PolarisProfiler: A Novel Metadata-Based Profiling Approach for Optimizing Resource Management in the Edge-Cloud Continnum

Andrea Morichetta†, Víctor Casamayor Pujol†, Stefan Nastic†, Schahram Dustdar†,
Deepak Vij*, Ying Xiong*, Zhaobo Zhang*
†TU Wien surname@dsg.tuwien.ac.at, *Futurewei Technologies, Inc.

***Abstract*—Resource provisioning is vital in large-scale, geographically distributed, and hierarchically organized infrastructures, and, at the same time, it represents one of the stiffest challenges in their management. The goal is to optimally allocate infrastructure resources to jobs, ensuring jobs' Service Level Objectives (SLOs) while retaining high resource utilization across the entire resource pool. In this context, accurate workload profiling is crucial to achieving optimal resource management, giving more context to the system. However, approaches either make static guesses or use runtime profiling – that may be delayed by sandbox testing – and fall short in providing fast and accurate information. We aim to overcome these challenges with a novel profiling approach and methodology, the PolarisProfiler. We discard the consistency assumptions and assume a broader and less influenced perspective. We use apriori available, static metadata to enable generic and immediate job profiling based on historic execution traces. The PolarisProfiler proposes a novel dynamic profiling model, a generic workload profile generator, and a metadata-based profile classifier. We illustrate the practical feasibility of our approach by evaluating the PolarisProfiler in a case study. We target machine learning workloads, leveraging a publicly available dataset from Alibaba. We offer a reference implementation of our profiling methodology, combining a density-based hierarchical clustering technique and an interpretable decision-tree model for the classifier. We test the PolarisProfiler for job duration estimation. Despite being based solely on static, apriori metadata, we obtain convincing results compared to the state-of-the-art, yielding an estimation error rate of 5% for the 80% of profiled jobs.**

I. Introduction

Optimal resource provisioning in virtualized and shared computing infrastructures is one of the main challenges faced by infrastructure providers and operators [1]. The core question is how to enforce Service Level Objectives (SLOs) while retaining high resource utilization across the entire resource pool, whether in a Kubernetes cluster in the Cloud or in a KubeEdge [2] cluster at the Edge. Current resource provisioning and management techniques try to address challenges such as workload scheduling and placement, resource overcommitment and oversubscription models [3], resource bursting [4], and live migration. Workload characterization plays an important role in many of the solutions trying to facilitate resource provisioning in the Edge-Cloud continuum. The most notable technique for workload characterization is so-called workload profiling.

Workload profiling: state of the art and limitations. The main approaches to workload profiling can be classified into one of two general categories: (a) they either attempt to exploit available information about past workload executions (historical data) to learn the workload's characteristics or (b) they attempt to collect information about the workload's properties by actively observing it - usually by running the workload in a sandbox and probing it with synthetic traffic. Furthermore, they make at least one of the following assumptions: (i) *Environment consistency* - that is, they assume that the sandboxed execution environment used for profiling faithfully resembles the production execution environment; (ii) *Performance consistency* - that is, the runtime performance of similar workloads will remain consistent over time. (iii) *Time consistency* - that is, there are no time constraints on how long it takes to make profiling decisions, i.e., the workload profile can be created ad hoc when needed; (iv) *Occurrence consistency* - that is, the same workload will run multiple times, and it will reoccur in the same shared computing environment in the (near) future.

Unfortunately, these assumptions typically do not hold in practice. (i) The execution environment is typically inconsistent across multiple workload runs. The primary reason is the infrastructure heterogeneity resulting from software and hardware updates, such as adding a new generation processor [5]. Additionally, due to phenomena known as "noisy neighbors," the existing physical resources available in a host node can significantly vary. This scenario can cause significant variance in workload performance, rendering the profiles useless. (ii) Further, several authors have pointed out that the runtime performance of similar workloads is not consistent during their lifetime. It typically varies with time, even if the same preconditions are met, such as using the same input data [6], [7]. (iii) The time allocated to the profiler to generate the workload's profile can significantly vary. It is use-case specific and typically inconsistent for different resource provisioning techniques. For example, time spent profiling a workload while it is pending to be scheduled must be orders of magnitude shorter than profiling a workload to prevent a bootstrapping problem when predicting SLO violations. (iv) Finally, previous work has shown that most general-purpose workloads are recurrent only to a limited degree, that is, only between 40% and 60% of workloads are reported to be recurrent [8], [7], [9]. By only looking at a single workload's history, approximately every other workload will fail to be successfully profiled.

Research challenges and requirements. We identified the following research challenges that result from the inherent lack of consistency along time, performance, occurrence, and environmental dimensions. The main research challenges which motivate our work are: **(RC-1)** How can we derive accurate workload profiles in the face of a small sample size caused by non-recurrent workloads? **(RC-2)** How can we

represent profiled characteristics so that they can capture the workloads' performance variance? **(RC-3)** How can we make the profiling process general, non-invasive, and transparent so that it can seamlessly facilitate various resource provisioning and management techniques?

To address the **RC-1**, we take a pragmatic approach to increase the sample size by continuously analyzing all available workloads from a shared infrastructure. Our approach must only utilize the de-facto standard data, which is typically readily available for any virtualized computing infrastructure. Further, to make our approach practically scalable, we need to be able to derive the profiles in a fully automated manner. Finally, to make our approach generic, it must not rely on any particular assumption and precondition regarding the data and its preprocessing or preparation, e.g., feature engineering.

Addressing **RC-2** requires a novel view of profile representation. Specifically, it is necessary to move away from traditional profiles, which attempt to represent the workload's runtime properties as static profile characteristics and cannot capture their intrinsic performance variance. Instead, we need to adopt dynamic concepts that capture varying degrees of confidence and naturally reflect workloads' performance variance. Finally, to capture performance variance faithfully, we must characterize the profiles based on actual data collected from the production virtualized infrastructure.

Finally, we need to build agile profiling decisions to address the **RC-3** and make our approach generally useful for various resource provisioning and management techniques. We want to assign the profiles as soon as a new workload arrives (to be non-invasive). Furthermore, we want to base the profile assignment on widely adopted and well-known information, such as the workload's metadata (to be transparent).

