We now study whether `actual_cpu_util` prediction improves by using `cpu_power` or `systemboard_power`. This is motivated by the fact that in Table 2, `cpu_power` and `systemboard_power` also showed positive correlation with `actual_cpu_util`. We seek to find whether the error in prediction of `actual_cpu_util` from `cpu_power` and `systemboard_power` is low enough. We did not conduct ablation studies for KNNR model, because the model size of KNNR is large (refer Table 5)

TABLE 6. RMSE for models trained with different feature combinations.

Feature(s)	Dataset	SVRPOL	SVRRBF
overall_power cpu_power systemboard-_power	Synthetic benchmark	9.44	4.96
	Build server	15.90	13.26
	DenseNet training	22.45	4.97
	DLRM training	15.96	12.98
overall_power cpu_power	Synthetic benchmark	10.24	4.89
	Build server	15.19	7.23
	DenseNet training	12.16	3.47
	DLRM training	16.32	10.52
systemboard-_power	Synthetic benchmark	29.97	24.04
	Build server	30.25	24.30
	DenseNet training	68.30	13.49
	DLRM training	19.83	23.12
cpu_power	Synthetic benchmark	10.33	5.75
	Build server	15.94	10.33
	DenseNet training	11.93	4.45
	DLRM training	15.34	10.51

TABLE 7. RMSE for models trained using `overall_power`, `cpu_power` and `systemboard_power` features, for different values of hyper-parameter ϵ.

Hyper-parameter	Dataset	SVRPOL	SVRRBF
$\epsilon = 0.05$	Synthetic benchmark	9.44	4.97
	Build server	15.86	13.30
	DenseNet training	22.27	4.98
	DLRM training	15.94	13.03
$\epsilon = 0.15$	Synthetic benchmark	9.44	4.96
	Build server	15.92	13.26
	DenseNet training	22.64	4.97
	DLRM training	15.95	12.94

and grows linearly with training dataset size and hence not suitable for inference in embedded systems (BMC).

Table 6 shows RMSE in predicting `actual_cpu_util` from different feature sets using combinations of features `overall_power`, `cpu_power`, and `sysboard_power`. We observe that these RMSE values are much higher than that obtained on using `overall_power` (refer Table 4). This can be explained from the fact that the readings of `cpu_power` and `systemboard_power` sensor are highly noisy, as confirmed from Figure 10. These readings are instantaneous readings whereas that of `overall_power` are average readings and hence, are noise-free. These results and the explanation justify our choice of correlating `actual_cpu_util` with `overall_power`, but not with `cpu_power` and `systemboard_power` (refer Table 4).

Table 6 shows error metrics for the two models trained with default value of hyper-parameter $\epsilon = 0.1$. As `cpu_power` and `systemboard_power` features are noisy, we have also done ablation study by varying the value of ϵ for models trained with all the three features `overall_power`, `cpu_power` and `systemboard_power`. From Table 7, for $\epsilon = 0.05$ and $\epsilon = 0.15$, we did not notice any reduction in RMSE in the two SVR models (SVRPOL and SVRRBF) for all datasets, compared to their corresponding values in Table 6.

These results from ablation study justify our choice of hyper-parameter value $\epsilon = 0.1$ (refer Table 4).

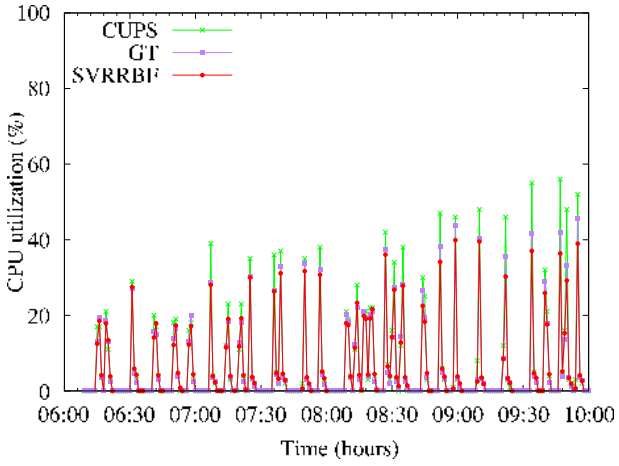

FIGURE 12. CPU utilization reported by CUPS, `actual_cpu_util` (GT) and `detected_cpu_util` (SVRRBF) for production data (build server).

