AI4DL: Mining Behaviors of Deep Learning Workloads for Resource Management

Josep L. Berral
Barcelona Supercomputing Center,
Universitat Politecnica de Catalunya
josep.berral@bsc.es

Chen Wang
IBM Research
chen.wang1@ibm.com

Alaa Youssef
IBM Research
asyousse@us.ibm.com

Abstract

The more we know about the resource usage patterns of workloads, the better we can allocate resources. Here we present a methodology to discover resource usage behaviors of containers training Deep Learning (DL) models. From monitoring, we can observe repeating patterns and similitude of resource usage among containers training different DL models. The repeating patterns observed can be leveraged by the scheduler or the resource autoscaler to reduce resource fragmentation and overall resource utilization in a dedicated DL cluster. Specifically, our approach combines Conditional Restricted Boltzmann Machines (CRBMs) and clustering techniques to discover common sequences of behaviors (phases) of containers running the DL training workloads in clusters providing IBM Deep Learning Services. By studying the resource usage pattern at each phase and the typical sequences of phases among different containers, we discover a reduced set of prototypical executions representing the majority of executions. We use statistical information from each phase to refine resource provisioning by dynamically tuning the amount of resource each container requires at each phase. Evaluation of our method shows that by leveraging typical resource usage patterns, we can auto-scale containers to reduce CPU and Memory allocation by 30% compared to statistics based reactive policies, which is close to having *a-priori* knowledge of resource usage while fulfilling resource demand over 95% of the time.

1 Introduction

Deep Learning (DL) platforms on the Cloud, e.g., IBM's Deep Learning as a Service (DLaaS) [8] [12], Google Cloud AI Platform [1] or Kubeflow [2], deserve special attention when managing resources. Classic high-performance computing environments tend to provision dedicated machines for each particular training job. However, cloud-native machine learning services can have multiple containers co-located on the same machine, and each alternates strides of data fetching and data processing. For this reason, allocating a fixed amount of resources for each container may result in resource over-provisioning, while dynamically provisioning resources for each container requires proactive policies to avoid under-provisioning and Quality of Service (QoS) degradation.

Proper auto-scaling of containers requires understanding the dynamics of resource usage. As every container can run training for a different model, there is no universal policy that applies to all. Also, complex and shifting behaviors are hard to model with classical time-series algorithms, which are not suitable for fitting sudden spikes or random burstiness. However, as individual behaviors become frequent and repetitive, we can learn and leverage such information to develop auto-scaling policies. Thus, we propose techniques for sophisticated time-series behavior modeling, like CRBMs.

Identifying the behavioral patterns of an application is challenging. Even more so when considering the lack of prior knowledge of the complete set of unique patterns to be found. Previous techniques [17] require domain knowledge to label and identify different parts of an application execution, e.g., introduced by the programmer, or map/reduce phases in Spark applications, which are not viable in more general workload, such as deep learning. Besides, the DL training workload includes routines from data loading to computation, showing random spikes in memory and CPU usage. Existing time-series/phase characterization methods (such as ARIMA and ARMA) are not suitable for such random burstiness and sudden changes. Finally, to predict phases to empower preventive auto-scaling, we need a model characterizing the transitions of phases in a DL training session. Through discovering phases from monitoring metrics, we can capture the resource demands that dynamically change over time without the need to be intrusive to the user's ML models.

In this paper, we propose to use a Conditional Restricted Boltzmann Machine (CRBM) to encode the multi-variant time series of resource usage, where we further apply an unsupervised clustering method to discover phases and their behaviors automatically. CRBMs are capable of learning intricate patterns, such as phases with random burstiness and spikes, as observed in DL workloads. Thus, CRBMs based

phase detection can discover such patterns and inspire more accurate resource provisioning accordingly. To explore the viability of a phase prediction based preventive auto-scaling policy, we attempt to use the tree and the graph to model the transition of phases based on all phase sequences learned from our containers.