Contributions. In this paper, we introduce PolarisProfiler – a novel profiling approach that leverages apriori available, static metadata to enable generic and immediate workload profiling based on historic execution traces. More specifically, the main contributions include:

- *A generic workload profile generator*, that automatically derives workload profiles for shared computing infrastructure, based only on the readily available resource usage data, without any specific assumptions (e.g., specifically-tailored feature engineering).
- *A novel model for representing dynamic profiles*, which can be used to capture the dynamic nature of the workload's runtime properties. Our dynamic profiles can be continuously updated, even after initial workload profiling, to reflect the workload's varying performance over time.
- *A metadata-based profile classifier*, which efficiently classifies new workloads and assigns runtime profiles to them by only considering their apriori available, static metadata. This way, new workloads get nearly instantly assigned to profiles.
- *A comprehensive case study*, which describes an example implementation of our profiling methodology. Despite only relying on static, apriori metadata, our methodology yields an error rate below 5% for the 80% of classified workloads. These results are competitive with the state-of-the-art approaches, with the difference that their specific focus is the estimation of AI workloads duration.

We publicly release the code to allow transparency and reproducibility of our results[1].

II. PolarisProfiler Model & Methodology

PolarisProfiler derives *profiles* from a continuous and global characterization of workloads' infrastructure usage. The goal of PolarisProfiler is to profile new workloads solely leveraging information that is known at workload deployment time.

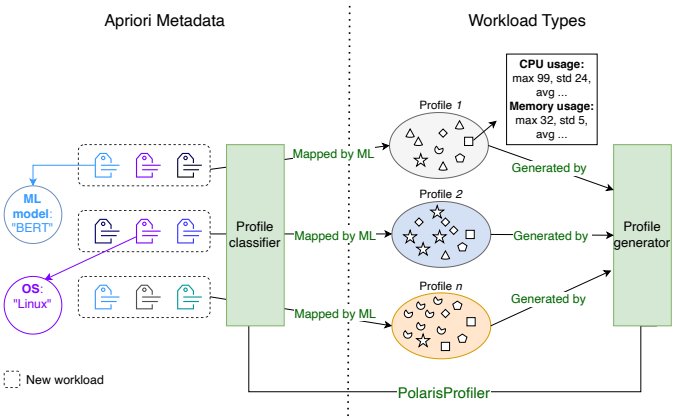


Fig. 1: Overview of the PolarisProfiler's model

A. Model

Figure 1 gives an overview of the PolarisProfiler's model. Every profile exposes *apriori, static metadata* for matching new workloads. We use the term *apriori* to specify that we collect metadata information available at the submission (deployment or provisioning) phase. Examples are user data, application data, and OS parameters. The term *static* identifies invariant metadata. It represents core characteristics that remain stable during the workload life cycle. This metadata is input for the *profile classifier*. This model assigns the workload to the profile that best matches its metadata characteristics (left side of Fig. 1). Hence, we link the submitted workload to the profile's dynamic characteristics. Thus, we leverage the profile runtime information to design appropriate optimization strategies before the workload executes.

The profiles summarize the runtime features of similar workload traces (right side of Fig. 1). At creation time, we look at the similarity in the workloads' execution pattern concerning resource usage. We collect and use a combination of resource usage records over time in aggregated form (e.g., average, quantiles, and deviation) or raw. This way, we have an accurate and information-rich multidimensional representation of workloads and their behavior over time. This approach lets us deal with **RC-2**, i.e., the representation of

[1]https://github.com/polaris-slo-cloud/Profiling/edit/master/ml_data-profiling/README.md

profile information. This way, we know that they will end up in a group, which includes similar processes that behave consistently. Furthermore, thanks to the fact that we use static and a priori metadata, we avoid sandboxing. The richness of the proposed profile enables the extraction of n-dimensional trends, anomalies, and seasonalities from the profile resource records.

B. Workload profile generator

Figure 2 depicts the two main processes for generating and assigning profiles. The first process is the *Workloads' profiles generator* (top box in Figure 2); it is in charge of generating or updating representative profiles. The second process is the *metadata-based profiles classifier* (bottom box of Figure 2); it is responsible for associating incoming workload to the profiles via apriori, static metadata.

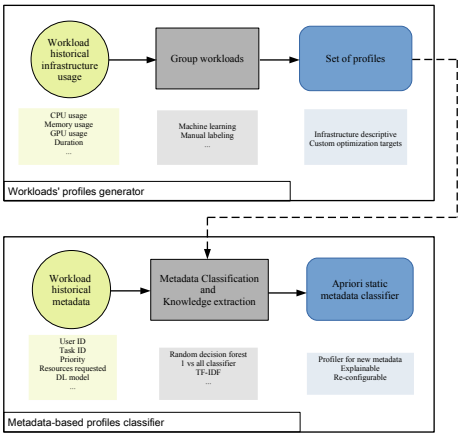


Fig. 2: Overview of PolarisProfiler's main components and processes (partial view).

The workloads' profiles generator uses historical workloads usage traces to address **RC-2**, i.e., describe the workloads' runtime properties. The traces include CPU, memory, GPU, disk usage, or execution duration measures. The role of the workloads' profiles generator is to create profiles that summarize similar workloads usage patterns. A basic approach is to use manual labeling. This approach requires a series of well-defined guidelines and rules to ensure accuracy and consistency. Furthermore, this process requires the work of multiple people to ensure quality, and it needs regular reviews and updates. Although formally feasible, at-scale manual labeling can be an impractical solution [10], [11], and rules updating can be cumbersome. Therefore, machine learning techniques can be more efficient and effective. A possible solution is semi-supervised techniques [12]. We can learn underlying patterns in the data through a small set of labeled entries. Still, label definition is challenging and always relies on static rules. Therefore, unsupervised learning techniques, which do not need any previous knowledge. For example, clustering [13], [14], [15] (eventually with the help of autoencoders [16]) represent for us the preferred solution as they can discover patterns.

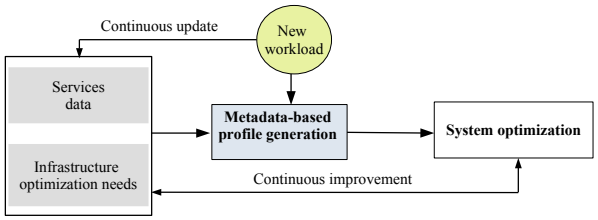


Fig. 3: High-level perspective of the methodology for resource provisioning techniques.