G. COMPARISON OF NASCENT SVRRBF WITH CUPS

To compare SVRRBF with the existing proprietary BMC functionality (CUPS), we collected `detected_cpu_util` (SVRRBF), `actual_cpu_util` (GT), and the corresponding CPU utilization readings from CUPS of BMC firmware during a build server workload. As shown in Figure 12, the error between `detected_cpu_util` and `actual_cpu_util` is inline with RMSE shown in Table 4. We observed that there is a minor difference in CPU utilization readings reported by CUPS and `actual_cpu_util`. This could be the case as `actual_cpu_util` is retrieved from the OS in host-complex and the CUPS readings from the BMC firmware. Also, CUPS uses a sampling based running-average. A small value of RMSE confirms that NASCENT predicts the utilization accurately and is also comparable with an industry-grade BMC functionality.

Apart from this CUPS functionality, to the best of the authors' knowledge, no previous work proposed a non-invasive technique for detecting server utilization. Main limitation of CUPS functionality is that, it is available only in servers from few vendors (viz., Fujitsu, DellEMC) requiring additional license cost, whereas NASCENT is agnostic to server vendors and can be applied to all servers in a bare-metal cloud with heterogeneous servers without any licensing cost.

H. RESULTS ON DIFFERENT POOLS

To evaluate NASCENT further, we used a pool of "DellEMC PowerEdge R630" servers and another pool of "HPE Proliant DL380G9" servers. We generated an SVRRBF model on a server from each pool and tested it on another machine from the same pool. From Table 8, we did not notice a significant deviation in RMSE in the two SVRRBF models for all the datasets, compared to their corresponding values in Table 4 on Fujitsu RX2540M2 servers. The results from these pools are in-line with the evaluation of NASCENT on the "Fujitsu RX2540M2" servers.

TABLE 8. RMSE for models trained using `overall_power` feature for different pools (Server Vendor and CPU models).

Pool	Dataset	SVRRBF
Fujitsu Primergy RX2540M2 & Intel Xeon(R) E5-2660 v4	Synthetic benchmark	4.19
	Build server	2.89
	DenseNet training	3.40
	DLRM training	8.73
DellEMC PowerEdge R630 & Intel Xeon(R) E5-2620 v4	Synthetic benchmark	4.97
	Build server	3.38
	DenseNet training	4.92
	DLRM training	9.34
HPE Proliant DL380G9 & Intel Xeon(R) E5-2690 v4	Synthetic benchmark	4.64
	Build server	2.97
	DenseNet training	4.19
	DLRM training	9.21

I. SALIENT FEATURES OF NASCENT

1) PRIVACY AND SECURITY GUARANTEES

By leveraging the BMC-complex features, NASCENT satisfies security and privacy requirements. NASCENT is a processor agnostic approach with no additional license costs and no need to upgrade existing hardware. Also, NASCENT does not use any static power consumption-based thresholds.

Leveraging BMC-complex features also has other advantages. First, the network to which the BMC network port is connected is not exposed to the public internet. This reduces the risk of being exploited by attackers from outside. Second, recent advancements in the BMC technology such as the rise in the usage of OpenBMC compatible hardware in servers [17], [61] and its visibility to power-hungry components (*e.g.*, GPUs, smart NICs) provide opportunities to predict the overall server utilization accurately. Finally, our approach is secure as OpenBMC keeps implementation details open for everyone, thus enables fast fixes to security issues. In the future, using OpenBMC, we plan to extend this work to other workloads such as memory-bound and GPU-bound.

2) APPLICATIONS OF CPU UTILIZATION DETECTION

Cypto-jacking is a cyber-attack where hackers compromise servers and then, illicitly mine cryptocurrency on these compromised servers. Related works such as Bijmans *et al.* [44], Musch *et al.* [45], Hong *et al.* [46], and Kirat *et al.* [47] detect compromised servers based on abnormally high utilization reported by the host-complex agents. NASCENT complements these works by detecting CPU utilization in a non-invasive manner. In the ''energy-proportional computing'' [62]–[65], the overall power consumption of a server is proportional to the amount of useful work done (utilization). To achieve energy-proportional computing, power consumption and CPU utilization data is crucial. We envision that the idea of energy-proportional computing can be enabled for the bare-metal cloud using the utilization detection enabled by NASCENT. This would allow using energy-proportionality as a parameter while provisioning servers to tenants.

