POLITECNICO
MILANO 1863

Optimizing on-demand GPUs in the Cloud for Deep Learning Applications Training

Arezzo Jahani, Marco Lattuada, Michele Ciavotta, Danilo Ardagna, Edoardo Amaldi, Li Zhang

Arezzo Jahani, Marco Lattuada, Michele Ciavotta, Danilo Ardagna, Edoardo Amaldi, and Li Zhang. Optimizing on-demand gpus in the cloud for deep learning applications training. pages 1–8. IEEE, 2019

The final publication is available via http://dx.doi.org/10.1109/CCCS.2019.8888151

Optimizing on-demand GPUs in the Cloud for Deep Learning Applications Training

Arezoo Jahani
University of Tabriz
Tabriz, Iran
a.jahani@tabrizu.ac.ir

Marco Lattuada
Politecnico di Milano
Milan, Italy
marco.lattuada@polimi.it

Michele Ciavotta
University of Milano-Bicocca
Milan, Italy
michele.ciavotta@unimib.it

Danilo Ardagna
Politecnico di Milano
Milan, Italy
danilo.ardagna@polimi.it

Edoardo Amaldi
Politecnico di Milano
Milan, Italy
edoardo.amaldi@polimi.it

Li Zhang
IBM
Yorktown Heights, NY, USA
zhangli@us.ibm.com

***Abstract*—Deep learning (DL) methods have recently gained popularity and been used in commonplace applications; voice and face recognition, among the others. Despite the growing popularity of DL and the associated hardware acceleration techniques, GPU-based systems still have very high costs. Moreover, while the cloud represents a cost-effective and flexible solution, in large settings operations costs can be further optimized by carefully managing and fostering resource sharing. This work addresses the online joint problem of capacity planning of virtual machines (VMs) and DL training jobs scheduling, and proposes a Mixed Integer Linear Programming (MILP) formulation. In particular, DL jobs are assumed to feature a deadline, while multiple VM types are available from a cloud provider catalog, and each VM has, possibly, multiple GPUs. Our solutions optimize the operations costs by (i) right-sizing the VM capacities; (ii) partitioning the set of GPUs among multiple concurrent jobs running on the same VM, and (iii) determining a deadline-aware job schedule. Our approach is evaluated using an ad-hoc simulator and a prototype environment, and compared against first-principle approaches, resulting in a cost reduction of 45-80%.**

Keywords: Cloud, Scheduling, Optimization models, on-demand GPUs

I. Introduction

Nowadays, deep learning (DL) methods have numerous applications in many fields like voice and face recognition as well as machine translation and classification tasks [1]. Self-driving cars, voice-activated assistants, language recognition [2], as well as brain cancer detection are just a small selection of successful application cases. DL uses common machine learning algorithms as Convolutional Neural Networks (CNNs) or Recursive Neural Networks (RNNs) to identify latent knowledge in the training dataset [3] and to generalize this knowledge through a black-box approach to new data. CNNs reduce the number of architecture parameters employing only a few fully connected layers that, conversely, are broadly adopted in traditional neural networks (NNs). CNNs are usually able to achieve good accuracy by also reducing the number of matrix multiplications required to process massive data sets [4]. RNNs include fewer features when compared to CNNs and, unlike feed-forward NNs, can use their internal memory to process arbitrary sequences of inputs; they are widely used to process time-series and to implement, e.g., speech recognition systems [5] as particularly suitable to treat sequence-related items. 5, since what a user spoke last will impact what he/she will speak next [5].

DL application training is computing intensive and for this reason it is usually backed by GPUs, which can perform matrix multiplications in parallel and are able to accelerate the models' training and evaluation in production environments [6]. As a matter of fact, CNN training is frequently offloaded from CPUs to GPUs [7] achieving a 5 to 40x performance gain [8]. It should not come as a surprise then that the GPU market that was already worth around 200 million USD in 2016 is experiencing an annual growth rate over 30% that analysts believe will remain unchanged until 2024 [9].

Despite all of the mentioned advantages for GPUs, the cost of GPU-based systems is usually high: high-end GPU based servers like NVIDIA DGX-2 cost up to 500k USD [10] and in public clouds GPU based virtual machines (VMs) time unit cost is 5-8x higher than high-end CPU-only VMs [11]. The efficient use of GPUs is hence an important aspect that deserves to be addressed.

This paper aims at addressing the joint capacity planning of multi-nodes GPU-based VMs and DL job scheduling by proposing a Mixed Integer Linear Programming (MILP) formulation. In particular, we assume that DL training jobs have a deadline, and that it is possible to configure multiple shared nodes using VMs of different types, featuring one or possibly multiple GPUs, taken from a cloud provider catalog. Our solution: (i) selects the right-size VM capacity for each node, (ii) partitions multiple GPUs of a VM among multiple jobs that are running

978-1-7281-0875-9/19/$31.00

concurrently on it, and (iii) determines an optimized job schedule.

Interestingly, while multi-job scheduling has been studied in some areas like manufacturing and logistics, to the best of our knowledge, the joint capacity planning and DL job scheduling problem has not been addressed yet.