Each profile contains relevant and specific workloads insights gained from clustering them. At the same time, it assembles the apriori, static metadata associated with each workload it integrates. We use the profiles to implement resource provisioning strategies, e.g., workload bootstrapping, predictive monitoring, or improving resource management by estimating workload duration, as shown in §III-D. In summary, using the introduced mechanisms and techniques, the *Workloads' profiles generator* addresses **RC-1** and **RC-2** and provides the input for **RC-3**.

C. Metadata-based profile classifier

The next step is to guarantee the association of new workloads to profiles through metadata. This process should be "fast" (where this notion depends on the target goal), scalable, and flexible. The implementation options include manually extracting information and defining rules or using automated mechanisms. Again, at scale, manual approaches are unsuitable. Therefore, we look at automatable machine learning methods to provide better results and faster updates. Generally, we can handle this as a *nearest neighbor* problem or a multi-label classification problem (e.g., through rule-based models or decision trees). Alternatively, we can use neural networks. However, neural networks do not guarantee interpretability, an essential property for making decisions transparent. For this reason, we rely on white box models; in particular, we select a specific variation of decision trees or random forests, eXtreme Gradient Boosting (XGBoost). This method improves on the Gradient Boosting Tree (GBT); it handles sparse input data and distributes and scales the execution efficiently [17].

To conclude, we use workloads' a priori static metadata to classify it into a profile leveraging a decision tree method. This process addresses **RC-3** and the time and computational constraints imposed. While we provide examples of relevant metadata features (as in Figure 2, bottom), we do not bind ourselves to any predefined set; an accurate metadata taxonomy is out of this paper's scope.

D. Application in resource provisioning techniques

For the PolarisProfiler, we envision a continuous improvement process. Figure 3 clarifies our perspective, providing a high-level overview. In such scenarios, the workload typically arrives with a set of static metadata features and infrastructure demands. The information flow we aim to produce with our approach leads to two paths. First, the new workload gets a

profile assignment that aids the system's resource provisioning strategies to make the best decision. At the same time, when the workload starts, we store the workload's runtime behavior, thus updating and enriching the selected profiles. With this feedback loop, we continuously improve infrastructure usage. At the same time, by monitoring the PolarisProfile, we can recognize when the profiles' definition or classification is deteriorating, thus triggering actions like re-clustering.

The PolarisProfiler aims to solve various concerns in resource provisioning and management. First, it can help the scheduling process by facilitating more informed decisions [18]. Knowing the profile of a workload apriori can help in sampling more suitable machines [19] and filtering and scoring the ones best tailored to that model to serve the request. Furthermore, *aaS solutions must satisfy users' SLOs [1], [20]. In this regard, achieving it in the bootstrapping phase takes work. There is a need to bring an application online and satisfy the defined SLOs by only leveraging little information. In this context, the PolarisProfiler provides the information needed to assess the application behavior. If we consider FaaS, there is a gap in how to tailor the correct resources from a heterogenous infrastructure [21], [22], [23] for specific functions. Here, the PolarisProfiler aids in pairing the function characteristics with the most appropriate node configuration by highlighting patterns in node usage and application behavior. Finally, the upsurge of machine learning (ML) is bringing in a new, complex class of workload. As we show in this paper, the PolarisProfiler provides tools and mechanisms to infer ML workload characteristics, like resource usage and duration, and use the infrastructure better.

III. Case Study

We provide a reference implementation of PolarisProfiler and its main profiling processes. Specifically, we develop a profiling approach to optimize the scheduling of Machine Learning (ML) workloads. In our use case, we focus on assessing the workload duration as it is a crucial feature to plan and schedule workloads around it, ensuring efficient use of resources while meeting SLOs.

The rationale for targeting machine learning workload is that it represents a current challenge for large and distributed systems [24]. The need for a large amount of data and an increased necessity for the computing power of energy poses serious questions [25] and calls for optimization strategies from the AI and systems communities. Furthermore, the variety of algorithms and the specific behavior of ML models push to consider more dimensions in the search for resource usage patterns and optimize their use to guarantee the best and the most transparent service for users [26]. Therefore, we cannot only rely on CPU metrics.

Our study considers two months of ML job traces from the Alibaba Platform for Artificial Intelligence (PAI) [27]. The platform's main target is businesses within the Alibaba group. It enables AI pipelines, offering different levels of abstraction, from a canvas UI where the users can drag and connect the elements for their pipeline to containers. Once submitted, the supported frameworks [2] translate each workload into tasks with different roles, e.g., *parameter servers (PS)* and *workers* for a training job and *evaluator* for inference. Each task has one or more instances, deployed using Docker, and can run on multiple machines. This dataset is relevant for our case study as it shows several key characteristics. First of all, it contains real traces, reporting real machine usage. Furthermore, it discloses descriptive static and apriori metadata. The most suitable metadata contained in Alibaba's dataset is the user's name (*user*), the job name (*job name*), the model used (*workload*), and the type of the task, e.g., if it is training or inference and which architecture uses (*task name*). Plus, the Alibaba trace comes with a *group* tag, i.e., meta-information specified by tasks, such as entry scripts, command line parameters, data source, and sinks.

TABLE I: Stratified sampling of 100 001 elements based on the workload metadata feature.

Workload	Size	Sampled size
bert	10 940 142	29 818
ctr	9 128 957	24 881
graphlearn	4 888 371	13 323
inception	10 781 289	29 385
nmt	13 537	37
resnet	60 863	166
rl	849 626	2 316
vgg	11 768	32
xlnet	15 632	43
Total size	36 690 185	100 001

We initially take an outsider perspective regarding the Alibaba dataset, where we do not have insights about the system. First, we construct our case study filtering out all the jobs that are not *terminated* since we do not have the resources usage information for them, obtaining circa 36 million instances. Then, we use stratified sampling to reduce the set to a manageable size. We base the stratification on the workload type, which, through the model names, gives us an explicit and more transparent understanding of the jobs and their instances. We extract a dataset D with a cardinality $|D| = 100\,001$ elements. Table I shows the categories and their sampled sizes. Our goal is to create homogeneous profiles from the infrastructure usage perspective. Thus, we rely on 17 usage metrics.[3]

Consequently, we need to understand if we can build a dynamic profile model, i.e., if we can look at the historical workload in the infrastructure and find profiles that contain workloads that expose similar behavior. In our case, we rely on the Hopkins statistics [28]. This test measures the "clusterability" of data, relying on the hypothesis that the data follows a Poisson point process. It outputs a score: if equal or above 0.3, the data have random distribution; the closer the values go to zero, the more the data could follow clusters. We

[2]PAI accepts frameworks like TensorFlow, PyTorch, Graph-Learn, and RLlib.

