Optimizing IT FinOps and Sustainability through Unsupervised Workload Characterization

Xi Yang[1], **Rohan R. Arora**[1], **Saurabh Jha**[1], **Chandra Narayanaswami**[1], **Cheuk Lam**[2], **Jerrold Leichter**[2], **Yu Deng**[1], **Daby M. Sow**[1]
[1]IBM Thomas J. Watson Research Center
[2]IBM Software
{xi.yang, rohan.arora, saurabh.jha}@ibm.com, chandras@us.ibm.com,
{cheuk.lam, jerry.leichter}@ibm.com, {dengy, sowdaby}@us.ibm.com

Abstract

The widespread adoption of public and hybrid clouds, along with elastic resources and various automation tools for dynamic deployment, has accelerated the rapid provisioning of compute resources as needed. Despite these advancements, numerous resources persist unnecessarily due to factors such as poor digital hygiene, risk aversion, or the absence of effective tools, resulting in substantial costs and energy consumption. Existing threshold-based techniques prove inadequate in effectively addressing this challenge. To address this issue, we propose an unsupervised machine learning framework to automatically identify resources that can be de-provisioned completely or summoned on a schedule. Application of this approach to enterprise data has yielded promising initial results, facilitating the segregation of productive workloads with recurring demands from non-productive ones.

Introduction

Infrastructure as Code (IaC) and IT automation tools have greatly enhanced the convenience and flexibility of provisioning a variety of IT resources, such as virtual machines (VMs), Kubernetes/OpenShift clusters, and database-as-a-service (DBaaS) instances. Also, these tools have simplified the deployment of applications on or with the aforementioned resources. However, an unintended consequence of this automation is the proliferation of redundant resources. This issue has sparked attention towards resource optimization, financial operations (FinOps) (Bryant 2022), and the reduction of carbon footprints in hybrid clouds.

Several prior approaches (CloudWatch 2013; VMWare 2020; Google 2023) identify idle workloads, which we consider a subset of non-productive workloads, via threshold-based methods. In these approaches, thresholds are manually predefined based on domain knowledge. However, their common challenge is the utilization of excessively conservative thresholds, leading to the oversight of a significant number of non-productive workloads. Recently, Duan et al. employed a supervised learning technique, constructing a binary classifier, to distinguish non-productive workloads from productive ones (Duan et al. 2019). However, it requires extensive manually tagged data, which can be time-consuming and labor-intensive to acquire.

In this paper, we tackle these challenges by introducing a comprehensive, data-driven, fully unsupervised framework. Our framework characterizes workloads in two phases: inactive and active. By leveraging the phase abstraction, the workloads can be categorized into three distinctive types: 1) non-productive, persisting in the inactive phase; 2) constantly productive, consistently remaining active; and 3) alternating productive, intermittently switching between the two phases. For the alternating workloads, we further measured whether there are repeatable patterns within a specified time unit (e.g., daily, weekly). Timetabling can then be derived accordingly to optimize resource consumption.

Productivity is intricate and varies across different scenarios (Kim et al. 2017). For certain workloads, productivity could be quantified by the number of transactions completed or web requests processed, while for others, its significance lies in facilitating a specific workflow or process. Thus, explicitly elaborating rules for characterizing workload patterns is challenging. Recent advancements in data acquisition tools have enriched available data, providing the potential to derive more comprehensive and automated workload characterization. Metrics data collected by Application Performance Management (APM) tools, as well as service topology information, can contribute to a holistic understanding of productivity. As an emerging work, in this paper, we primarily focus on resource utilization metrics, while our framework is easily adaptable to more extensive datasets.

The characterization of workloads offers the potential to devise suitable strategies for minimizing resource consumption. For instance, non-productive workloads could be identified as candidates for permanent removal or deletion, with the option to archive associated storage if necessary. Conversely, active workloads with recurring demands could be dynamically scaled up or down, powered off during periods of inactivity, or migrated to a serverless architecture. Given that VMs continue to constitute a significant portion of enterprise data centers, our initial investigation focuses on VMs, while the experimental findings provide insights into expanding our framework to encompass other workload types and even the applications running on them.