To summarize our contribution, we build a mechanism to discover behavioral phases from resource usage metrics, to estimate resource demand, and to use such information to devise container auto-scaling policies for deep learning workloads. We evaluated this approach with more than 5500 containers running training jobs in clusters providing IBM DLaaS service. When comparing to naive reactive policies, we show that using our phase detection can reduce over-allocation by up to 30% for total CPU and memory allocation, given a 5% tolerance range. In comparison, both approaches have similar under-provisioning of less than 5% of the time. We also show that the executions of DL training jobs exhibit a set of typical behaviors, where a graph model summarizing the phase transitions coincides with the real stage changes in the DL training process. When comparing the graph model to the tree, we observe that though a tree representation produces "prototypes" (standard execution behaviors), a graph model can embed transition probabilities.

2 Background and Related Work

Previous work on workload characterization [5–7] focuses on using time-series ML techniques to forecast application resource demands. They generally focus on a particular application to select forecasting models or to tune the re-scaling time window. [16] is the first to present an approach using both CRBMs and Hidden Markov Models for discovering Spark application profiles. We adopt this approach and extend it, to reduce resource over-provisioning for DL training jobs. Other works [13] focus on modeling workloads to discover patterns and classify applications, so heterogeneous workloads can be co-located to achieve higher resource utilization. Such works usually use HMMs to identify patterns but require knowing a-priori, namely the number of regimes, so they are not generalizable. More straightforward methods like clustering achieve the same results, also with CRBM generalization.

General workload modeling is a known challenge. Existing approaches usually assume that the arrival of jobs determines the resource demand. [9, 18, 21] focus on Cloud services and how to manage and scale service resource allocation according to the arrival of jobs or clients. Those predictors use ARMA and ARIMA models, well-known for capturing time-dependent structures, including trends and seasonality. Yet they are not able to discover phases through the whole training session of DL models, nor can they capture phases with some randomness in spikes and burstiness. Works like [14] use ML ensembles to forecast workload on job arrivals by memorizing past job runs. These methods cannot be applied to predict the dynamic usage of a particular job.

For DL training jobs, the critical resources needed include CPU, memory, networking, or GPUs. Our modeling is agnostic of the resource type and can be applied as far as container resource usages are monitored as time series. We here only focus on CPU and memory because auto-scaling these resources are practically achievable for containers. Dynamically allocating GPU among containers by nature is hard, so we leave the GPU usage analysis as a future challenge. Networking resource is critical for distributed DL training [11, 15], thus various topology-aware policies are applied to optimize placement of training tasks. However, while existing profiling methods consider manually labeling their code or identifying parts of such DL architectures, a Cloud user may be reluctant to share information about their DL models or specific configurations. Also, as different training sessions use different types of data and models, they are too varied across users and executions. Therefore, we consider a black-box approach.

Major cloud service providers offer Machine Learning (ML) platforms as cloud services to train, debug, store, and deploy a diverse set of ML models at scale. These include IBM Watson Machine Learning (WML) [10], Amazon Sagemaker [19], Microsoft Azure Machine Learning [4], and Google Cloud AI Platform [1]. Cloud computing giants also contribute their solutions for cloud-native machine learning to open source. These efforts include Kubeflow [2] and FfDL [12]. From publicly available documentation about these platforms, we can see that the majority of the above platforms target a cloud-native environment[1], where their model training workloads are containerized. Users only need to pay for the actual usage for running containers, namely vcpu-hours (for a given memory size). However, allocating resources for containers according to their peak usage for all ML training jobs can drain allocatable resources in a cluster quickly. Namely, the actual cluster utilization may be low while other training jobs are waiting in queues. Therefore, understanding the resource usage behavior of ML workloads can help auto-scale containers dynamically and improve the overall resource efficiency.

3 Methodology

The proposed methodology collects container resource usage (i.e., CPU and memory consumption), creates a model encoding these metrics to capture dynamics over the time dimension (behavior), clusters similar behaviors as unique phases, reduces the whole execution to a sequence of phases, and then estimates the resource requests per each phase. Figure 1 depicts the framework of AI4DL.

3.1 Phase Discovery and Detection

The resource usage metrics from the DL training containers, including CPU and memory, are firstly encoded using a Conditional Restricted Boltzmann Machine (CRBM) [20].

[1] Azure Machine Learning targets a dedicated computing environment [4].