[3]Namely: the number of instances for that job (*inst num*), the starting and ending time (*start time* and *end time*), the planned resource usages (*plan cpu*, *plan mem*, and *plan gpu*. Plus, the dynamic utilization metrics like *CPU usage*, *memory usage* (average and maximum), *GPU usage*, *GPU memory usage* (average and maximum), number of inputs and outputs (*read count* and *write count*), number of bytes exchanged (*read*, and *write*) and the total job *duration*.

TABLE II: Results of preliminary data analysis on the most verbose static *apriori* metadata features.

Metadata feature(s)	Avg. Silhouette score		
	Euclidean	*Cosine*	*Manhattan*
Workload	0.17	0.03	**0.21**
Task name	**-0.07**	-0.19	-0.10
(Workload, Task name)	0.02	-0.01	**0.08**

rely on the Python `pyclustertend` library for our analysis, that uses as default distance "Minkowski," which results in the standard Euclidean distance.[4, 5] For the set D, the Hopkins score is **0.0033**, letting us believe in the possibility of obtaining meaningful profiles.

A. Fixed labeling

As we point out in the introduction, most profiling methods rely on *occurrence consistency* [8], [7], [9]. Therefore, we assemble a *baseline test* to evaluate the performance of single or combined static apriori metadata if set as profile labels. The idea is to mimic the "expert" view, which uses information about the workload. For this task, we rely on *workload* and *task name*, who represent the most intelligible metadata.

We analyze how well a single or a small group of metadata features can group workload that behave similarly, i.e., that is close in our 17-dimensional problem (considering the 17 resource utilization metrics). For the evaluation, we use the well-established unsupervised metric Silhouette coefficient (silhouette) [29] (SC_{score}). It tells in a $[-1, 1]$ range how well each point lies within its group.[6] For a better assessment, we consider three distance measures: *Euclidean*, *Cosine*, and *Manhattan*. Table II summarizes the findings. The results suggest difficulty identifying homogeneous groups, i.e., points with the silhouette strictly greater than 0. However, *workload* labels contain relevant, but not sufficient, information for distinguishing the workloads. In this case, the *Manhattan* distance performs better than the other. Having a transversal look at the distance metrics, the *Cosine* distance is the one that provides more uncertain results, probably due to its use mainly related to categorical data. This analysis suggests that a combination of metadata labels will be required to identify profiles, further, leveraging workload's historical resource usage will also ensure that the obtained groups are cohesive. The goal is to have profiles that prove to be more cohesive than the groups obtained in this baseline test using just one metadata label.

B. Developing the workload profile generator

After establishing our baseline, we inspect methods for generating profiles. In particular, we aim at building dynamic profiles as previously described. In this case, we need to explore methodologies that can help us extract homogeneous groups in an unsupervised way. In this context, clustering is amongst the most popular approaches. Three main categories of clustering are partitional, hierarchical, and density-based.

[4]https://pyclustertend.readthedocs.io/en/master/

[5]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html

[6]values closer to 1 representing a better fit

TABLE III: Parameters for the clustering grid search.

Search Category	Values
Model	*HDBSCAN* *OPTICS*
Data transformation	*StandardScaler* *MinMaxScaler* *RobustScaler* *PowerTransform*
Distance metric	Euclidean Manhattan
Min cluster points	50, 100, 200, 300, 400, 600, 1 000

Density-based methods offer two main features that fit our case study. First, they do not require us to specify the desired number of clusters apriori. Second, they generate an "outliers' group," i.e., workloads not fitting any cluster. This last characteristic lets us explore irregular workloads and detect peculiar behaviors. We aim to have fine-grained clustering; therefore, we focus on methods that generate groups at different data densities. The main algorithms are *HDBSCAN* [30], [31] and *OPTICS* [32].

At first, we evaluate the best configuration for *HDBSCAN* and *OPTICS* on the case study dataset D. The first parameter we examine is the data transformation tool for the dynamic workload feature. We consider the *StandardScaler*, the *MinMaxScaler*, the *RobustScaler* – particularly suitable for noisy datasets – and the *PowerTransform*, which produces a monotonic transformation. An essential element in clustering is the distance metric. For our scenario, we choose the *Euclidean* and the *Manhattan*. Finally, both *HDBSCAN* and *OPTICS* need to input a parameter specifying the number of minimum points per cluster, *MinPoints*. We choose a range that includes the potential advantage of having many small and accurate clusters and the possibility of large representative groups. Table III summarizes the parameters.

We extract several statistics for each clustering result ($\mathcal{C}$). We consider the number of clusters generated, how many outliers $\mathcal{O}$ the clustering detects, and the average cluster size. Furthermore, we rely on unsupervised performance metrics, such as the overall SC_{score} and the Davies Bouldin Score (DB Score). In addition, in our use case, we want to maximize the number of clustered points to have a significant representation in the profiles. Finally, we want to have an adequate number of clusters. We want to have more than one big group and avoid many small clusters. Therefore, we look at having a good balance between the number of clusters and their cardinality (mean |C|). Table IV summarizes the main statistics for HDBSCAN and OPTICS. We can see how HDBSCAN fits our requirements better, as highlighted in bold in Table IV. Additionally, we show how clustering techniques can accurately deduce profiles of homogeneous workloads.