Our approach is evaluated through an ad-hoc simulator and compared with first principle approaches, namely First-in-First-out, Earliest Deadline First, and Priority Scheduling rules. Results are validated by also performing a test on a real prototype environment. Simulation results show how our solution allows for reducing costs by 45-80% when compared against the best heuristic. Moreover, the gap measured between the real and predicted costs in the prototype environment is less than 7%. Finally, scalability analyses demonstrate that the scheduling of 400 jobs on 40 candidate nodes can be computed in less than 12 minutes, demonstrating that our solution can be used in practice.

The rest of the paper is organized as follows. Section II reviews the related works. Section III describes the problem settings while Section IV introduces the MILP formulation. Experimental results are presented in Section V. At last, Section VI draws some conclusions and outlines future works.

II. Related Work

GPUs can be shared to perform multiple tasks in parallel and, inversely, several GPUs can be assigned to the same job to achieve a significant performance boost. The scenario hugely increases in complexity when the Cloud computing is added to the equation since several types of Virtual Machines (VMs) featuring different GPUs might be available [12], entailing decisions about the number and types of VMs to be used and the sequencing of jobs on the resources (GPU resource management in job scheduling).

This section reviews the relevant works from the literature, broadening the scope to include papers of task scheduling in the High-Performance Computing (HPC) and Cloud computing fields.

There has been considerable work done in the domain of GPU scheduling for HPC systems mainly to improve load balance and performance of both CPU and GPUs [13], [14], and yet few are the solutions addressing AI applications. GPU scheduling in HPC is also considered in [15] where a scheduling algorithm is proposed to improve the GPU utilization in the scenario where multiple applications share the same GPU.

When scheduling tasks to resources, it is common to rely on the idea of a budget. Often published solutions implement a temporal budget, i.e., a specific resource is assigned for different time slices to jobs with different priorities. An example of this policy is presented in [16] where a GPU is assigned considering both posterior and a priori enforcement policies. A resource budget is implemented in [17], where the number of GPU processing cores assigned to a job depends on its priority. Resource over-allocation and under-allocation techniques, designed to improve responsiveness and GPU utilization, respectively, are also presented and discussed. Another budget-constrained scheduler is proposed in [18] to handle large bags of tasks on clouds characterized by different performance and costs. The main objective is the minimization of the completion time. Bag of tasks scheduling is also considered in [19], which proposes a greedy method able to meet the workflow deadline while minimizing the resource leasing costs.

In [20] the authors explore the scenario where existing linear dependency between compute-intensive, stochastic, and deadline-constrained multi-stage jobs exists. The problem is addressed considering three objective functions, namely the number, the usage, and utilization of rented VMs.

A scheduling technique for GPU as a service is proposed in [21] which provides complete management of cloudified GPUs in the public cloud environment. A new concurrent applications scheduling is proposed in [22] based on two states of single node and multi-node jobs.

GPUs virtualization can lead to significant utilization increase [21]. Furthermore, it is also possible to enhance GPU virtualization by adding a new feature to it: remote access. In this way, it is possible to provide GPUs to applications that are being executed in other cluster nodes. This technique, also known as resource disaggregation, allows to transparently share the GPUs of a server among many applications running in different nodes of the cluster with a reduced overhead (less than 4% when a high-performance network fabric is used) [23]. A successful middleware to implement this approach is rCUDA [24], which enables the concurrent remote usage of CUDA-enabled devices in a transparent way. An extensive survey for GPU virtualization techniques and scheduling methods is provided in [25]. Although there exist several scheduling methods to schedule job tasks into GPUs, varying from priority-based to load-balancing-based approaches, they perform fine-grained scheduling, being implemented at hypervisor or OS level.

As regards the DL training, Xiao et al. [26] propose Gandiva, a scheduling framework able to improve latency in training DL models on a GPUs cluster by exploiting heterogeneity and recurrent behaviors of DL jobs while running mini-batch iterations. Finally, a seminal work on scheduling multi-GPUs among competing jobs on high-end servers is [27]. The paper proposes a topology-aware scheduling policy for DL jobs in cloud environments, which provides a placement strategy to schedule jobs on a Power8 machine based on NVLink able to satisfy workload requirements preventing also application interference.

III. Problem Statement

This section aims at introducing the design assumptions and system models used as a reference throughout the paper. Notably, we envision a system where multiple jobs

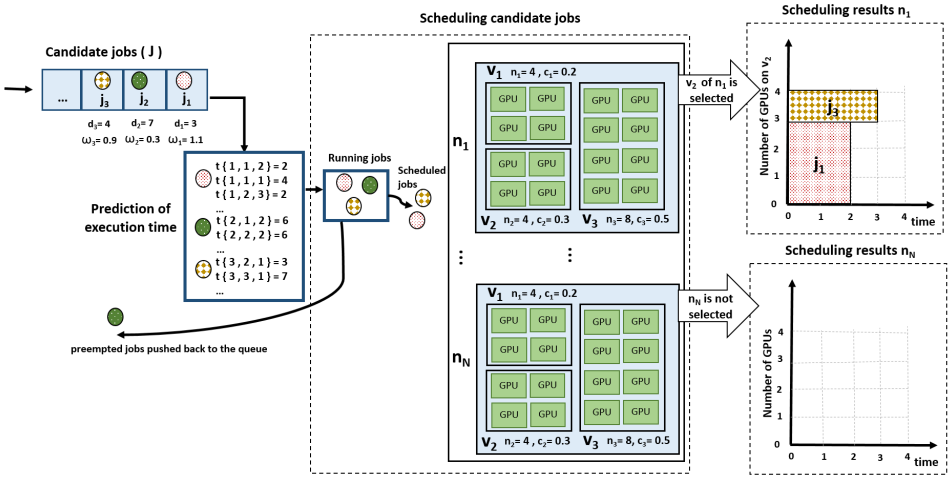


Figure 1: Scheduling problem.