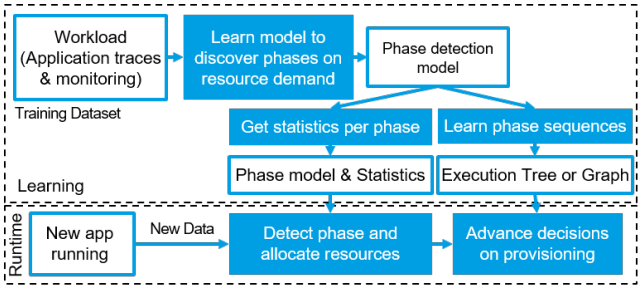

Figure 1: Framework of AI4DL

CRBM is a type of Neural Network, capable of encoding multi-dimensional data into a vector of features, which is ideal for analyzing time-series patterns. Different from hashing functions, CRBMs can produce similar outputs from similar inputs, a property we are exploiting next. CRBM takes container metrics in a sliding time window as input, namely metric at time t plus the history $t-1 \ldots d$, where d represents the *delay* (i.e., a hyper-parameter denoting the size of the time window). The output taken from the CRBM is the encoding from its hidden layer, a vector of h features encoding the input values. Such a vector embeds the instant metrics plus their previous d history steps. Thus, the CRBM can identify behaviors in the time window of $d+1$ steps, where either d (through re-training) or the aggregation time interval of samples is adjustable for long or short training sessions. CRBM at each time step of $1 \ldots T$ produces a $(T-d)$ encoding vector of size h, capturing the *behavior* of the resource usage at each time t.

As CRBM encodes similar behaviors into similar codes, proximity-based clustering methods, like k-means, can be used to group similar behaviors. As the total number of different phases is unknown a-priori, we need a prior analysis to determine k as a hyper-parameter [16]. Via passing real-time container metrics through the encoding and the clustering, the phases can be discovered at each time step. Given the metrics from a full execution of a container, we produce the sequence of phases, retrieve the statistics and the type of behavior for each phase, and leverage it to allocate resources. Notice that, as CRBM embeds time in the encoding vector, a phase behavior does not only describe the magnitude of CPU and memory usage but also their dynamics in the time window. Figure 2 shows the pipeline of phase discovering.

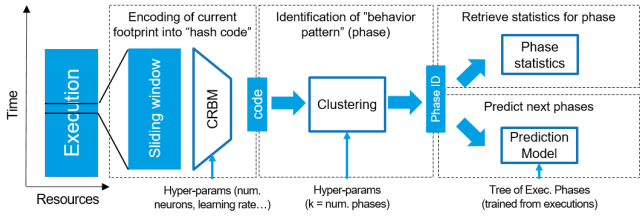

Figure 2: Pipeline of the Characterization Mechanism. Models (CRBM, k-means, and stats) are trained sequentially. For phase discovery, metrics are passed through the pipeline to identify the phase and produce the stats.

3.2 Resource Provisioning Policies

To show the benefit of our phase-detection method in resource allocation, we compare the following policies.

Dynamic policies allocate resources, knowing *a-priori* the consumption of the application given a specific time window. The options considered are *maximum* (maximum observed resources for that period) and *stat.rules* (mean + 2 standard deviations of observed resources for that period).

Adaptive policies allocate resources with the observed *a-posteriori* demand given the previous time window. Each time window is provided resources according to the actual usage in the previous one, considering both *maximum* and *stat.rules*. With sudden changes, such a reactive approach may result in a lag in provisioning, leading to a temporary Quality of Service degradation or an out of memory error to kill the container.

Phase-based policies allocate resources by discovering the current phase of the execution and using current phase statistics to allocate resources. Here we consider *stats.rules* because *maximum* captures the maximum deviations across all containers.

3.3 Modeling Executions as Phase-series

A full life cycle of a container can be encoded as a sequence of phases, representing behavior changes in resource usage. Containers running similar workloads display similar behaviors (e.g., first phases are corresponding to *Memory.load*, next phases to *Intensive.CPU*, last phases to *Memory.unload*). A good representation of phase sequences can indicate what types of behaviors the containers are undergoing, how much resources they need, and in which order they consume resources.