Problem Statement and Framework

To elevate FinOps practices and foster workload sustainability within enterprise-grade VM-based applications, cloud-

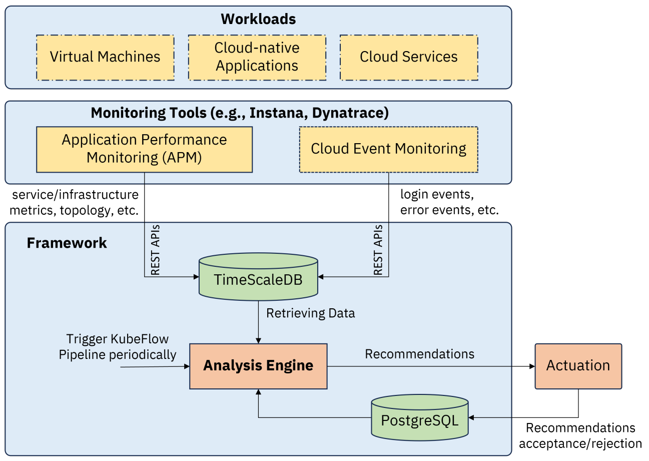

Workloads
Virtual Machines
Cloud-native Applications
Cloud Services
Monitoring Tools (e.g., Instana, Dynatrace)
Application Performance Monitoring (APM)
Cloud Event Monitoring
service/infrastructure metrics, topology, etc.
login events, error events, etc.
Framework
REST APIs
REST APIs
TimeScaleDB
Retrieving Data
Trigger KubeFlow Pipeline periodically
Analysis Engine
Recommendations
Actuation
PostgreSQL
Recommendations acceptance/rejection

Figure 1: Framework Overview

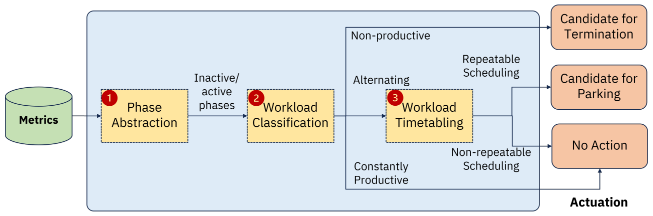


Figure 2: Analysis Engine

native applications, and cloud services, we designed a workload characterization framework, as illustrated in Figure 1.

Our workload characterization is rooted in the data monitored by APM tools (e.g., service or infrastructure metrics, topology) and, when accessible, cloud event monitoring services (e.g., login events, error events). For each monitoring tool, corresponding serverless applications are established. These applications function periodically, extracting essential utilization and service metrics, as well as event details, using REST APIs from the monitoring tools. The gathered data is then stored within TimeScaleDB (Instana 2023; Sysdig 2023), a database optimized for time-series data.

The data retrieved from TimescaleDB can be fed into an *analysis engine* for workload pattern analysis. This engine, depicted in Figure 2, consists of three components:

1. *Phase abstraction:* Serving to partition the time periods and characterize them as active or inactive phases.
2. *Workload classification:* Categorizing workloads into 1) non-productive (consistently inactive), 2) consistently productive (maintaining active), and 3) alternating productive (switching between active and inactive).
3. *Workload timetabling:* Devising a timetable for alternating workloads, provided they exhibit repeatable patterns.

The analysis engine can be periodically triggered via a Kubeflow pipeline. Leveraging our framework, organizations can unlock their insights into workload status, fostering informed actuation (as illustrated in Figure 1) to optimize resource allocation and operational effectiveness.

Methodology

In this section, we mainly focus on the *analysis engine*, where we design three integral components—phase abstraction, workload classification, and workload timetabling—for workload characterization, as shown in Figure 2.

Phase Abstraction

Subsequence Clustering To effectively capture the evolving patterns of workloads throughout their recorded time span, we approached the task as a subsequence clustering problem. For this purpose, we employed a Multi-series Toeplitz Inverse Covariance-based Clustering (M-TICC) algoriathm (Yang 2021), which is adept at automatically learning subsequence clusters in an unsupervised manner.

Taking multivariate time-series $\{\mathbf{x}_t^n | t \in [1, T^n]\}$ as input, where $n \in [1, N]$ is the index of workloads, and T^n is the number of timestamps for the n-th workload, our goal is to simultaneously *partition and cluster* the subsequences based on their latent time-invariant patterns, by learning a mapping from each timestamp $\mathbf{x}_t^n$ to a certain cluster $\{k | k \in [1, K]\}$. Considering the interdependence of neighboring states, instead of treating each state independently, we explored the patterns within a sliding window $\omega \ll T^n$. The $\mathbf{X}_t^n$ is represented as a $M\omega$-dimension random variable (obtained by concatenating the M-dim metrics of $\mathbf{X}_t^n$ in ω) and will be fit into K Gaussian distributions, with the k-th distribution corresponding to the k-th cluster.