are submitted and run concurrently and where no information about the upcoming jobs is available. A deadline is associated with each job, and a penalty is incurred in case of delay. Multiple nodes can be set up on-demand in the cloud and configured with different VM instance types; each VM features, possibly, multiple GPUs. We have three main goals: (i) determining the best VM type for each node among the ones available in the provider catalog; (ii) partitioning the GPUs among multiple jobs running on the same VM instance; (iii) determining an optimized job schedule.

The joint capacity allocation and jobs scheduling problem is considered in an online setting, and it is solved every time a new job arrives or one of the jobs in execution ends. In substance, when a job terminates, the resources associated are freed and the system should be able to reassign efficiently. Moreover, since training jobs are long-running batch applications, when a job ends or a new one enters the system, the running jobs:

- can continue their execution with the same (or a different) number of GPUs on the same (or on a different) VM type.
- can be preempted and pushed back in a waiting queue.

The system is shown in Figure 1; it can manage up to N nodes that can be individually configured with the VM type v from the cloud provider catalog (denoted by V). Each VM type $v \in V$ features different characteristics such as the number of GPUs s_v, and time unit cost c_v. At a certain point in time the system needs to run a collection of jobs J already submitted and each job $j \in J$ is characterized by a deadline d_j. Jobs are never rejected and can possibly be delayed: the job tardiness is denoted by τ_j. To assign a priority to different training applications, a tardiness weight ω_j is associated with each job j.

Job execution times across different VM types and with a different number of GPUs g can be estimated by relying on our previous work [28], which proposed machine learning models to predict the training time of DL applications. The proposed models are based on linear regression and are able to learn the required execution time for each GPU-based job from a training set of experimental runs of the target DL network rather accurately (with an average percentage error of about 10%). In the following t_{jvg} denotes the predicted execution time of job j when it is running on VM type v with g GPUs.

Finally, in order to cope with performance prediction errors, the problem is solved not only every time an event occurs (a job that ends or is submitted) but also periodically, every H units of time.

Figure 1 shows an example with three jobs (j_1, j_2, and j_3). Their deadlines and tardiness weights are $(3, 7, 4)$ hours and $(1.1, 0.3, 0.9)$, respectively. The N nodes (n_1 to n_N) can be configured with three VM types. Two of them (v_1 and v_2) have four GPUs while v_3 features eight GPUs. Each VM type has its own time unit cost $(0.2, 0.3, 0.5)$ \$/h. Execution time estimates are reported in the box *prediction of execution time*; they are obtained by relying on the approach presented in [28]. The figure shows that only j_1 and j_3 are accepted and run while j_2 is not scheduled in the current time interval and is sent to the waiting queue. Furthermore, only the first node (n_1) is chosen to run the jobs while the others are not started.

IV. Problem Formulation

In this section, we present a Mixed Integer Linear Programming (MILP) formulation for the problem of assigning DL jobs in the cloud considering deployments on heterogeneous nodes, under soft deadline constraints with tardiness costs.

We introduce four input sets: the set of candidate jobs J, the set of nodes N, the set of VM types V, and the set of the possible numbers of available GPUs on each VM v, which is denoted with G_v. In particular, $\forall v \in V$, $G_v = \{1, \ldots, s_v\}$, where s_v is the number of GPUs available on

Table I: Notation

Problem parameters	
J	set of submitted jobs
N	set of nodes
V	set of VM types
G_v	set of available GPUs number on type v VM
s_v	number of GPUs for a type v VM
c_v	time unit cost of a type v VM
d_j	deadline of job j
ω_j	tardiness weight of job j
$\widehat{\omega}_j$	tardiness weight for job j in case it is postponed
M_j	maximum execution time of job j
t_{jvg}	execution time of job j when running on type v VM with g GPUs
M_j	maximum execution time for job j
H	scheduling time interval
μ	a penalty coefficient for unused GPUs

Problem variables	
w_n	1 if node n is chosen and 0 otherwise
y_{nv}	1 if type v VM is chosen on node n and 0 otherwise
z_j	1 if job j is executed and 0 otherwise
x_{jnvg}	1 if job j is running on VM of type v on node n with g GPUs, and 0 otherwise
τ_j	tardiness of job j
$\widehat{\tau_j}$	worst case tardiness of job j if it is postponed

VM type v. For each job j, M_j denotes its maximum execution time, i.e., $M_j = \max_{v,g} t_{jvg}$. H is a constant parameter which denotes the periodic re-scheduling time interval. The adopted notation is summarized in Table I.