The resource usage of different containers may show similar patterns in their phase sequences. By modeling the phase sequence in a tree or graph, we can observe more intuitively on how variant these phase sequences are across different containers. The wider and more balanced the tree is, the more types of behavioral changes are there in containers. The branches that represent the majority of the phase sequences are common execution flows ("prototypes"). Also, by representing the phase-sequences as a graph, we can read the probabilities of transitioning from phase to phase. Thus the graph helps to infer the sequence of phases. E.g., the auto-scaler can proactively estimate and provision for the bursty future resource demands before the actual usage peaks. Even though we are not predicting future phases in this work, we are observing in our experiments that the variety in our 5000 DL applications is reduced to around 6 prototypical sequences.

4 Experimental Results

4.1 Phase discovery and detection

First, we evaluate phase discovery as follows. We collect resource usage metrics from containers training DL models in clusters that provide the IBM DLaaS service for internal

researchers. We use 5000 container traces for training, and 500 for testing and the modeling process.

Training the CRBMs Encoder: Training a CRBM model is similar to training a regular Neural Network, where the architecture is pre-defined, as well as the number of hidden units. In our time series encoding problem, we also need to specify the window size for the delay, namely the $t_{1...d}$ time window for metric analysis. Besides, the hyper-parameters (i.e., the number of training epochs and the learning rate), the training procedure can be tuned to obtain a more accurate model. The loss function to train the CRBM model is the Square Sum of the Error on data reconstruction (how well data is encoded). We use this loss to perform a search for CRBM hyper-parameters on the DLaaS container resource usage traces. We choose the delay to include the previous $delay = 3$ metric samples. Therefore, if the sampling period for the resource usage metrics is per 15 seconds, the time window to detect phases is one minute, including the current metric data and 3 data points sampled just before it. By tuning hyper-parameters, we find best results (minimum loss) with learning rate $lr = 10^{-3}$, 2000 epochs and 10 hidden units.

Clustering Behaviors: Due to the property of CRBMs that the similarity between inputs and encodings holds, we proceed to cluster the resource usage behaviors in sliding time windows (of 15 seconds + 45 seconds of history). Thus the similitude on resource usage patterns can be learned. Here we use the *k-means* method just for simplicity, as it only requires tuning the number of clusters, k. We proceeded to search for the best k for the lowest Square Sum Within clusters [3], finding that the best k without potential over-fitting is 5.

Phase Information: From generating phase sequences for each execution, we observe trends and different variability in resource usage per discovered phase. Figure 3 displays the principal behaviors of each detected phase and the three most common prototype sequences found. There are typically 5 phases. 1) A phase of warm-up, denoted as phase 3. 2) A phase of memory loading and unloading, denoted as phase 2. 3) A phase for intensive memory but variable CPU usage, denoted as phase 5. 4) A phase for stable use of CPU and memory (phase 1). 5) A phase of gradual increasing memory usage with CPU spikes (phase 4). Notice that phases are identified not only by the current resource usage but with several preceding history data in the time window, which is determined by the delay. For example, in the third prototype, we can see that it takes two steps to stabilize from phase 5 (CPU variation) to phase 1 (CPU stabilization).

From phase statistics, we can see how the resource usage in the warm-up (phase 3) has high variability, as the mechanism can not identify the pattern until having enough metric data (CRBM delay). The difference between CPU in phases 1 and 5 is high vs. low variance. Phase 4 displays a high variation in memory and appears to have an increasing trend for memory usage. Via the tree and graph representation of phase sequences in the following sub-section, we show how these phase representations coincide with the stages of DL model training. However, we can obtain those in an unsupervised way.

4.2 Phase-based Resource Allocation

When applying the produced phases to resource provisioning, we expect to use the statistics (mean and variance) of usage in the phase to allocate resources for the next auto-scaling cycle. We choose an auto-scaling cycle of 10 minutes, which can be configured by cluster administrators. The previously described policies predict how much CPU and Memory should request. Figure 4 compares the actual resource usage with various auto-scaling policies that predict the resource allocations for the testing data-set of DLaaS containers.

The $ground.truth$ indicates the actual resource usage of all containers. Policies $current.step.max$ and $current.step.rules$ (*maximum* and $stat.rules$ for $current.step$) show two resource allocation policies we would apply if we, as an oracle, knew beforehand the resource usage of the forthcoming period. $current.step.max$ is provisioning according to the maximum future usage while $current.step.rules$ is provisioning the average plus double standard deviation. The policies are impractical as they depend on *a-priori* knowledge, but they show the best possible policy that can ever achieve. The rescaling period is every 10 minutes. Policies *prev.step.max* and *prev.step.rules* use the information observed in the previous cycle (*a-posteriori* knowledge) to provision the next cycle, expecting the current pattern to continue in the next auto-scaling cycle. Finally, *phase.rule* shows our methodology that detects the phase right before the provisioning cycle and obtains the information of resource usage in that phase, then provisions resources accordingly in the next cycle.