V. CONCLUSION AND FUTURE WORK

NASCENT is a machine-learning-based non-invasive solution to find CPU utilization of bare-metal servers in a BMaaS cloud where monitoring agents are not preferred because of privacy and security reasons. The core idea is to leverage the server's sensor readings that are accessible via the BMC hardware module as input features of machine learning models. In addition to these features, the models are trained with CPU utilization data of a few servers collected by monitoring agents before provisioning. Finally, the learned models are deployed in servers not running monitoring agents, and the models are applied to infer the CPU utilization of servers. Our evaluation shows NASCENT can infer CPU utilization with low error (RMSE between 2.9 and 9.3) and is comparable to utilization information by proprietary BMC firmware functionalities (CUPS). NASCENT is open source and available at https://github.com/iithcandle/dhi-ojas.

A. FUTURE WORK

We plan to use the `detected_cpu_util` and respective power consumed for computing energy-proportionateness of the bare-metal servers to help efficient server selection during provisioning. Also, we plan to extend NASCENT for accelerator utilization detection and study its performance. Currently, NASCENT uses uni-variate regression; that is, a single independent variable is used to train a model to predict one dependent variable. In our solution, the independent variable is `overall_power` and the dependent variable is `actual_cpu_util`. In the future, we shall improve our solution by using multi-variate multiple-regression, where we train models on more than one independent variable, and it can predict more than one dependent variable. The independent variables would be `gpu_power`, `hdd_power`, `memory_power`, `nic_power` and `cpu_power`. A model trained on these variables would predict GPU utilization, disk utilization, memory utilization, NIC utilization and CPU utilization individually. It would be a more complex model as there is interdependence among independent variables for hybrid workloads.

ACKNOWLEDGMENT

The authors would like to thank Intel Technology India Pvt Ltd for generous support under Intel India Ph.D. Fellowship Program, and also would like to thank Vaibhav S. Chauhan, master's student at IIT Hyderabad, for giving a helping hand in coding during initial phase of the project.

REFERENCES

[1] A. Mosayyebzadeh, A. Mohan, S. Tikale, M. Abdi, N. Schear, T. Hudson, C. Munson, L. Rudolph, G. Cooperman, P. Desnoyers, and O. Krieger, ''Supporting security sensitive tenants in a bare-metal cloud,'' in *Proc. USENIX ATC*, Jul. 2019, pp. 587–602.
[2] Google. *Google Cloud Bare Metal Solution*. Accessed: Dec. 12, 2021. [Online]. Available: https://cloud.google.com/bare-metal
[3] Oracle Corporation. *Oracle Cloud*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.oracle.com/cloud/
[4] IBM Corporation. *IBM SoftLayer*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.ibm.com/cloud/bare-metal-servers