We introduce the following decision variables:

- w_n is a binary variable that is set to 1 if node n is selected, and is 0 otherwise.
- y_{nv} is a binary variable that is set to 1 if VM type v is selected on node n, and is set to 0 otherwise.
- z_j is a binary variable that is set to 1 if the job j is executed and 0 otherwise, i.e., the job j is pushed back to the queue.
- x_{jnvg} is a binary variable that set to 1 if job j is run on VM type v of node n with g GPUs, and to 0 otherwise.
- τ_j is a non-negative variable denoting the tardiness of the job j.
- $\widehat{\tau_j}$ is a non-negative variable that corresponds to the *worst case tardiness* of job j, in case job j is pushed back to the queue and reconsidered at the next re-scheduling event.

The joint capacity planning and DL jobs scheduling problem can be formulated as:

$$\min \sum_{j\in J} \omega_j \tau_j + \sum_{j\in J} \widehat{\omega}_j \widehat{\tau_j} + \mu \sum_{n\in N}\sum_{v\in V}\left(s_v\, y_{nv} - \sum_{j\in J}\sum_{g\in G_v} g\, x_{jnvg}\right) + \sum_{j\in J}\sum_{n\in N}\sum_{v\in V}\sum_{g\in G_v} \frac{g}{s_v} c_v\, t_{jvg}\, x_{jnvg}$$

subject to:

$$\sum_{v\in V} y_{nv} = w_n \quad \forall n \in N \tag{1}$$

$$x_{jnvg} \le y_{nv} \quad \forall j \in J,\ \forall n \in N,\ \forall v \in V, \forall g \in G_v \tag{2}$$

$$x_{jnvg} \le z_j \quad \forall j \in J,\ \forall n \in N,\ \forall v \in V, \forall g \in G_v \tag{3}$$

$$\sum_{v\in V}\sum_{g\in G_v} x_{jnvg} \le w_n \quad \forall j \in J,\ \forall n \in N \tag{4}$$

$$\sum_{n\in N}\sum_{v\in V}\sum_{g\in G_v} x_{jnvg} = z_j \quad \forall j \in J \tag{5}$$

$$\sum_{j\in J}\sum_{g\in G_v} g\, x_{jnvg} \le s_v \quad \forall n \in N,\ \forall v \in V \tag{6}$$

$$\sum_{n\in N}\sum_{v\in V}\sum_{g\in G_v} t_{jvg}\, x_{jnvg} \le d_j + \tau_j \quad \forall j \in J \tag{7}$$

$$(H + M_j)(1 - z_j) \le \widehat{\tau_j} + d_j \quad \forall j \in J \tag{8}$$

$$\sum_{n\in N} w_n = min\{|N|, |J|\} \tag{9}$$

$$y_{nv} \in \{0,1\} \quad \forall n \in N,\ \forall v \in V \tag{10}$$

$$z_j \in \{0,1\} \quad \forall j \in J \tag{11}$$

$$x_{jnvg} \in \{0,1\} \quad \forall j \in J,\ \forall n \in N, \forall v \in V, \forall g \in G_v \tag{12}$$

$$\tau_j \ge 0 \quad \forall j \in J \tag{13}$$

$$\widehat{\tau_j} \ge 0 \quad \forall j \in J \tag{14}$$

Constraints (1) enforce that, for each selected node n, exactly one VM type is chosen. Constraints (2) ensure that only deployments on the chosen VM type are feasible while Constraints (3) ensure that only deployments for the executed jobs are feasible. Constraints (4) bound the allocation of jobs to the active nodes. Moreover, Constraints (5) serve the purpose of associating exactly one deployment choice to each executed job. Constraints (6) bound the number of allocated GPUs to the available capacity of the selected VM. g is the number of GPUs that are selected by variable x_{jnvg} for each job and it must not exceed the number of available GPUs s_v of the selected VM type. Constraints (7) try to meet the job deadlines by introducing the tardiness τ_j in case of violation. Constraints (8) define the worst case tardiness $\widehat{\tau_j}$ for jobs whose execution is postponed (characterized by $z_j = 0$). The worst case tardiness will be equal to zero for the jobs selected for execution (with $z_j = 1$) and equal to $(H + M_j - d_j)$ for waiting jobs.

Constraints (9) ensure that the number of selected nodes is equal to the minimum between the number of available nodes and of available jobs so as to force the execution of all available jobs. Notice that, in presence of idle resources, the decision of postponing some jobs to the next time slot cannot reduce their execution costs. Indeed, since jobs can be preempted, postponing their execution will only make their deadlines more strict, possibly requiring additional resources and imposing higher costs to the system. Thus, it is always better to run all available jobs if resources are available. Constraints (10)-(14) define the decision variables' domain.