4.3 Behavior Predictability

Once we obtain phase sequences for different containers, we proceed to create their aggregated execution tree and probabilistic graph. When considering the tree modeling, we find that the variations of phase sequences in a smaller number of containers greatly widen the tree. However, $\sim 90\%$ of the executions fall into one or two branches. By packing the similar sequences of phases, and repeating alternation between phases, the tree is reduced to $20-25$ possible paths, where 6 of them represent $\sim 90\%$ of the executions, namely the discovered prototypes. To represent the phase sequences in the probability graph, we model each unique phase as a node and the transitions between phases as directed edges with attributes showing the transition probability. By examining the graph that models all containers, we can observe the probability of transiting to a particular phase when the current phase is detected. Figure 5 shows the graph and tree generated from the metric data obtained from DLaaS containers.

As an example, from the graph, we can infer that after detecting the warm-up (phase 3, as $\phi 3$) in a container, there is 80% chances that the container will transit to $\phi 2$, indicat-

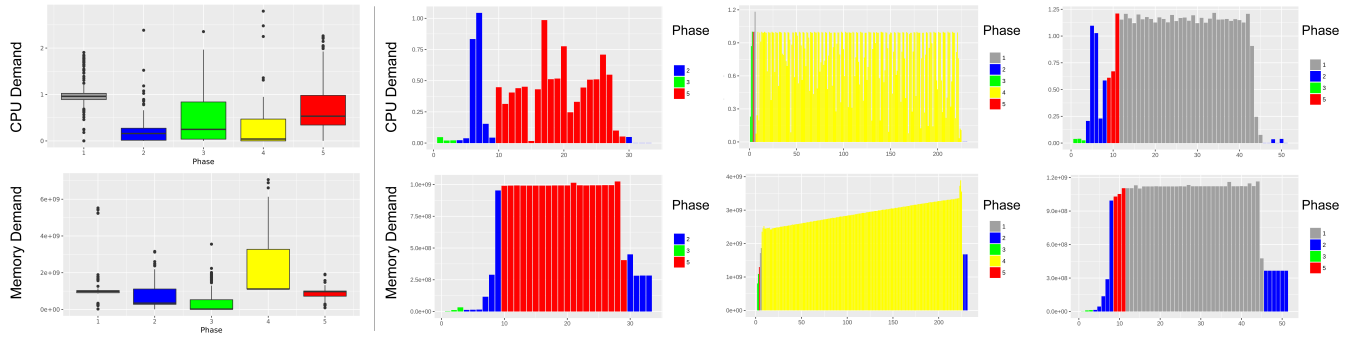

Figure 3: Variance on CPU (top) and Memory (bottom) demand for the discovered phases (color code), and its most frequent prototypes. The phase detects the trend in the last d observations (every 15 seconds) for CPU and memory

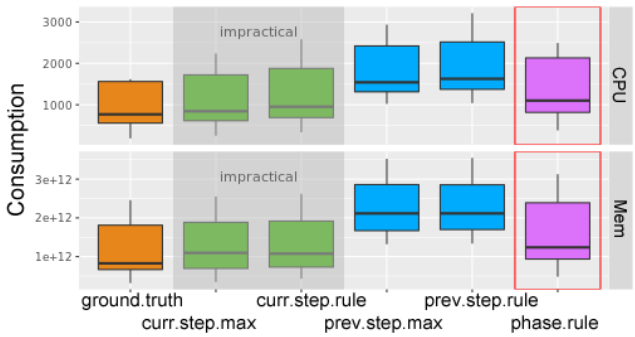

Figure 4: Comparison of the ground truth with the different strategies and with phase detection (ours marked red)