[5] Rackspace. *Fully Managed Hosting Services and Solutions*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.rackspace.com/cloud/bare-metal
[6] Packet An Equinix Company. *Packet-Bare-Metal Server and on Premise Cloud Provider*. Accessed: Dec. 12, 2021. [Online]. Available: https://metal.equinix.com/product/servers/
[7] NetMagic Solutions. *Managed Dedicated Hosting*. Accessed: Dec. 12, 2021. [Online]. Available: https://ecl.ntt.com/en/support-ecl2/baremetal-server/
[8] Go4Hosting. *Bare-metal Dedicated Servers by Go4Hosting*. Accessed: Dec. 12, 2021. [Online]. Available: https://go4hosting.in/services/bare-metal-servers
[9] X. Zhang, X. Zheng, Z. Wang, H. Yang, Y. Shen, and X. Long, "High-density multi-tenant bare-metal cloud," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 483–495.
[10] Canonical Corporation. *Metal as a Service*. Accessed: Dec. 12, 2021. [Online]. Available: https://maas.io/
[11] Openstack Ironic Community. *Openstack Ironic*. Accessed: Dec. 12, 2021. [Online]. Available: https://wiki.openstack.org/wiki/Ironic
[12] Intel Corporation. *Intel Remote Management Module*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000007182/server-products.html
[13] Aspeed Technologies Inc. *Aspeed 2500 Server Management Hardware*. Accessed: Dec. 12, 2021. [Online]. Available: http://www.aspeedtech.com/server_ast2500
[14] Dell Technologies. *Integrated Dell Remote Access Controller (iDRAC)*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.delltechnologies.com/en-in/solutions/openmanage/idrac.htm
[15] Hewlett Packard Enterprise. *Integrated Ligths Out (iLO)*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.hpe.com/psnow/doc/c04154343
[16] Fujitsu Limited. *Integrated Remote Management Controller (iRMC) ServerView*. Accessed: Dec. 12, 2021. [Online]. Available: https://sp.ts.fujitsu.com/dmsp/Publications/public/ds-irmc-s5-en.pdf
[17] J. Frazelle, "Opening up the baseboard management controller: If the CPU is the brain of the board, the BMC is the brain stem," *Commun. ACM*, vol. 63, no. 2, pp. 38–40, 2020.
[18] J. Frazelle, "Open source firmware," *Commun. ACM*, vol. 62, no. 10, pp. 34–38, Sep. 2019.
[19] B. Rabenstein and J. Volz, *Prometheus: A Next-Generation Monitoring System*. Dublin, Ireland: USENIX Association, May 2015.
[20] Prometheus Community. (2012). *Prometheus Node Exporter for Machine Metrics*. Accessed: Dec. 12, 2021. [Online]. Available: https://github.com/prometheus/node_exporter
[21] Oracle Corporation. *Oracle Bare Metal Servers, no Agents or Hypervisor Needed*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.oracle.com/in/cloud/compute/bare-metal.html
[22] J. Hilland, "Redfish overview," in *Proc. 10th Int. Conf. Utility Cloud Comput. Companion (UCC Companion)*, Austin, TX, USA. New York, NY, USA: Association for Computing Machinery, 2017, p. 119, doi: 10.1145/3147234.3158209.
[23] Dell Technologies. *Monitoring Performance Index of CPU, Memory, and Input Output Modules*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.dell.com/support/manuals/en-in/idrac9-lifecycle-controller-v3.1-series/idrac_3.15.15.15_ug/monitoring-performance-index-of-cpu-memory-and-input-output-modules?guid=guid-61137907-32af-4f1f-a0f4-e11954e1196a&lang=en-us
[24] Dell Technologies. *RACADM Command Line Interface Reference Guide*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.dell.com/support/manuals/integrated-dell-remote-access-cntrllr-8-with-lifecycle-controller-v2.00.00.00/racadm_idrac_pub-v1/systemperfstatistics?guid=guid-d4540613-8f8a-4cd7-a8ae-b0ffdae8e62e&lang=en-us
[25] Hewlett Packard Enterprise. *TelemetryService HPE Server iLO Redfish API*. Accessed: Dec. 12, 2021. [Online]. Available: https://hewlettpackard.github.io/ilo-rest-api-docs/ilo5/#telemetryservice-v1_0_0-telemetry service
[26] OCP Foundation. (2011). *Open Compute Project*. Accessed: Dec. 12, 2021. [Online]. Available: https://www.opencompute.org
[27] Open Power Foundation. (2014). *Open Power9 Servers*. Accessed: Dec. 12, 2021. [Online]. Available: https://openpowerfoundation.org/
[28] T. Latzo, J. Brost, and F. Freiling, "BMCLeech: Introducing stealthy memory forensics to BMC," *Forensic Sci. Int., Digit. Invest.*, vol. 32, Apr. 2020, Art. no. 300919.
[29] D. Borgetto, M. Maurer, G. D. Costa, J. M. Pierson, and I. Brandic, "Energy-efficient and SLA-aware management of IaaS clouds," in *Proc. 3rd Int. Conf. Future Syst., Energy, Comput. Commun. Meet (e-Energy)*, May 2012, pp. 1–10.
[30] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. Conf. USENIX Annu. Tech.*, 2009, p. 28.
[31] C. Lin, P. Liu, and J.-J. Wu, "Energy-aware virtual machine dynamic provision and scheduling for cloud computing," in *Proc. 4th Int. Conf. Cloud Comput.*, Jul. 2011, pp. 736–737.
[32] F. Y.-K. Oh, H. S. Kim, H. Eom, and H. Y. Yeom, "Enabling consolidation and scaling down to provide power management for cloud computing," in *Proc. 3rd USENIX Conf. Hot Topics Cloud Comput.*, Jun. 2011, p. 14.
[33] G. Han, W. Que, G. Jia, and W. Zhang, "Resource-utilization-aware energy efficient server consolidation algorithm for green computing in IIOT," *J. Netw. Comput. Appl.*, vol. 103, pp. 205–214, Feb. 2018.
[34] B. Arzani, S. Ciraci, S. Saroiu, A. Wolman, J. Stokes, G. Outhred, and L. Diwu, "PrivateEye: Scalable and privacy-preserving compromise detection in the cloud," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement.*, 2020, pp. 797–815.
[35] P. Xiao, Z. Hu, D. Liu, G. Yan, and X. Qu, "Virtual machine power measuring technique with bounded error in cloud environments," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 818–828, Mar. 2013.
[36] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Proc. 3rd Int. Conf. Commun. Mobile Comput.*, Apr. 2011, pp. 122–125.
[37] P. Garraghan, Y. Al-Anii, J. Summers, H. Thompson, N. Kapur, and K. Djemame, "A unified model for holistic power usage in cloud datacenter servers," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, Dec. 2016, pp. 11–19.
[38] G. Dhiman, K. Mihic, and T. Rosing, "A system for online power prediction in virtualized environments using Gaussian mixture models," in *Proc. 47th Design Autom. Conf. (DAC)*, 2010, pp. 807–812.
[39] M. S. Inukonda, S. Mittal, and S. H. Kottapalli, "A solution architecture of bare-metal as a service cloud using open-source tools," IIT Hyderabad, Sangareddy, Telangana, Tech. Rep., 2019-CSE-CANDLE-02, 2019.
[40] S. Cook, C. Bock, P. Rivett, T. Rutt, E. Seidewitz, B. Selic, and D. Tolbert, *Unified Modeling Language (UML) version 2.5.1*, Standard formal/17-12-05, Object Manage. Group (OMG), Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1
[41] Canonical Corporation. *MAAS Concepts and Terms*. Accessed: Dec. 12, 2021. [Online]. Available: https://maas.io/docs/concepts-and-terms
[42] *Stress Ng*. Accessed: Dec. 12, 2021. [Online]. Available: https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html
[43] Sebastien Godard. *Multiprocessor Statistic Reporter*. Accessed: Dec. 12, 2021. [Online]. Available: https://man7.org/linux/man-pages/man1/mpstat.1.html
[44] H. L. Bijmans, T. M. Booij, and C. Doerr, "Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, 2019, pp. 1627–1644.
[45] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, "Thieves in the browser: Web-based cryptojacking in the wild," in *Proc. 14th Int. Conf. Avail., Rel. Secur.*, Aug. 2019, pp. 1–10.
[46] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, and H. Duan, "How you get shot in the back: A systematical study about cryptojacking in the real world," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1701–1713.
[47] D. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal analysis-based evasive malware detection," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 287–301.
[48] Correlation Coefficient. *Pearson and Spearman Correlation Coefficient*. Accessed: Dec. 12, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119454205.ch10
[49] T. Cover, "Estimation by the nearest neighbor rule," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 1, pp. 50–55, Jan. 1968.
[50] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[51] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 988–999, Sep. 1999.
[52] S. R. Gunn, "Support vector machines for classification and regression," *ISIS Tech. Rep.*, vol. 14, no. 1, pp. 5–16, 1998.
[53] InfluxDB Community. (2019). *InfluxDB Documentation*. [Online]. Available: https://docs.influxdata.com/