Determining the mean vectors $\{\mu_k | k \in [1, K]\}$ is equivalent to matching each timestamp to a cluster, yielding clustering assignments $\mathbf{P} = \{P_k | k \in [1, K]\}$, where $P_k \subset \{1, ..., T^n | n \in [1, N]\}$ denotes the indices of timestamps (sliding windows) belonging to the k-th cluster. Meanwhile, the inverse covariance matrices $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_k | k \in [1, K]\}$ aims to characterize time-invariant patterns within each cluster. $\boldsymbol{\Theta}_k$ is constrained to be block-wise Toeplitz, comprising ω sub-blocks $A^{(i)} \in \mathbb{R}^{M \times M}$, $i \in [0, \omega - 1]$, with the objective of capturing the partial correlations of M features within a timestamp and across different timestamps. The objective function of M-TICC is defined as:

$$\arg\min_{\boldsymbol{\Theta},\mathbf{P}} \sum_{k=1}^{K} \Bigg[\sum_{n=1}^{N} \sum_{\mathbf{X}_t^n \in P_k} \Bigg(\overbrace{-\ell\ell(\mathbf{X}_t^n, \boldsymbol{\Theta}_k)}^{\text{Log-likelihood}} + \beta \overbrace{\mathbb{1}\{t-1 \notin P_k\}}^{\text{Consistency}} \Bigg) + \lambda \overbrace{||\boldsymbol{\Theta}_k||_1}^{\text{Sparsity}} \Bigg], \tag{1}$$

which consists of three terms detailed as follows:

1. *Log-likelihood* that $\mathbf{X}_t^n$ belongs to cluster $\boldsymbol{\Theta}_k$.
2. *Consistency* encouraging neighbored events $\{\mathbf{X}_{t-1}, \mathbf{X}_t\}$ to be assigned into the same cluster due to temporality.
3. *Sparsity* controls sparseness of $\boldsymbol{\Theta}_k$ to prevent overfitting.

Herein, β and λ are regularization coefficients. They are determined by cross validation as 4 and 1e-5, respectively.

Abstracting Clusters as Inactive and Active Phases Given the K clusters learned by M-TICC, we further abstracted them as two phases (*Inactive* and *Active*), denoted as $\mathbf{Ph} = \{Ph_c | c \in \{I, A\}\}$, where $Ph_c \subset \{1, ..., T^n | n \in$

$[1, N]\}$. I and A indicate *Inactive* and *Active*, respectively. For each metric $m \in [1, M]$, we computed cluster-wise median value over $\{\mathbf{x}_t^n[m] | t \in P_k\}$, among which we pick the cluster with the minimal median value. By doing so, we designated each metric with a cluster, and then a majority voting was employed: If the k-th cluster possesses the most minimal medians across different metrics, it is assigned as an inactive phase Ph_I; Otherwise, the cluster falls into the active phase Ph_A. Our goal was to abstract the cluster with the smallest metrics as inactive while all other clusters as active, considering that the inactive phase is in general stable, while the active phase usually processes diverse patterns.

Workload Classification

Given the phase abstraction results, we categorized workloads into three distinctive classes based on their historical phases over the preceding τ days, where τ is usually set as 7 in prior studies (CloudWatch 2013; VMWare 2020) while applying the threshold-based rules. When time is measured in hours, the tracing back window is denoted as $\omega_c = \tau * 24$. The three categories of workloads are elucidated as follows.

1. *Non-productive:* Remaining in the inactive phase, i.e., $t \in Ph_I, \forall t \in [T - \omega_c, T]$;
2. *Constantly productive:* Remaining in the active phase, i.e., $t \in Ph_A, \forall t \in [T - \omega_c, T]$;
3. *Alternating:* Switching between the two phases, i.e., $(t \in Ph_I, \exists t \in [T - \omega_c, T]) \wedge (t \in Ph_A, \exists t \in [T - \omega_c, T])$.

The workload classification outcomes offer valuable insights for well-targeted actions to optimize resource consumption. Specifically, *non-productive* workloads are potential candidates for termination to enhance efficiency. On the other hand, *constantly productive* workloads may not require immediate intervention and can continue their operations without specific actions. For the *alternating* workloads, more intricate analyses are necessary. Therefore, we developed a timetabling approach, detailed in the next section.

Workload Timetabling

For *alternating* workloads, we investigated whether they exhibit repeatable scheduling patterns with a certain periodicity (e.g., daily, weekly). Let pu denote possible time values with a specific unit for a periodicity (e.g., $pu \in [0, 23]$ for daily periodicity with an hourly unit, $pu \in [1, 7]$ for weekly periodicity with a daily unit). For each alternating workload, we computed the probability of it being inactive at pu:

$$\text{prob}_I(pu)^n = \frac{\mathbb{1}(\lfloor t \rceil = pu \wedge t \in Ph_I)}{\sum_{c=\{I,A\}} \mathbb{1}(\lfloor t \rceil = pu \wedge t \in Ph_c)}, \quad (2)$$

where $t \in [1, T^n]$. $n \in [1, N^A]$ is the index of alternating workloads. Moreover, we measure the frequency with which it switches to active phase along the horizon $t \in [1, T^n - 1]$:

$$\text{freq}_{IA}^n = \frac{\mathbb{1}[(t \in Ph_I) \wedge (t+1 \in Ph_A)]}{T^n / Z}, \quad (3)$$

where Z is a normalizer. When time is measured in hours, Z is 24 for daily periodicity and 7*24 for weekly periodicity.