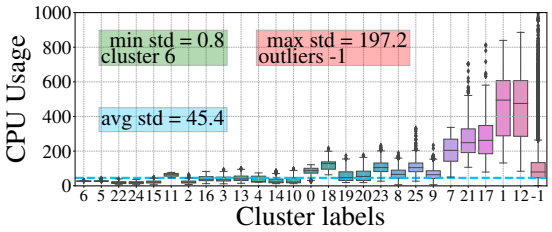
1) HDBSCAN evaluation

Here, we inspect the results of the selected HDBSCAN approach for generating dynamic profile models. First, we examine the performance in terms of cluster separation, relying again on the SC_{score}. We keep out from this analysis the "out-

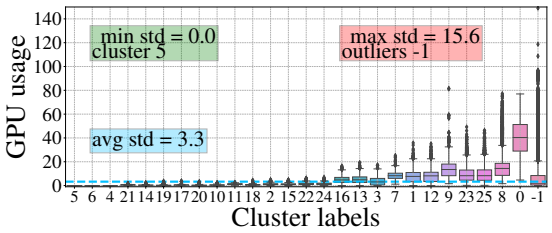
TABLE IV: Summary of the clustering results

		# outliers	# clusters	mean |C|	avg SC_{score}	DB score
HDBSCAN	mean	**40331.1**	68.6	**1993.6**	0.44	**1.52**
	min	**18059**	**10**	**254.9**	**0.23**	**1.19**
	max	**64770**	**243**	**5859.5**	0.65	**2.01**
	std	11124.0	70.3	1509.3	0.08	0.19
OPTICS	mean	79578.3	**66.2**	889.5	**0.60**	1.19
	min	66708	1	120.6	-1.00	0.98
	max	98936	271	2979.5	**0.81**	1.40
	std	7643.2	79.3	752.1	0.20	0.10

liers group." [7] Overall, clusters 8 and 9 have a good SC_{score}, despite their large size. Profiles 5, 6, and 11 are the ones that have the best SC_{score}. Their low cardinality and sample fit suggest that they represent particular and homogeneous job instances. However, profile 10 has a significant amount of not well-fitted samples. Even if this last behavior is not negligible, it is unrealistic to expect perfect results with such cardinality. Overall, the results are well grounded and show how, in the case study, HDBSCAN is a good candidate to implement our workload profile generator.



(a) CPU usage.



(b) GPU working utilization.

Fig. 4: Boxplot representing the features distribution in the clustered profiles.

a) Dynamic infrastructure usage data

We now represent the range of workloads performance within the profiles to understand the core dynamic profile model. In detail, we examine the distribution of resource usage values across the profiles and their variability in each cluster. Figure 4 depicts the results. Due to the page limit, we focus on the most representative features for the case study: CPU usageand GPU utilization. The figure shows the boxplots of the feature values grouped by the cluster labels. The plots sort the profiles, in the x-axis, by the considered feature standard deviation, in ascending order; higher values are at the right of the plot. We sort by the standard deviation to highlight the value of cohesiveness within each profile. The y-axis shows, for each feature, their values.The green and red text boxes report the minimum and maximum standard deviation, respectively. The turquoise lines and boxes show the average standard deviation value instead.

The overall results show that most of the profiles have relatively low variation. The exception is the "outliers group," labeled as "-1," which naturally contains all the workloads that do not fit in the main profiles. A particular case is maximum memory usage, where profile 18 has a broader value range than the outliers group. As a possible cause, this profile contains few workload samples and might include peculiar workloads. On the contrary, sizeable profiles, like 8 and 9, show a good homogeneity, with generally few noisy points present. Overall, this first analysis suggests that the HDBSCAN clustering has managed to find homogeneous groups of jobs. Furthermore, such representation demonstrates the contribution of profiles to the estimation of the runtime characteristics of a workload.

b) Metadata

Analyzing the metadata in the clusters is essential for **RC-3**, i.e., assigning profiles to new workloads. [8] For seven out of ten jobs, most of the values end in profiles 8 and 9, suggesting that these large groups contain various but similar workloads. These two profiles include, for the large part, "bert" workload. Furthermore, besides the "rl" workload, which characterizes profiles 5 and 6, the other workload feature values are scattered in the other clusters. Moreover, the clustering approach discarded the "resnet," "nmt," and "vgg" values. Looking at the cardinality of these values, which is lower than 500 – our minimum cluster size – we can understand why they are not in clusters. Indeed, the last four values for task name distribution all have a cardinality below 200. These results show how the HDBSCAN-based profiling helps to distinguish workloads in the case study. This outcome is significant, considering that different workloads might show different patterns. Finally, some users and groups have a higher representation than others in the profiles pair 8 and 9 or the 19 and 20 pair. These two groups mainly refer to "bert," as previously seen, and "graphlearn." This outcome suggests that certain users focus on specific implementations, like "bert" and "graphlearn" and that these implementations have very specific meta-information embedded in the "group" metadata. Ultimately, this outline of the metadata distribution suggests that the clustering based on dynamic data can identify patterns in the metadata features and that combining these values in input can lead to accurately detecting profiles.

C. Developing the metadata-based profile classifier

The final, essential step in the presented methodology is assigning a profile to newly submitted workloads. This task has to happen fast and by leveraging static, apriori metadata. We illustrate through the case study how to build such a classifier and discuss its performance. Furthermore, besides assigning new jobs to the profiles, we aim to understand the relevance of

[7] We refer to the image `HDBSCAN_silh_results_box.pdf` in our repository for the visualization.

[8] Related heatmap figures in the repository.

metadata features in the decision-making process through the model, which maps the input to the labels. Therefore, we rely on the interpretable eXtreem Gradient Boosting (XGBoost) classification model due to its performance in classifying and its *white box* characteristics.

1) Training the classifier

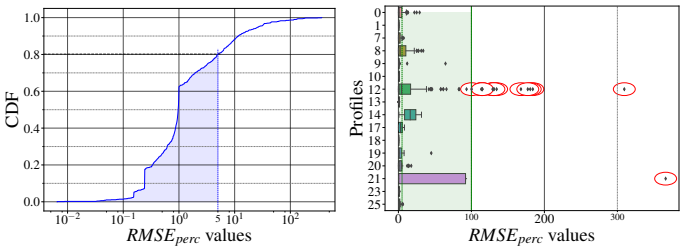
We use the dataset of clustered elements $D^{\mathcal{C}}$, leaving out the outliers group. From each clustered workload, we extract their static, apriori metadata features, namely: *job name*, *user*, *task name*, *group*, and *workload*. Overall we obtain a set with a cardinality of 75 398 and a dimension of 5, i.e., the metadata features. For the model generation, we subdivide the collection in *training* and *validation* sets, with an 80-20 ratio. The XGBoost algorithm has limited support for categorical data. So, we must transform the input features into numerical ones. Valid approaches are *one-hot encoding* or recurring to the *embedding* networks. The latter requires a long training time; therefore, we use the former approach. After this transformation, the set dimension grows to 21 547. We store the data as a sparse matrix to optimize the computation. In this case, we use the standard hyperparameters for XGBoost. Our aim in the case study is to analyze its performance and avoid overfitting.