[54] Grafana Community. *Grafana Monitoring Solution*. Accessed: Dec. 12, 2021. [Online]. Available: https://grafana.com/
[55] Jenkins Community. *Jenkins Automation Server*. Accessed: Dec. 12, 2021. [Online]. Available: https://jenkins.io/
[56] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, Apr. 2011, doi: 10.1145/1961189.1961199.
[57] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[58] P. Mattson *et al.*, "MLPerf training benchmark," 2019, *arXiv:1910.01500*.
[59] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," 2019, *arXiv:1906.00091*.
[60] S. Mittal, P. Rajput, and S. Subramoney, "A survey of deep learning on CPUs: Opportunities and co-optimizations," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 21, 2021, doi: 10.1109/TNNLS.2021.3071762.
[61] (2013). *OpenBMC*. [Online]. Available: https: github.com/openbmc/openbmc
[62] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.
[63] B. Subramaniam, "Metrics, models and methodologies for energy-proportional computing," in *Proc. 14th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2014, pp. 575–578.
[64] C.-H. Hsu and S. W. Poole, "Measuring server energy proportionality," in *Proc. 6th ACM/SPEC Int. Conf. Perform. Eng.*, Jan. 2015, pp. 235–240.
[65] D. Wong, "Peak efficiency aware scheduling for highly energy proportional servers," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 481–492.