Taking the daily periodicity as an example, where $pu \in [0, 23]$ has hours as time unit, the values of prob$I(pu^n)$ and freq$_{IA}^n$ can be employed to determine the timetabling for an alternating workload n. For instance, if freq$_{IA}^n > 2$, indicating that the active phase occurs more than twice per day, no parking suggestions would be recommended, since frequent transitions between active and inactive phases make parking less advantageous. Conversely, if freq$_{IA}^n < 1/7$, indicating the active phase occurs less than once per week, daily periodic patterns can be hardly captured, and thus, no parking suggestion would be given. Otherwise, if prob$I(pu^n)$ remains consistently high for an extended period, it is considered as non-productive during that time, and parking suggestions would be offered accordingly. This involves creating a schedule that aligns with fluctuating demands, reducing allocated resources during inactive phases, and potentially turning off workloads when demand is low. A serverless design may be suitable for implementing the timetabling.

Dataset

Our data was collected from thousands of VMs in an active enterprise IT operational environment, representing diverse production workloads. Drawing from existing idle VM detection methods (CloudWatch 2013; VMWare 2020; Google 2023), we mainly focused on the metrics shown in Table 1.

The utilization metrics are computed by dividing actual usage by the maximum capacity (allocation). VMEM is between 0 and 1. Since the fraction of these VMs are burstable, VCPU can reach up to 1.5. We selected VMs with recorded resource utilization metrics available for $\sim$30 days—77.6% of the VMs—to allow us to gain insights into long-term resource utilization patterns and make informed decisions to optimize resource allocation and enhance efficiency.

Experiments

Phase Abstraction

To validate the effectiveness of subsequence clustering in phase abstraction, we analyzed the distribution of metrics across different clusters. The cluster number K is determined as 4, based on discussions detailed in the next section. The clustering results are illustrated in Figure 3. To better distinguish the Clus_1, Clus_2, and Clus_3 in NET and IO, we display their medians in Table 2. It is evident that Clus_1 consistently exhibits lower metric values compared to other clusters, indicating it is relatively more inactive.

Based on Table 2, for each metric, we compared the median values and selected the cluster with the minimal value. Afterward, majority voting led us to designate Clus_1, with

Metric	Description
VCPU_UTIL	CPU utilization = CPU usage/capacity
VMEM_UTIL	Memory utilization = Memory usage/capacity
NET	Network throughput in kB/s
IO	I/O throughput in kB/s

Table 1: Metric Description.

Clus_Idx	VCPU	VMEM	NET	IO
1	0.0110	0.1747	1	8
2	0.0788	0.5017	12	28
3	0.2989	0.4350	18	39
4	0.2142	0.4241	10895	178

Table 2: Cluster-wise median for each metric.

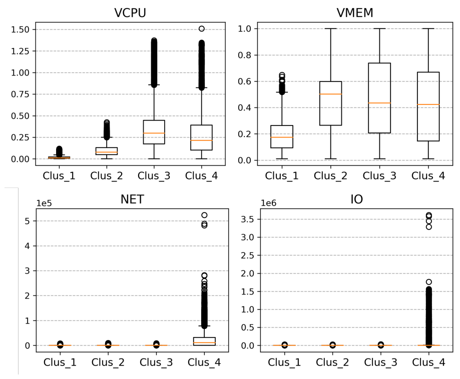


Figure 3: Cluster-wise distribution for each metric.

the most minimal medians, as the *inactive* phase, while considering all other clusters as the *active* phase. Across all VMs, most of their timestamps fall into the inactive phase, as delineated in Figure 4, i.e., a histogram (% of VMs) for the percentages of active timestamps (i.e., the number of active timestamps divided by the total timestamps) per VM.