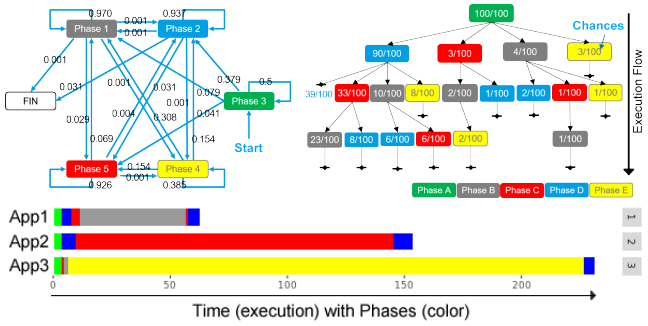

Figure 5: Graph and Tree obtained from DLaaS executions with $exec.time > 10$ minutes, and 3 example executions.

ing an increase in memory usage. As Figure 3 (variability plot) shown, the container is likely to remain in $\phi 2$ until $500MB - 1GB$ is allocated (by its median and 3rd quartile), then transits to $\phi 5$, expecting a demand of 0.5 to 1 CPU. From $\phi 5$, there is a high chance to transit towards $\phi 1$, where it needs at least 1 CPU and similar memory usage. Combining phase transitions in the graph and resource estimation per phase, we can proactively allocate $1GB$ of memory and $0.5CPU$ during $\phi 2$, knowing that at least $1CPU$ must be available in the next period. Additionally, if the phase remains in $\phi 5$, with 0.97 probability, it remains using 0.5 CPU, allowing us to share it with another application in $\phi 5$.

Let's see another example. Given a graph with higher memory, namely the graph is with a node representing a phase with $m > 1$ preceding phases, we find that transitions exiting $\phi 4$ tend to return, indicating that during those phase iterations, memory usage increases to $3.5GB$ with a stable CPU usage of 0.5 to 1 vCPU. By using an adaptive policy, we would under-allocate the resources, discovering the new demand later after entering the phase. With our phase prediction, we can predict the phase and scale-up.

Finally, the above behaviors observed in phases can map to the stages in the DL model training. $\phi 3$ corresponds to the warm-up stage. $\phi 2$ and $\phi 4$ map to the data loading stage. (discovered by $\phi 2$ and $\phi 4$), $\phi 1$ and $\phi 5$ reflect the computational stage, where the container is computing gradients for the DL model. $\phi 2$ can be the parameter unloading stage as well, as it mainly indicates the memory changes.

5 Conclusions

Here we presented a methodology for characterizing Deep Learning workloads based on IBM's DLaaS containers running DL model training. Our characterization model aims at reducing resource over-provisioning and enabling potential auto-scaling for containers. Using CRBMs, we can discover execution phases with specific resource usage behavior, with objectives towards dynamic auto-scaling. Furthermore, containers' resource usage can be clustered in "prototypes" with similar phase sequences, revealing typical execution profiles to be used for resource demand forecasting, resource usage pattern discovery, or resource usage anomaly detection.

The evaluation shows that by detecting phases while running the container, we can adjust resource provisioning dynamically, achieving better resource utilization than naive adaptive policies. Our auto-scaling policies are closer to an ideal oracle, which can foresee the actual resource usage ahead of time. Future work will focus on phase forecasting towards preemptive provisioning, and the generalization of phase modeling to broader types of workloads, like batch and other types of machine learning workloads, etc.

6 Discussion

As this work focuses on a method for characterizing executions in order to enhance application understanding, analysis, and management, one of the principal interests is to identify which scenarios can benefit from the provided approach, in terms of workload, architecture, and resource management policies. Here we focus on Deep Learning workloads, showing characteristic behaviors of CPU and Memory. Other workload types may have other bottleneck resources, e.g., network bandwidth for data-shuffling applications. As different types of resource usages may show different behaviors and demands along time, it is of interest to know which scenarios require which level of modeling sophistication. Do we need more straightforward or more sophisticated modeling methods to complement the one presented here?

Additionally, one of the challenges for this approach is to measure its potential contribution to complex or advanced resource management policies. It is essential to see up to which point those policies can directly benefit from the provided phase information. Knowing the requirements for those advanced resource management policies and strategies is fundamental to adopt this method for a more generalized execution environment.

Moreover, how this approach can be reinforced by predicting future phases, will depend on the extent of application of long-term planners that can utilize the statistics of the predicted phases.