Finally, concerning the objective function, the first term $\sum_j \omega_j \tau_j$ corresponds to the weighted tardiness of all running jobs, and the second term $\sum_j \widehat{\omega}_j \widehat{\tau_j}$ to the worst-case weighted tardiness of the jobs that are postponed ($\widehat{\omega}_j$

is the tardiness weight associated with jobs that are not executed and we set $\widehat{\omega}_j > \omega_j$ to favor jobs execution and to penalize their deferral). The third term of the objective function $\sum_n \sum_v (s_v y_{nv} - \sum_j \sum_g g x_{jnvg})$ corresponds to the difference between the number of used GPUs and the number of available GPUs from each selected node and μ is a penalty coefficient for unused GPUs. Since the objective function is minimized, the system tends to exploits all available resources for each a node (in this way multi-GPU nodes do not have idle resources). The fourth term of the objective function $\sum_{j,n,v,g} \frac{g}{s_v} c_v \, t_{jvg} \, x_{jnvg}$ corresponds to the sum of the execution cost associated with each job, which is given by the time unit cost of the selected VM type v (c_v) times the execution time of the job evenly sharing the cost of multiple GPUs ($\frac{g t_{jvg}}{s_v}$) (e.g., if a node includes four GPUs and two are allocated to j_1, one is allocated to j_2 and one to j_3, the time unit cost allocated to j_1 is half of the VM cost and is twice the one of j_2 and j_3). As a final consideration, note that we are neglecting overhead costs due to re-configuration costs of running VMs, since this would typically require a time that is orders of magnitude shorter than DL training jobs.

V. Experimental Results

To demonstrate the effectiveness of our optimization model, we evaluated it in a variety of system and application configurations by simulating and performing tests on a large set of randomly generated instances representative of realistic scenarios (presented in Section V-A). In order to evaluate the benefits achievable, our solution is compared in Section V-B with first principle methods such as First-in-First-out (FIFO), Earliest Deadline First (EDF), and Priority Scheduling (PS). Furthermore, we consider the scenario, presented in Section V-C, of implementing the produced solution in a prototype production environment to verify how our approach can be actually exploited in a real system. Finally, the results of a scalability analysis are presented in Section V-D.

A. Experimental setup

As representatives of long-running GPU applications we selected the training of heterogeneous neural networks (i.e., Alexnet, Resnet, VGG, and DeepSpeech) implemented into different deep learning frameworks (i.e., PyTorch and Tensorflow). They are a good representative of the possible applications candidate to be executed in the considered scenarios since they are time-consuming, their execution time significantly benefits from the availability of GPUs, and they can be easily interrupted and moved to other GPUs/VMs by exploiting checkpoints. Moreover, despite belonging to the same class of applications, they present different types of workload. Alexnet is mainly disk-intensive whereas VGG is essentially computational-intensive, and the performance of DeepSpeech is heavily determined by the GPU memory size and speed. Finally, Resnet is characterized by a balanced type of workload.

Table II: Characteristic of the Target Nodes

VM type	GPU type	# GPU	Cost($/hour)
Standard NC6	K80	1	0.56
Standard NC12	K80	2	1.13
Standard NC24	K80	4	2.25
Standard NV6	M60	1	0.62
Standard NV12	M60	2	1.24
Standard NV24	M60	4	2.48
In-house server 1	Quadro P600	2	0.11
In-house server 2	GTX 1080Ti	8	1.13
In-house server 3	Quadro P600	8	0.44

Different instances of such applications have been generated varying the batch size and the number of epochs of the training.

The target executions platforms are six different types of Microsoft Azure VMs and three in-house servers. Table II summarizes the type and number of GPUs available and their hourly costs. The costs of in-house servers have been estimated by considering energy and cooling costs. It is worth noting that the costs of machines with the same type of GPUs are linear in their number: at this granularity (small numbers of GPUs belonging to a large cluster), the economy of scale does not provide any real benefit, so basically, the user pays for GPU usage. This type of economic model contributes to the motivation of adopting the objective function described in Section IV.

To verify the effectiveness and generality of the proposed approach, several random problem instances have been generated. We consider instances with the number of nodes from 1 to 40. The number of submitted jobs in each instance is set equal to 10 times the number of available nodes.

As in other literature proposals, the job inter-arrival times have been generated according to Poisson Distributions [27]. Two classes of instances have been generated, which differ in the mean of the distribution: $30s$ for the first class and $45s$ for the second class. Both the means are smaller than the execution time of shortest jobs: in this way, multiple jobs are loaded in the system in each time slot. For each combination of the number of nodes and of the inter-arrival average time, five different instances are built. The other parameters are set as follows. The re-scheduling time interval (H) is set to one hour. The deadline d_j for each job is randomly generated according to a uniform distribution in the range $[min(t_{jvg}), 2 \cdot max(t_{jvg})]$. The tardiness weights ω_j are randomly generated in the interval $[0.36, 1.08]$ \$/hour with a uniform distribution. Finally, the worst case tardiness weight $\widehat{\omega}_j$ is set equal to $100 \cdot \omega_j$ and the μ parameter is set equal to 1 (given the objective function set in the problem formulation, any positive value forces the use of all available GPUs).

The proposed methodology, the generator of the random instances, and an ad-hoc event-based simulator have been implemented in a Python library which relies on Gurobi Optimizer 8.0. We set the relative mixed integer programming gap (the difference between the current upper and lower bounds of the MILP solver) to 5%. All the results

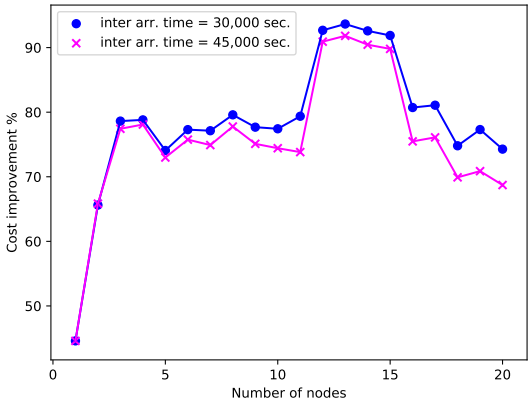


Figure 2: Improvement of the proposed model against best heuristic

have been collected by running the implementation on an Intel Xeon Silver 4114 server exploiting 12 cores and 32 GB of memory.