Workload Classification

Based on the phase abstraction results, we classify VMs into three categories. Herein, we set $\tau = 7$ following prior works (CloudWatch 2013; VMWare 2020). Figure 5 is a histogram (% of VMs) for the percentages of active phases (i.e., number of active phases divided by the total number of phases) per VM. Among the given VMs, 35.6% are categorized as non-productive, 27.2% are alternating, and 37.2% are constantly productive. Due to the lack of ground-truth labels, it is hard to directly evaluate this characterization results. Prior literature (Koomey and Taylor 2015; James M., Forrest, and Kindler 2008; Duan et al. 2019) has suggested that more than 30% of enterprise data centers is "comatose"—not performing productive work. Hence, our identification of

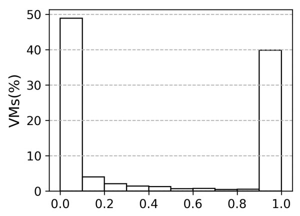


Figure 4: Histogram: number of active timestamps divided by total number of timestamps

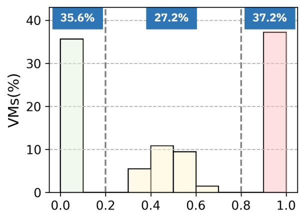


Figure 5: Histogram: number of active phases divided by total number of phases

VM Category	VCPU	VMEM	NET	IO
Non-productive	0.0080	0.1470	1	6
Alternating	0.0263	0.2275	2	11
Constantly Productive	0.1134	0.5275	23	35

Table 3: Category-wise median for each metric.

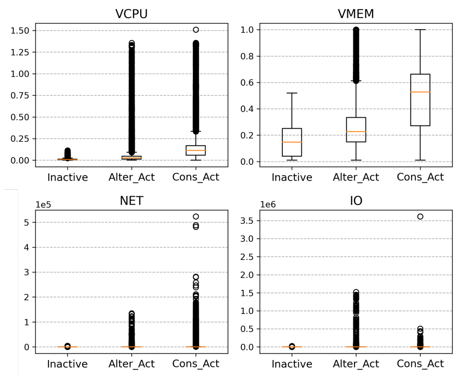


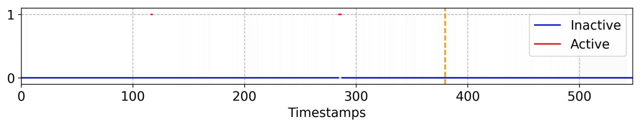
Figure 6: Category-wise distribution for each metric.

35.6% non-productive turns out to be a reasonable number. In comparison, using threshold-based approaches (CloudWatch 2013; VMWare 2020; Google 2023), which rely on predefined thresholds derived from prior knowledge, we are only able to identify 1.1% as non-productive. All of these VMs identified by the threshold-based methods are also categorized as non-productive by our approach. This finding suggests that existing threshold-based methods may use overly conservative thresholds, resulting in a considerable number of non-productive VMs being overlooked. Our approach, which is more data-driven, has the potential to identify a larger proportion of non-productive VMs.

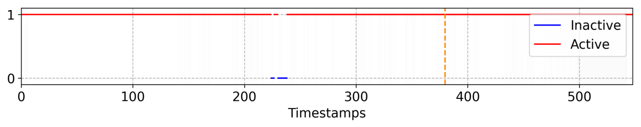
The distributions of metrics for the three types of VMs are shown in Figure 6, and their median values are displayed in Table 3. It is evident that the non-productive VMs in general have lower levels of resource utilization compared to other two types, reflected by the smaller median values, while it is not always the case considering all their timestamps. This observation aligns with the suggestions put forth in (Kim et al. 2017) that the requirement idleness may not always be reflected as resource idleness. For instance, non-productive VMs might appear productive when they are engaged in activities such as virus scans or system updates.

The workload classification results are depicted in Figure 7, showcasing the characterization of three types of VMs: 1) non-productive, 2) constantly productive, and 3) alternating. The x-axis represents timestamps, and the y-axis signifies the two phases: 0 denotes inactive, and 1 denotes active. The data used for classification spans the last 7 days (i.e., $\tau = 7$), as indicated by the orange dotted line in the figure. Specifically, the non-productive VM consistently remains in the inactive phase (0), the constantly productive VM remains in the active phase (1), and the alternating VM demonstrates intermittent transitions between the two phases (0 and 1).

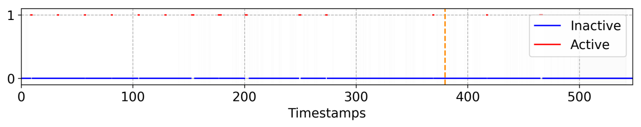
To select an optimal number of subsequence clusters, we varied K from 2 to 8 as shown in Figure 8 and measured:



(a) Non-productive



(b) Constantly productive



(c) Alternating

Figure 7: Illustration of the two phases for three VM classes.