Table V summarizes the results on the validation set per profile. We can appreciate that the results are excellent for most of the profiles, except for profile 6, where the classifier can not correctly label any of its points. In general, we obtain an *accuracy* of 95.19% and a *weighted avg* F1-Score of 90%. Overall, the results show a good capability of the trained model in predicting profiles independently from their size and starting just from apriori knowledge about a workload and its instance(s). This result gives us a promising path towards reproducing the proposed profiling approach, given the selected case study scenario where the granularity of information is partially insightful. Furthermore, the sample selected and the resulting profiles extracted from it are wide-ranged enough to constitute a complex undertaking for the model. Finally, an additional advantage of this approach is the speed with which the model can label new workloads. We do not need any dry runs on sandboxes or runtime profiling.

2) Results explanation

A key feature is to obtain explainable results. We achieve that using the SHAP eXplainable AI (XAI) approach [33], [34], of the top twenty features in the XGBoost model. [9] Particularly relevant are the *task name* in its "*ps*" value, the *graphlearn workload* type, and a specific user. The *task name*: "*ps*" category refers to using a Parameter Server (PS) architecture for models' training. In this case, one or more nodes play the role of a PS, broadcasting current weights to learners before each step and aggregating gradients from them, which is an easy way to retain a global view [35], [36]. This behavior might represent a demarcation with other training architecture. Similarly, Graph Neural Networks (GNNs)

[9]The figure `SHAP_summaryplot_allclasses.pdf` is available in the repository



(a) CDF of the $RMSE_{perc}$ in the duration prediction test.

(b) Boxplot of the $RMSE_{perc}$ aggregated by the profile labels predicted for the points in the set.

Fig. 5: Summary of the $RMSE_{perc}$ results for the duration prediction test.

(*workload*: "*graphlearn*") have a very distinct behavior as they deal with graph data in the input. In particular, their distributed execution using Alibaba's developed framework can differentiate them from other workloads. The same goes for the NLP model labeled as (*workload*: "*bert*"), which characterizes profiles 8 and 9. Furthermore, the *job name* `94b340f2cdedf37303d41bf2` is the most recurrent in our dataset, and it occurs in profiles 5 and 6. If we link this outcome with what we found in §III-B1, we can match that these two clusters had very specific and defined resource usage values with a constantly low standard deviation. Therefore, it is easy to associate this metadata with a relevant decision boundary. Overall, the use of the SHAP explainability tool reinforces the idea of our profiling approach, i.e., that the static apriori metadata represents a suitable and rich vehicle to match jobs to distinct profiles.

D. Test case: predicting the jobs' duration

Finally, we test the capability of profiles to embed relevant information. To do so, we extract a set of 1 000 workloads $\mathcal{J}_{sample}$ randomly sampling them from the dataset and making sure they are not part of our train and validation set. Once the classifier assigns each of the sampled workloads $j \in \mathcal{J}_{sample}$ to a profile $p \in \mathcal{P}$, we use the average duration $\hat{d}_p$ to assign the workload the predicted duration. Naturally, other more sophisticated approaches can be better for the estimation other than the mean; we leave this aspect for future research work. Afterward, we use the normalized Root Mean Squared Error $RMSE_{perc}$ to compute the loss between $\hat{d}_p$ and the actual job duration $\hat{d}_j$.

Figure 5a gives us a high-level understanding of how this approach predicts duration values. It shows the Cumulative Density Function (CDF) of the $RMSE_{perc}$. The $RMSE_{perc}$ is below the 5% for 80% of the samples, as depicted in the highlighted blue area. 90% of the predictions have an $RMSE_{perc}$ error below the 20%. However, Figure 5a highlights some outliers, with $RMSE_{perc}$ values even above 1 000 %. We aim to identify these outliers better in Figure 5b, which shows the boxplots representing the $RMSE_{perc}$ values for each profile. The outliers, rimmed by the red circle, are the ones with $RMSE_{perc}$ values above 100%, as highlighted by the green area at the left of the plot. We can see that they

TABLE V: Class-level classification score reports.

Profile	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	macro avg	weight. avg
precision	0.90	0.99	1.00	1.00	1.00	0.66	0.00	0.94	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.84	0.99	0.84	0.93	0.95
recall	1.00	1.00	1.00	0.99	1.00	1.00	0.00	0.99	0.98	1.00	0.98	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.26	0.99	0.30	1.00	0.90	0.95
f1-score	0.94	1.00	1.00	0.99	1.00	0.80	0.00	0.96	0.99	1.00	0.99	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.41	0.91	0.46	0.91	0.90	0.94
support	415	1172	177	607	143	134	68	134	2509	2316	128	64	1104	468	135	158	370	116	99	435	437	57	383	1489	426	1536	15080	15080

belong to two profiles, namely, 12, and 21. Overall, these results show both the quality of the profiling approach and the optimization possibilities the methodology brings. In brief, by leveraging the profiling methodology developed in this use case and simply using the average value duration for each profile, we have been able to predict the duration of 80% of the workloads with an error of only 5%. This capability opens a wide range of optimization capabilities for Cloud infrastructure by leveraging already available metadata.

IV. Related work

A. Profiling

Building statistics and extracting workload patterns has increased interest in the last decades. For example, Dryad [37] extracted workload statistics for the distributed workload. The research developed more sophisticated approaches from that time, given new computational and methodological capabilities. Ahn Vu Do et al. [38] used Canonical Correlation Analysis to find the relationship between performance and resource usage, relying on the "consistency" assumptions. In terms of building profiles that estimate SLO-related parameters, Sinan [39] offers an ML-fueled approach to help reduce QoS violations. The models predict the application performance given specific resource allocation measures. PARTIES [40] offers online profiling. As Kairos and SLearn approach, it uses runtime information, discarding apriori knowledge, highlighting the difficulties of having information from user-submitted workloads. On the same line, Kaushik et al. [41] profile application at runtime for improving vertical scaling. Inagaki et al. [42] worked on profiling microservices to detect runtime bottlenecks. Gibilisco et al. [43] also focus on runtime profile sampling for Spark performance. Manner et al. [44] perform dynamic profiling through simulations. Rao et al. [45] combine static and dynamic profiling with a focus on Spark. On the contrary, we use available static metadata as any workload is submitted, making the apriori matching trouble-free. Other works [46], [47], [48] collect "offline" runtime information, running the workloads in exclusive mode. This approach, though, suffers from the environment's inconsistency.