B. Comparison with First Principle Models

In this section, we compare the results achieved by our method against First-in-First-out (FIFO), Earliest Deadline First (EDF), and Priority Scheduling (PS) (this last based on the tardiness weights ω_j). All first principle methods rely on our model to right-sizing the VMs: the configuration is obtained by exploiting the model with the number of nodes $|N|$ and number of candidate jobs $|J|$ set to 1. For the sake of brevity, we report the average results across the different instances generated for each scenario. Figure 2 reports the average savings that can be achieved by the proposed model against the best first principle result, which is computed as:

$$\frac{cost_{best\ heuristic} - cost_{proposed\ model}}{cost_{best\ heuristic}} \cdot 100 \qquad (15)$$

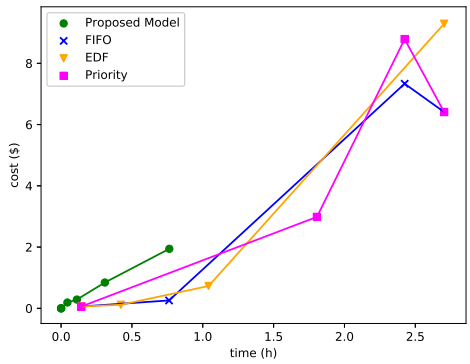
As shown in Figure 2, the model has about 45% improvement against the best heuristic in the $|N| = 1$ scenario, which grows to about 65% in the $|N| = 2$ scenario and then to about $[70, 80]$ for $|N| > 2$ with a peak of more than 90% when N is between 12 and 15.

Figure 3 presents the results regarding the costs for time slots on two representative examples. Figure 3a reports the results for a small size problem with 1 node, 4 jobs and $600s$ inter-arrival time while Figure 3b reports the results for a large size problem with 3 nodes, 30 jobs, and $45s$ inter-arrival time. The results show how the proposed model has always a smaller cost than the others. The reported cost in each time slot is the sum of the per job cost of the used VM (weighted on the number of used GPUs) in the time slot plus the tardiness cost (of any ending job) computed as follows:

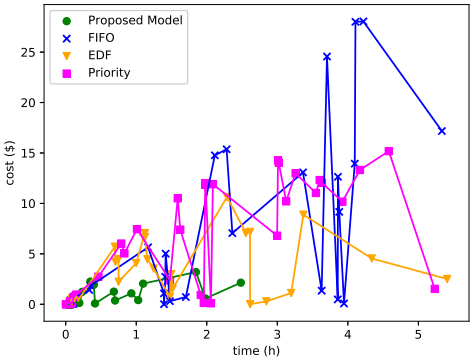
$$cost_j = c_v t_{slot} \cdot \frac{g\, x_{jnvg}}{n_v} + \omega_j\, \tau_j \qquad (16)$$

Table III: Total cost of the proposed model and heuristics

	total cost [$]			
Instance	Proposed	FIFO	EDF	PS
Small size problem	3.26	14.06	10.19	18.25
Large size problem	18.37	20.73	94.37	21.51



(a) Small size problem: 1 node and 3 jobs



(b) Large size problem: 3 nodes and 30 jobs

Figure 3: Cost per time slot of the Proposed Model against first principle heuristics.

where t_{slot} denotes the time slot duration. To better compare the results obtained on the examples of Figure 3, the cost of the complete schedule is shown in Table III. The results show how the total cost of the proposed model is always lower than that of the other first principle models both in the small size and in the large size problem.

C. Experiment on the real system

The effectiveness of our optimization model is also evaluated on a real prototype environment deployed on Microsoft Azure. In this scenario, we consider four application (listed in Table IV), one node with six VM types (NC6, NC12, NC24, NV6, NV12, and NV24), which include two different types of GPUs,

Jobs are submitted every 300 seconds, their tardiness weights and deadlines are randomly selected in the ranges described in Section V-A. The overall schedule computed in the different time slots with the proposed model is depicted in Figure 4. The first time slot starts when $JJ1$

Table IV: Applications of the real system

Job	Application	Images	Epochs	Batchsize
$JJ1$	VGG19 (PyTorch)	130,000	1	32
$JJ2$	VGG19 (TensorFlow)	130,000	1	32
$JJ3$	Alexnet (PyTorch)	130,000	12	256
$JJ4$	Resnet50 (PyTorch)	130,000	3	64