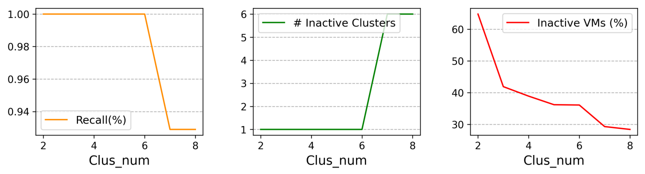


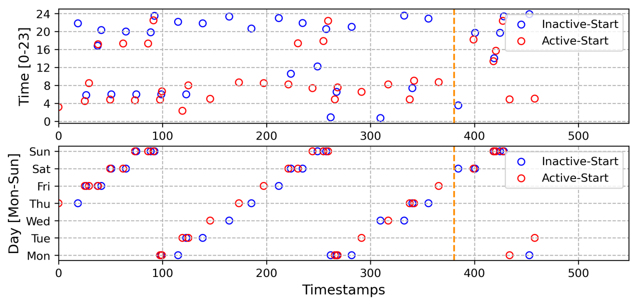
Figure 8: Selecting the optimal cluster number.

1. *Recall(%):* Indicating that among the VMs identified by the conservative threshold-based methods, how many of them can be accurately classified by our method.
2. *Number of Inactive Clusters:* Indicating how many clusters the threshold-based methods identified VMs belong to. A lower count is preferred to prevent non-productive VMs from being dispersed across multiple clusters.
3. *Non-productive VMs(%):* Indicating the percentage of VMs identified as non-productive.

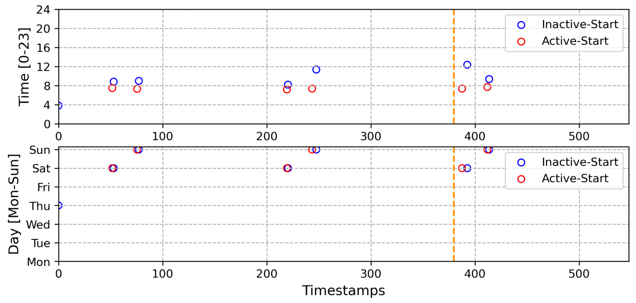
Taking all above factors into account, we set $K = 4$.

Workload Timetabling

For alternating VMs, we conducted further analysis for their repeatable scheduling patterns. Figure 9 illustrates this process for two distinct VMs, with daily and weekly periodicity, respectively. For each of them, we check the start clock time ([0-23]) during a day and start day ([1-7]) during a week for the two phases. For VM_1, the frequency of switching to active phase is $\sim$1.14 per day, indicating an almost daily periodicity. When assessing the probabilities of remaining in the inactive phase, we observed consistent values above 90% from 10 PM until 7 AM the following day. Consequently, we recommend parking the VM during this period and resuming its usage when it becomes active again, thereby optimizing resource consumption. For VM_2, the frequency switching to active phase is $\sim$1.82 per week, indicating a periodicity nearly occurring almost twice per week—specifically on Saturday and Sunday. Besides, the VM tends to switch to the active phase during weekends, when we identified a high probability of staying in the inactive phase, exceeding 90% from 10 PM until 6 AM on the subsequent day. Thus, we suggest parking during this timeframe as well as on weekdays, offering further potential for resource optimization.



(a) VM_1: Daily periodicity



(b) VM_2: Weekly periodicity

Figure 9: Illustration of the timetabling.

Acting on the Recommendations

For VMs categorized as non-productive, once the recommendation is accepted by the SRE or application owner teams, we expect an automation engine to capture an image of the VM and shut it down. This creates an opportunity for VMs to be consolidated, which in turn creates the possibility for the low-utilized hosts to either be switched to a low power consumption mode (if available) or turned off. Similarly for VMs deemed alternating, we expect SREs or application owner teams to incorporate the recommendation into products such as ParkMyCloud [1], which would then suspend the VM per the schedule defined. As we are in the early stages of building this technology, we expect the SRE or application teams to accept or deny the recommendation before an automation engine is invoked to take the necessary action. For constantly productive VMs, we don't expect the SRE or application owner teams to take any action.