MARUTHI SESHIDHAR INUKONDA (Member, IEEE) received the M.Tech. degree in CSE from IIT Roorkee. He is currently pursuing the Ph.D. degree with the Department of Computer Science Engineering, IIT Hyderabad. He has about 13 years of systems programming experience in storage software product companies (Veritas, NetApp, EMC^2). He has two journal publications, and a patent in storage software on linux. His research interests include linux internals, storage software, software engineering, software architecture, computer architecture, continuous internet forensics, and datacenter infrastructures for cloud computing, high-performance computing, and big-data clusters. He is a Life-Time Member at Cloud Computing Innovation Council of India (CCICI) and Internet Society (ISOC). He is an Associate Member at the National Cyber Safety and Security Standards (NCSSS), India. He was a recipient of "Best cloud innovator of the year in cloud management" by CCICI at XaaS Cloud Conference, in 2021. In his free time, he does freelancing and mentoring at FOSS champs in upskilling students and professionals in free and open-source software used in datacenters.

ATHARVA RAJENDRA KARPATE received the B.E. degree in electronics and communication from SRCOEM, Nagpur, India, in 2020. He is currently pursuing the master's degree with Purdue University. He was interning at IIT Hyderabad during the research work. His research interests include cloud computing, embedded systems, computer networks, the Internet of Things, and wireless sensor networks.

BHEEMARJUNA REDDY TAMMA (Senior Member, IEEE) received the Ph.D. degree from IIT Madras, India, in 2007. Then, he worked as a Postdoctoral Fellow with the University of California at San Diego (UCSD) Division, California Institute for Telecommunications and Information Technology (CALIT2), prior to taking up a faculty position at IIT Hyderabad, in 2010. He is currently a Professor with the Department of Computer Science and Engineering, IIT Hyderabad, India. He has published over 100 articles in refereed international journals and conferences. His research interests include converged cloud radio access networks, SDN/NFV for 5G, network security, and green ICT. He was a recipient of Visvesvaraya Young Faculty Research Fellowship at IIT Hyderabad and iNautix Research Fellowship for his Ph.D. tenure at IIT Madras. He is a co-recipient of Top Cited Article Award from Elsevier publishers, the Best Academic Demo Award at COMSNETS 2018, the Best Poster Award at ICACCI 2018, the 2nd Best Paper Award at IEEE ANTS 2017, and the Best Paper award at ICACCI 2015 conferences. He is a member of ACM and served as the General Co-Chair for National Conference on Communications (NCC) 2018, the TCP Co-Chair for IEEE ANTS 2015, the TCP Vice Chair for IEEE ANTS 2014, and a Ph.D. student Forum Co-Chair for IEEE ANTS 2013 conferences.

SPARSH MITTAL (Senior Member, IEEE) received the B.Tech. degree from IIT, Roorkee, India, and the Ph.D. degree from Iowa State University (ISU), USA. He worked as a Postdoctoral Research Associate with the Oak Ridge National Laboratory (ORNL), USA. He was as an Assistant Professor at IIT Hyderabad. He is currently working as an Assistant Professor with the Department of Electronics and Communication Engineering, IIT Roorkee, and the Joint Faculty with the Mehta Family School of Data Science and Artificial Intelligence. He was the graduating topper of his batch in B.Tech. degree and has received fellowship from ISU and performance award from ORNL. He has published more than 100 research papers at top venues. His research has been covered by insideHPC, HPCwire, Phys.org, and scientific computing. He has given invited talks at the ISC Conference at Germany, New York University, and the University of Michigan. He is also an Associate Editor of *Journal of Systems Architecture*. He has received research funding from SERB, Intel, and SRC (USA).

PRAVEEN TAMMANA received the Ph.D. degree from The University of Edinburgh. He is currently an Assistant Professor in computer science with IIT-Hyderabad. Prior to IITH, he worked as a Postdoctoral Researcher with Princeton University. His research interests include the intersection of networking, security, and machine learning for networks.