Recently, most of the research focused on characterizing Machine Learning workload. Some works [49], [50] approach the profiling using historical execution traces containing hardware attributes and runtime data to forecast the duration of a DNN's training iteration. Aryl [51] leveraging the former approach to estimate the DNN workload duration, using the history of the runs of the same workload. SCHEDTUNE [52] leverages historical execution traces to build profiles to predict resource usage. Our case study differentiates from that as we follow a more generic approach, i.e., we do not focus solely on training and do not consider hardware assumptions.

On the contrary, we are more generic and resilient to environmental and infrastructure changes. Based on Habitat, EOP [53] aims at characterizing deep learning inference tasks by looking at three main characteristics of the DNN, such as the *batch size*, *Height-weight-weight*, and *Height-weight-weight*. Again, this approach targets a narrow problem and makes strong apriori assumptions on the features that can better represent the workloads. Shin et al. [54] developed an approach to profile the workload of AI applications. Their focus is on preventing out-of-memory cases by studying TensorFlow internals. Conversely, we do not look at the internals of a specific framework. Instead, we use static and easy-to-obtain metadata.

Similarly to our method, Hu et al. [55] rank workloads using GPU time, correlating it to attributes, such as workload name, user, and submission time. They leverage these attributes to predict the workloads' priority in scheduling. This approach follows a similar methodology. However, we aim to provide a more generic approach to automatically extract these correlations and patterns. InfaaS [56] proposes using statically-profiled metadata, plus the tracking of dynamic state for high-level-requirement-based distributed inference serving. Other works [57], [13] use *unsupervised* approaches for workload characterizations and mapping. CloudCluster [14] is instead a method for clustering VM-to-VM traffic. Carver [58] uses statistical approaches to extract relevant features in storage systems. Fibratus is a method developed by Horovitz et al. [59] to correlate service-specific protocol data patterns with transactional flow patterns to provide additional insights for performance profiling using a hierarchical clustering method. While they share with our work the idea of not being intrusive with monitoring, they focus on clustering network traces. Kattepur et al. [60] have a methodology in principle similar to our approach, but, in practice, runtime based and focusing on robotics through fog networks. However, unlike the previous approaches, we use static, apriori, and readily available metadata to assign the new workload to profiles.

B. Workload duration estimation

Runtime estimates are a common practice used by modern scheduling systems to make decisions. Previous work used offline-based approaches to estimate the duration of workloads [8], [61] These works estimate the duration by using assumptions on specific features, e.g., task type and dataset size. Instead, our work relies on a generic approach that uses old dynamic information to infer the specific static and apriori metadata features to detect homogeneous profiles. Other approaches, like 3Sigma [6], rely on the total historical workload duration distributions to predict how long the new workloads will start. Similarly, Weng et al. [27] use the Alibaba dataset past estimation and a set of fixed parameters,

i.e., *group* and *user*, to estimate the workload completion time. Conversely, we create specific profiles to address such challenges. Other contributions follow different approaches. Kairos [62] does not require any a priori knowledge of task runtime. Instead, Kairos employs preemption to estimate the predicted remaining runtime of tasks from when they have already been completed. Similarly, Jajoo et al. [5] propose a learning-in-space approach (*SLearn*). They select and schedule only a portion of each workload's tasks. This method takes advantage of the similarities between the runtime characteristics of the tasks inside a single workload. Still, these and similar online approaches are subject to "environment inconsistency."

V. Conclusions

This paper introduced the PolarisProfiler, a novel methodology for profiling workload using only easily accessible resource usage information to build the profiles and static and apriori metadata features for workload assignment. We initially took a conceptual perspective, delineating the main characteristics of our definition of profiles. Further, we introduced the main tools and methodologies. We then consolidated our approach conceptualization, evaluating it through a case study. We conducted a comprehensive analysis of real ML workload traces, leveraging the Alibaba dataset. With this case study, we delivered two primary outcomes. Firstly, we outlined practical methods and algorithms to implement the previously defined conceptual approaches. We showed how clustering could help to build profiles and how white box classifiers, like XGBoost, can map new workloads to profiles based on metadata. Secondly, we presented how this approach can achieve good results on coarse-grained ML workload profiles, showing how an unsupervised, assumption-agnostic approach can provide precise workload duration estimation. Finally, we delineated challenges and roadmaps for future improvement of the presented method.

In the future, we intend to utilize the PolarisProfiler to optimize several problems. One of our future research goals is to use our profiling approach natively when designing SLO policies. To achieve this, we intend to embed its support in SLO Script, a language for implementing complex cloud-native elasticity-driven SLOs [63]. Furthermore, we want to explore profiling in the context of serverless management, where there is a need for better approaches for SLO awareness [23]. In addition, we are applying profiling to the various stages of scheduling, as already established in [19]. We also intend to extend the current PolarisProfiler in several directions. An essential part of the profiles is to be dynamic and adapt to changing workloads and environments. Therefore, we aim to develop strategies to adjust profiles at runtime. One way we would explore is constantly updating the profiles with the new, related workload and periodically re-cluster the underperforming ones. Furthermore, we want to improve the generalization of our approach, making it work with various objectives other than duration. This direction requires us to consider better prediction models for both the final SLO objective and the PolarisProfiler building blocks. For example, we aim at improving the extraction of the profile metadata from the clustering, e.g., employing vector-like representation of the metadata using autoencoders.

References

[1] S. Nastic *et al.*, "Sloc: Service level objectives for next generation cloud computing," *IEEE Internet Computing*, vol. 24, no. 3, pp. 39–50, 2020.

[2] Y. Xiong *et al.*, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 373–377.

[3] R. Householder *et al.*, "On cloud-based oversubscription," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 8, no. 8, pp. 425–431, 2014.

[4] S. M. Noonan, "Managing resource bursting," Aug. 16 2016, uS Patent 9,417,902.

[5] A. Jajoo *et al.*, "A case for task sampling based learning for cluster job scheduling," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 19–33.

[6] J. W. Park *et al.*, "3sigma: distribution-based cluster scheduling for runtime uncertainty," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–17.

[7] S. A. Jyothi *et al.*, "Morpheus: Towards automated {SLOs} for enterprise clusters," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 117–134.

[8] A. D. Ferguson *et al.*, "Jockey: guaranteed job latency in data parallel clusters," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 99–112.