Related Work

Phase Abstraction

To characterize workload patterns, (Janecek, Ezzati-Jivan, and Azhari 2021) proposed using host system tracing, which requires specific tooling like LTTng to tap into kernel-level events. To track the phases automatically, Sherwood et

[1]https://api.parkmycloud.com/

al. presented an architecture specifically for executed code (Sherwood, Sair, and Calder 2003). Also, some techniques based on machine learning have been proposed (Khanna et al. 2014; Bhattacharyya, Sotiriadis, and Amza 2017; Jandaghi, Bhattacharyya, and Amza 2018). For example, to detect workload phases for inspiring more accurate resource provisioning, (Berral, Wang, and Youssef 2020a) proposed combining conditional restricted Boltzmann machines and distance-based clustering (i.e., k-means) to discover behavioral phases from resource usage metrics for auto-scaling. Recently, (Hossain et al. 2022) presented an ML-based phase detection to identify uncommon, unknown customer-specific workloads, where Bayesian change point detection combined with distance-based clustering (i.e., Euclidean distance with dynamic time warping) is employed to generate fingerprints for different workloads. The above two methods rely on distance-based clustering methods, which are known for not being able to handle noise and outliers (Thrun 2021). The *model-based* clustering methods can address these issues by robustly learning a generative statistical model for each cluster. Therefore, in this work, we employ a *model-based* M-TICC (Yang 2021) for phase abstraction.

Workload Classification

Cloud service providers, such as Amazon Web Services (AWS) (CloudWatch 2013) and Google Cloud Platform (GCP) (Google 2023), along with virtualization providers, have developed tools to assist customers in identifying idle workloads based on resource utilization metrics. For VMs, these tools typically rely on static thresholds for metrics like CPU utilization, the number of Disk I/O operations, and network throughput observed over a period of 7 days or more. For example, in AWS CloudWatch, an EC2 instance is labeled as "idle" if, over the last 7 days, both of the following conditions are met: (a) average CPU utilization $< 2\%$; and (b) average network I/O < 5 MB. The threshold values themselves are conservative, thus only a small subset of VMs could be classified as non-productive.

Workload Timetabling

To use computing resources efficiently for both cost and sustainablity reasons, (Berral, Wang, and Youssef 2020b) combined conditional restricted Boltzmann machines and clustering techniques to discover common sequences of behaviors (phases) of container workloads, particularly those for deep learning (DL) applications. Statistical information from each phase is used to tune resources allocated to the container in each phase. They report an overall reduction in resource utilization. Along similar lines, (Gao et al. 2022) explored research on scheduling for DL applications in GPU data centers. In (Buchbinder et al. 2020), the authors developed a methodology to predict VM loads and then use it for scheduling VMs to physical hosts. Their approach models scheduling as a generalized static bin packing problem.

Discussions and Future Work

Prior to sharing our recommendations for the entire corpus with the respective application teams, we requested explicit feedback on 41 VMs—a smaller subset that were deemed non-productive by our approach. For 9 of the 41, the application owners initiated the process to decommission and sunset the VMs. For 4 of the 41, the application owners initiated a re-size action. Eighteen were deemed necessary by the application owners. Four of the necessary 18 VMs are conducting testing tasks (e.g., patch testing) and hence deemed essential. For the remaining 14 of the necessary 18 VMs, the application owners are considering consolidation actions. We are continuing to work with the application teams to wrap up this initial evaluation.

Based on this initial feedback, we are actively pursuing the following next steps to improve our recommendations:

1. *Shifting to absolute metrics:* Instead of leveraging CPU and memory utilizations (%), we will explore absolute values. When systems are over-provisioned, low utilization (%), as seen in those 4 cases, may be interpreted incorrectly as an indicator of inactivity.
2. *Extending metric scope:* While current metrics allow us to abstract workload phases, incorporating extra metrics, e.g., latency, will provide a more comprehensive insight into resource consumption and workload behavior.
3. *Gathering implicit feedback:* Collaborating with application owners, we are deploying additional tools to VMs, capturing snapshots of process info, login activity, active ports, and network connections every 2 hours. This enables the development of an implicit feedback mechanism, enhancing recommendation quality.
4. *Learning from user actions:* Implementing mechanisms to track user actions in response to recommendations, accompanied by an opportunity to explain their decisions.

In addition to the steps discussed above, we intend to undertake the following extensions:

1. *Enhancing relatability:* Recommendations should be relatable to Site Reliability Engineers and System Administrators, reducing perceived risk in taking action.
2. *Addressing cloud-native deployments and services:* The growth of SaaS deployments and container-based technologies calls for similar management technologies.
3. *Exploring other unsupervised ML techniques:* We aim to enhance phase abstraction by exploring alternative unsupervised ML techniques and leveraging ensemble learning for more accurate inactive/active phase identification.
4. *Refining repeatable patterns and recommendations:* While our current recommendations are periodically repeatable, we aim to detect irregular workload behaviors and capture finer-grained patterns to enhance precision.

During early stages of deployment, we expect the pipeline to run as a back-end process once a week. Specifically:

1. Delta metrics since the last run are extracted for in-scope workloads (or all workloads).
2. Phase abstraction and workload classification are re-run, resulting in recommendations (either terminating non-productive workloads or scheduling alternating ones).