Finally, the presented models are updatable upon training on new datasets and new types of applications. When to trigger the model updates is future work to explore.

Acknowledgment

This work has been partially financed by the Spanish Ministry of Education (G.A.CAS19/00366) and the Ministry of Science (G.A.IJCI-2016-27485).

References

[1] Google cloud ai platform: Create your ai applications once, then run them easily on both gcp and on-premise. https://cloud.google.com/ai-platform. Accessed: 2020-03-13.

[2] Kubeflow 1.0: Cloud-native ml for everyone. https://medium.com/kubeflow/kubeflow-1-0-cloud-native-ml-for-everyone- a3950202751. Accessed: 2020-03-13.

[3] J-means: a new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34(2):405 – 413, 2001.

[4] Microsoft Azure. Azure machine learning: Enterprise-grade machine learning service to build and deploy models faster. https://azure.microsoft.com/en-us/services/machine-learning/. Accessed: 2020-03-15.

[5] S. Baig, W. Iqbal, J. L. Berral, A. Erradi, and D. Carrera. Adaptive prediction models for data center resources utilization estimation. *IEEE Transactions on Network and Service Management*, 16(4):1681–1693, Dec 2019.

[6] S. Baig, W. Iqbal, J. L. Berral, A. Erradi, and D. Carrera. Real-time data center's telemetry reduction and reconstruction using markov chain models. *IEEE Systems Journal*, 13(4):4039–4050, Dec 2019.

[7] S. Baig, W. Iqbal, J. L. Berral, A. Erradi, and D. Carrera. Adaptive sliding windows for improved estimation of data center resource utilization. *Future Generation Computer Systems*, 104:212 – 224, 2020.

[8] Bishwaranjan Bhattacharjee, Scott Boag, Chandani Doshi, Parijat Dube, Ben Herta, Vatche Ishakian, K. R. Jayaram, Rania Khalaf, Avesh Krishna, Yu Bo Li, Vinod Muthusamy, Ruchir Puri, Yufei Ren, Florian Rosenberg, Seetharami R. Seelam, Yandong Wang, Jian Ming Zhang, and Li Zhang. IBM deep learning service. *CoRR*, abs/1709.05871, 2017.

[9] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE Transactions on Cloud Computing*, 3(4):449–458, Oct 2015.

[10] IBM Corporation. Ibm watson machine learning. https://developer.ibm.com/clouddataservices/docs/ibm-watson-machine-learning/. Accessed: 2020-03-15.

[11] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed DNN training. *CoRR*, abs/1905.03960, 2019.

[12] K. R. Jayaram, Vinod Muthusamy, Parijat Dube, Vatche Ishakian, Chen Wang, Benjamin Herta, Scott Boag, Diana Arroyo, Asser Tantawi, Archit Verma, and et al. Ffdl: A flexible multi-tenant deep learning platform. In *Proceedings of the 20th International Middleware Conference*, Middleware '19, page 82–95, New York, NY, USA, 2019. Association for Computing Machinery.

[13] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294, April 2012.

[14] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. Cloudinsight: Utilizing a council of experts to predict future cloud application workloads. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 41–48, July 2018.

[15] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 16–29, New York, NY, USA, 2019. Association for Computing Machinery.

[16] D. B. Prats, J. L. Berral, and D. Carrera. Automatic generation of workload profiles using unsupervised learning pipelines. *IEEE Transactions on Network and Service Management*, 15(1):142–155, March 2018.

[17] Andrea Rosà, Walter Binder, Lydia Y Chen, Marco Gribaudo, and Giuseppe Serazzi. Parsim: A tool for workload modeling and reproduction of parallel applications. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 494–497. IEEE, 2014.

[18] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507, July 2011.

[19] Amazon Web Service. Amazon sagemaker: Machine learning for every developer and data scientist. https://aws.amazon.com/sagemaker/. Accessed: 2020-03-13.

[20] G. W. Taylor, G. E. Hinton, S. T. Roweis, P. B. Schölkopf, and T. Hoffman J. C. Platt. Modeling human motion using binary latent variables. *Advances in Neural Information Processing Systems*, 19:1345–1352, 2007.

[21] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen. Workload predicting-based automatic scaling in service clouds. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 810–815, June 2013.