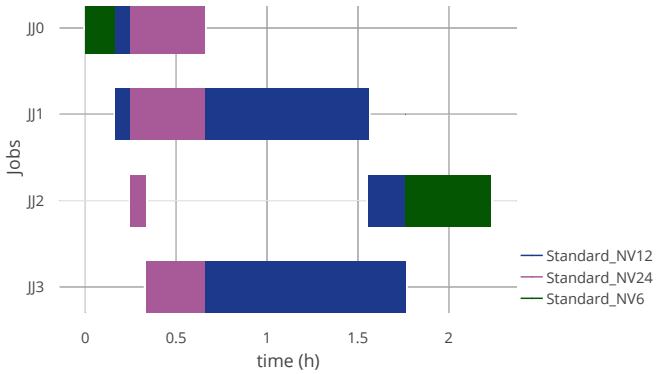


Figure 4: Job-Node Gantt representing the execution in the real system

Table V: Real/predicted cost per slot in the real system

Slot	Predicted cost	Real cost
1	0.21	0.21
2	0.21	0.21
3	0.43	0.43
4	1.66	2.47
5	4.70	4.25
6	3.61	3.70
7	1.65	2.03
sum	12.47	13.30
Overall Difference		6.61%

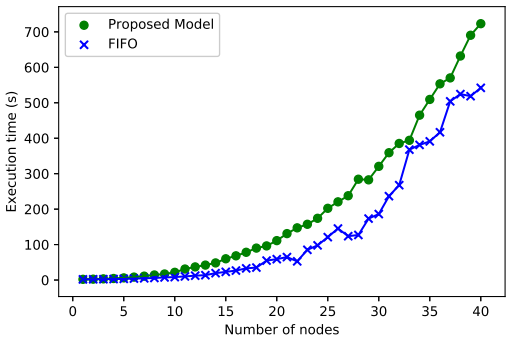


Figure 5: Proposed model vs FIFO: scalability results

is submitted and, for the only available node n_1, Standard NV6 is selected. After $300s$ $JJ2$ is submitted: according to the computed solution, Standard NV6 is shut down and an instance of Standard NV12 is power up where $JJ2$ starts its execution using 1 GPU while the other is used by $JJ1$ After other $300s$ $JJ3$ is submitted: again the currently used VM is shut down and a Standard NV24 is booted. Two of its four GPUs are assigned to $JJ1$, while $JJ2$ and $JJ3$ take one each. $900s$ after the submission of the first job, the last one is submitted. Differently from the previous time slots, the only running VM (Standard NV24) is not stopped. Nevertheless, $JJ3$ is preempted and its execution is suspended assigning its GPU to $JJ4$. Since no new job is submitted, the current slot ends when a job ($JJ1$) terminates. According to the solution identified for the next slot, Standard NV24 is powered off and Standard NV12 is started again. $JJ2$ and $JJ4$ continue their execution while $JJ3$ remains suspended. At the end of execution of $JJ2$ a new slot starts: in the next slot Standard NV12 execute $JJ3$ and $JJ4$. Finally, after the end of $JJ4$, the last slot starts where the only remaining job ($JJ3$) is executed on an instance of Standard NV6. What it has been just described is the simulated scenario built by combining the solutions computed by the proposed model at each time slot based on the available information. When the described scenario is actually implemented and run on the real system, while the submission of the jobs and the configurations of the different slots remain the same, the starting times of slot 5 and of the followings change. This time depends on the ending time of $JJ1$ which is estimated by the proposed model on the basis of a performance model. The real end of $JJ1$ is postponed with respect to the estimated one because of the overhead required to boot the initial VM (Standard NV 6) and to move the application from one VM instance to another (from NV6 to NV12 and then from NV12 to NV24). This overhead mainly includes the time required to boot the different VMs and to re-execute the training of the Neural Network from the last checkpoint. For this reason, the length of the slots starting from slot 4 and, consequently, of their costs in the simulated scenario and in the real vary. Cost details are presented in Table V: as expected, while costs for the first three time slots are the same, the other differ. For example, $JJ1$ terminates later than expected, resulting in a longer time slot 4 and, therefore, in higher costs. However, not all slots are longer: for example, a larger length of time slot 4 provokes a shorter time slot 5.

As shown in Table V, the predicted and real costs of the system are quite close to each other, and the proposed model can complete the scheduling process with a computed cost near to the real cost. The deviation between the predicted and real cost is less than 7% which is small in fact. As observed above, the deviations are mainly due to the fact that the optimization model neglects the VM setup time and switch. Since this is an accelerated experiment running less than 3 hours, this type of inaccuracies in real (longer) runs is less significant resulting in smaller error. This shows that our approach can be applied in practice.

D. Scalability analysis

This section presents the scalability results to show how the model scales with an increasing number of nodes and jobs. In such type of analysis, all the jobs are submitted simultaneously at time 0 and only the solution for the first slot is computed. In this campaign, we systematically explore scenarios with the number of nodes between 1

to 40 while the number of jobs is set to 10 times the number of nodes. The obtained results are summarized in Figure 5: even if the proposed model requires more time than heuristics like FIFO to find a solution, it still scales linearly and it takes in the worst considered scenario (40 nodes and 400 jobs) less than 12 minutes to find the optimal solution, making the proposed method feasible also for large systems.