[9] V. Jalaparti *et al.*, "Network-aware scheduling for data-parallel jobs: Plan when you can," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 407–420, 2015.

[10] D. Abadi *et al.*, "The beckman report on database research," *Communications of the ACM*, vol. 59, no. 2, pp. 92–99, 2016.

[11] M. Stonebraker *et al.*, "Data curation at scale: the data tamer system." in *Cidr*, vol. 2013, 2013.

[12] T. Khan *et al.*, "Workload forecasting and energy state estimation in cloud data centres: Ml-centric approach," *Future Generation Computer Systems*, vol. 128, pp. 320–332, 2022.

[13] J. L. Berral *et al.*, "{AI4DL}: Mining behaviors of deep learning workloads for resource management," in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[14] W. Pang *et al.*, "{CloudCluster}: Unearthing the functional structure of a cloud service," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1213–1230.

[15] H. Wang and B. Li, "Lube: Mitigating bottlenecks in wide area data analytics," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[16] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[17] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[18] S. Nastic *et al.*, "Polaris scheduler: Edge sensitive and slo aware workload scheduling in cloud-edge-iot clusters," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 206–216.

[19] V. Casamayor Pujol *et al.*, "Intelligent sampling: A novel approach to optimize workload scheduling in large-scale heterogeneous computing continuum," in *2023 18th Annual System of Systems Engineering Conference (SOSE), (to appear)*, 2023.

[20] T. Pusztai *et al.*, "A novel middleware for efficiently implementing complex cloud-native slos," in *IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021.

[21] Q. Zhang *et al.*, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 510–519.

[22] S. Nastic *et al.*, "A serverless computing fabric for edge & cloud," in *4th IEEE International Conference on Cognitive Machine Intelligence (CogMi)*, 2022.

[23] P. Raith *et al.*, "Serverless edge computing—where we are and what lies ahead," *IEEE Internet Computing*, vol. 27, no. 3, pp. 50–64, 2023.

[24] J. Verbraeken *et al.*, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[25] E. M. Bender *et al.*, "On the dangers of stochastic parrots: Can language models be too big?" in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623.

[26] C. Wan *et al.*, "Are machine learning cloud apis used correctly?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 125–137.

[27] Q. Weng *et al.*, "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*, 2022.

[28] A. Banerjee and R. N. Dave, "Validating clusters using the hopkins statistic," in *2004 IEEE International conference on fuzzy systems (IEEE Cat. No. 04CH37542)*, vol. 1. IEEE, 2004, pp. 149–153.

[29] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[30] R. J. Campello *et al.*, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.

[31] L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2017, pp. 33–42.

[32] M. Ankerst *et al.*, "Optics: ordering points to identify the clustering structure," in *ACM Sigmod record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[33] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[34] S. M. Lundberg *et al.*, "From local explanations to global understanding with explainable ai for trees," *Nature machine intelligence*, vol. 2, no. 1, pp. 56–67, 2020.

[35] M. Wang *et al.*, "Characterizing deep learning training workloads on alibaba-pai," in *2019 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2019, pp. 189–202.

[36] S. Li *et al.*, "Taming unbalanced training workloads in deep learning with partial collective operations," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 45–61.

[37] M. Isard *et al.*, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007, pp. 59–72.

[38] A. V. Do *et al.*, "Profiling applications for virtual machine placement in clouds," in *2011 IEEE 4th international conference on cloud computing*. IEEE, 2011, pp. 660–667.

[39] Y. Zhang *et al.*, "Sinan: Ml-based and qos-aware resource management for cloud microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 167–181.

[40] S. Chen *et al.*, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 107–120.

[41] P. Kaushik *et al.*, "A study of contributing factors to power aware vertical scaling of deadline constrained applications," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 500–510.

[42] T. Inagaki *et al.*, "Detecting layered bottlenecks in microservices," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 385–396.

[43] G. P. Gibilisco *et al.*, "Stage aware performance modeling of dag based in memory analytic platforms," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 188–195.

[44] J. Manner *et al.*, "Optimizing cloud function configuration via local simulations," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 168–178.

[45] B. Rao *et al.*, "Soda: A semantics-aware optimization framework for data-intensive applications using hybrid program analysis," in *2021 IEEE 14th International Conference On Cloud Computing (CLOUD)*. IEEE, 2021, pp. 433–444.

[46] D. Narayanan *et al.*, "{Heterogeneity-Aware} cluster scheduling policies for deep learning workloads," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 481–498.

[47] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 595–610.

[48] K. Mahajan *et al.*, "Themis: Fair and efficient {GPU} cluster scheduling," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 289–304.

[49] G. Yeung *et al.*, "Towards {GPU} utilization prediction for cloud deep learning," in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[50] X. Y. Geoffrey *et al.*, "Habitat: A {Runtime-Based} computational performance predictor for deep neural network training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 503–521.

[51] J. Li *et al.*, "Aryl: An elastic cluster scheduler for deep learning," *arXiv preprint arXiv:2202.07896*, 2022.

[52] H. Albahar *et al.*, "Schedtune: A heterogeneity-aware gpu scheduler for deep learning," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 695–705.

[53] Y. Xu *et al.*, "Eop: efficient operator partition for deep learning inference over edge servers," in *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2022, pp. 45–57.

[54] C. Shin *et al.*, "Xonar: Profiling-based job orderer for distributed deep learning," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 112–114.

[55] Q. Hu *et al.*, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[56] F. Romero *et al.*, "{INFaaS}: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[57] D. Van Aken *et al.*, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1009–1024.

[58] Z. Cao *et al.*, "Carver: Finding important parameters for storage system tuning," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 43–57.

[59] S. Horovitz *et al.*, "Non-intrusive cloud application transaction pattern discovery," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 311–320.

[60] A. Kattepur *et al.*, "A-priori estimation of computation times in fog networked robotics," in *2017 IEEE international conference on edge computing (EDGE)*. IEEE, 2017, pp. 9–16.

[61] K. Karanasos *et al.*, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 485–497.

[62] P. Delgado *et al.*, "Kairos: Preemptive data center scheduling without runtime estimates," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 135–148.

[63] T. Pusztai *et al.*, "Slo script: A novel language for implementing complex cloud-native elasticity-driven slos," in *IEEE International Conference on Web Services (ICWS)*, 2021.