With quite a few organizations leveraging digital workflow management tools for their enterprise operations, we expect these action recommendations to flow into such tools.

References

Berral, J. L.; Wang, C.; and Youssef, A. 2020a. AI4DL: Mining Behaviors of Deep Learning Workloads for Resource Management. In *HotCloud*.

Berral, J. L.; Wang, C.; and Youssef, A. 2020b. AI4DL: Mining Behaviors of Deep Learning Workloads for Resource Management. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association.

Bhattacharyya, A.; Sotiriadis, S.; and Amza, C. 2017. Online phase detection and characterization of cloud applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 98–105. IEEE.

Bryant, J. 2022. Driving into the Cloud: What is Finops? *ITNOW*, 64(3): 54–55.

Buchbinder, N.; Fairstein, Y.; Mellou, K.; Menache, I.; Joseph; and Naor. 2020. Online Virtual Machine Allocation with Predictions. arXiv:2011.06250.

CloudWatch. 2013. Use Amazon CloudWatch to Detect and Shut Down Unused Amazon EC2 Instances. https://aws.amazon.com/about-aws/whats-new/2013/01/08/use-amazon-cloudwatch-to-detect-and-shut-down-unused-amazon-ec2-instances/. Accessed: 2023-12-14.

Duan, J.; Li, G.; Asthana, N.; Zeng, S.; Dell'Era, I.; Chanana, A.; Agastya, C.; Pointer, W.; and Yan, R. 2019. CSI2: Cloud Server Idleness Identification by Advanced Machine Learning in Theories and Practice. In *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*, 243–248. Springer.

Gao, W.; Hu, Q.; Ye, Z.; Sun, P.; Wang, X.; Luo, Y.; Zhang, T.; and Wen, Y. 2022. Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision. arXiv:2205.11913.

Google. 2023. Idle VM Recommendations in Google Cloud. https://cloud.google.com/compute/docs/instances/idle-vm-recommendations-overview. Accessed: 2023-12-14.

Hossain, M.; Mebratu, D.; Hasabnis, N.; Jin, J.; Chaudhary, G.; and Shen, N. 2022. CWD: A Machine Learning based Approach to Detect Unknown Cloud Workloads. *arXiv preprint arXiv:2211.15739*.

Instana. 2023. Data Retention Policy in Instana. https://www.ibm.com/docs/en/instana-observability/current?topic=policies. Accessed: 2023-12-14.

James M., K.; Forrest, W.; and Kindler, N. 2008. Revolutionizing Data Center Energy Efficiency. https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf. Accessed: 2023-12-14.

Jandaghi, S. J.; Bhattacharyya, A.; and Amza, C. 2018. Phase annotated learning for apache spark: Workload recognition and characterization. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 9–16. IEEE.

Janecek, M.; Ezzati-Jivan, N.; and Azhari, S. V. 2021. Container workload characterization through host system tracing. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 9–19. IEEE.

Khanna, R.; Ganguli, M.; Narayan, A.; Abhiram, R.; and Gupta, P. 2014. Autonomic characterization of workloads using workload fingerprinting. In *2014 IEEE international conference on cloud computing in emerging markets (CCEM)*, 1–8. IEEE.

Kim, I. K.; Zeng, S.; Young, C.; Hwang, J.; and Humphrey, M. 2017. iCSI: A cloud garbage VM collector for addressing inactive VMs with machine learning. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, 17–28. IEEE.

Koomey, J.; and Taylor, J. 2015. New data supports finding that 30 percent of servers are 'Comatose', indicating that nearly a third of capital in enterprise data centers is wasted. https://www.anthesisgroup.com/wp-content/uploads/2019/11/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf. Accessed: 2023-12-14.

Sherwood, T.; Sair, S.; and Calder, B. 2003. Phase tracking and prediction. *ACM SIGARCH Computer Architecture News*, 31(2): 336–349.

Sysdig. 2023. Data Retention Policy in Sysdig. https://docs.sysdig.com/en/docs/administration/data-retention/. Accessed: 2023-12-14.

Thrun, M. C. 2021. Distance-based clustering challenges for unbiased benchmarking studies. *Scientific reports*, 11(1): 18988.

VMWare. 2020. vRealize Operations Report: Identify Idle VMs so that Resources Can be Reclaimed. https://digitalthoughtdisruption.com/2020/05/01/vrealize-operations-report-identify-idle-vms-so-that-resources-can-be-reclaimed/. Accessed: 2023-12-14.

Yang, X. 2021. Multi-series Time-aware Sequence Partitioning for Disease Progression Modeling. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*.