VI. Conclusions and future work

In this paper, we have proposed a MILP formulation for the online joint capacity planning of on-demand VMs and DL training job scheduling in cloud deployments. The effectiveness of the proposed model has been assessed by performing simulations and experiments in a real prototype environment. The results have shown how our solution allows obtaining savings in 45-80% range with respect to first principle scheduling methods, while the deviation of the expected costs in real systems is below 7%. The solution is effective even for large-size problem instances that can be optimally solved within 12 minutes. Future work will focus on further improving the scalability of the approach in order to manage very large scale data center clusters and disaggregated hardware resources.

Acknowledgement

This work has been partially funded by ATMOSPHERE (grant agreement no. 777154), a Research and Innovation Action funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação, RNP/Brazil.

References

[1] L. Zhong, L. Hu, and H. Zhou, "Deep learning based multi-temporal crop classification," *Remote sensing of environment*, vol. 221, pp. 430–443, 2019.

[2] W. Wang, W. Song, C. Chen, Z. Zhang, and Y. Xin, "I-vector features and deep neural network modeling for language recognition," *Procedia Computer Science*, vol. 147, pp. 36–43, 2019.

[3] A. Liu and Y. Laili, "Balance gate controlled deep neural network," *Neurocomputing*, vol. 320, pp. 183–194, 2018.

[4] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep neural networks with weighted spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018.

[5] X. Zhang, F. Chen, and R. Huang, "A combination of rnn and cnn for attention-based relation classification," *Procedia computer science*, vol. 131, pp. 911–917, 2018.

[6] S. H. Khan, M. Hayat, and F. Porikli, "Regularization of deep neural networks with spectral dropout," *Neural Networks*, vol. 110, pp. 82–90, 2019.

[7] T. Kim, O.-S. Kwon, and J. Song, "Response prediction of nonlinear hysteretic systems by deep neural networks," *Neural Networks*, vol. 111, pp. 1–10, 2019.

[8] S. Madougou, A. Varbanescu, C. de Laat, and R. van Nieuwpoort, "The landscape of gpgpu performance modeling tools," *Parallel Computing*, vol. 56, pp. 18–33, 2016.

[9] G. M. Insights, "Gpu as a service market size by product," *Available: www.gminsights.com/industry-analysis/gpu-as-a-servicemarket*.

[10] "Nvidia tesla gpu servers (gpx)," *https://www.thinkmate.com/systems/servers/gpx*.

[11] "Nvidia virtual gpu technology," *https://www.nvidia.com/en-us/design-visualization/technologies/virtual-gpu/*.

[12] H. Tan, Y. Tan, X. He, K. Li, and K. Li, "A virtual multi-channel gpu fair scheduling method for virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 257–270, 2019.

[13] J. Wu and B. Hong, "Collocating cpu-only jobs with gpu-assisted jobs on gpu-assisted hpc," in *CCGrid, 2013 13th IEEE/ACM International Symposium on*, pp. 418–425, IEEE, 2013.

[14] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing gpu concurrency in heterogeneous architectures," in *Microarchitecture, 47th Annual IEEE/ACM International Symposium on*, pp. 114–126, IEEE, 2014.

[15] C. Reano, F. Silla, D. S. Nikolopoulos, and B. Varghese, "Intra-node memory safe gpu co-scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 5, pp. 1089–1102, 2018.

[16] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proc. USENIX ATC*, pp. 17–30, 2011.

[17] Y. Kang, W. Joo, S. Lee, and D. Shin, "Priority-driven spatial resource sharing scheduling for embedded graphics processing units," *Journal of Systems Architecture*, vol. 76, pp. 17–27, 2017.

[18] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 351–359, IEEE, 2010.

[19] Z. Cai, X. Li, R. Ruiz, and Q. Li, "A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds," *Future Generation Computer Systems*, vol. 71, pp. 57–72, 2017.

[20] J. Zhu, X. Li, R. Ruiz, and X. Xu, "Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1401–1415, 2018.

[21] S. Iserte, R. Peña-Ortiz, J. Gutiérrez-Aguado, J. M. Claver, and R. Mayo, "Gsaas: A service to cloudify and schedule gpus," *IEEE Access*, vol. 6, pp. 39762–39774, 2018.

[22] V. T. Ravi, M. Becchi, W. Jiang, G. Agrawal, and S. Chakradhar, "Scheduling concurrent applications on a cluster of cpu-gpu nodes," in *CCGrid*, pp. 140–147, IEEE, 2012.

[23] C. Reaño, F. Silla, A. Castelló, A. J. Peña, R. Mayo, E. S. Quintana-Ortí, and J. Duato, "Improving the user experience of the rcuda remote gpu virtualization framework," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 14, pp. 3746–3770, 2015.

[24] "rcuda selected as one of the top 5 cuda," *http://www.rcuda.net/*.

[25] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "Gpu virtualization and scheduling methods: a comprehensive survey," *CSUR*, vol. 50, no. 3, p. 35, 2017.

[26] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX*, pp. 595–610, 2018.

[27] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware gpu scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing*, p. 17, ACM, 2017.

[28] E. Gianniti, L. Zhang, and D. Ardagna, "Performance prediction of gpu-based deep learning applications," in *Proceedings of the 9th International Conference on Cloud Computing and Services Science, CLOSER 2019, Heraklion, Crete, Greece, May 2-4, 2019.*, pp. 279–286, 2019.