Hierarchical Scheduling in on-demand GPU-as-a-Service Systems

Federica Filippini, Marco Lattuada
Dipartimento di Elettronica Informazione e Bioingeneria
Politecnico di Milano
Milan, Italy
name.lastname@polimi.it

Arezoo Jahani
Faculty of Electrical and Computer Engineering
University of Tabriz
Tabriz, Iran
a.jahani@tabrizu.ac.ir

Michele Ciavotta
Dipartimento di Informatica Sistemistica e Comunicazione
Università degli studi di Milano-Bicocca
Milan, Italy
michele.ciavotta@unimib.it

Danilo Ardagna, Edoardo Amaldi
Dipartimento di Elettronica Informazione e Bioingeneria
Politecnico di Milano
Milan, Italy
name.lastname@polimi.it

***Abstract*—Deep learning (DL) methods have recently gained popularity. Training this class of models is, however, computing-intensive, and frequently GPUs are used to boost performance. Although the costs of GPU-based systems are gradually reducing due to the high demand, they are still prohibitive: in public clouds, GPU-powered virtual machines (VMs) time unit price is 5-8x higher than CPU-only VMs. While the cloud remains the most cost-effective and flexible deployment, operation costs can be reduced, in large settings, by rightsizing and sharing resources among multiple processes. This work addresses the online joint capacity planning and job scheduling with due dates problem and proposes alternative matheuristic solution methods. Our objective is to optimize operation costs by: i) rightsizing the VM capacities at each node, ii) partitioning the set of GPUs among multiple concurrent jobs on the same VM, and iii) determining a due-date-aware job schedule. The effectiveness of the proposed hierarchical approach, coupled with an appropriate Mixed Integer Linear Programming formulation, is validated against first-principle methods by relying on simulation. The experiments prove that the efficiency of GPU-based systems evaluated in terms of costs can be improved by 50-70%. Finally, scalability analyses show that the proposed approach enables to solve problem instances with up to 100 nodes in less than one minute on average, making it suitable for practical scenarios.**

***Index Terms*—On-demand GPUs, Cloud, Scheduling, Optimization.**

I. Introduction

The introduction of the General Purpose computation on Graphic Processing Units (GPGPU), providing an interface to massive parallelism, has significantly extended the set of previously intractable problems that can be solved within a reasonable time. However, the opportunities offered by these platforms and development frameworks, which made an unprecedented computing capability accessible, have sparked a gold rush that generates an unrelenting demand for computational power. Consequently, GPU as a service market, worth over 700 million USD in 2019, is expected to grow with a compound annual rate of over 38% up to 2024 [7].

A field that has markedly benefited from the continuous evolution of GPUs (along with the libraries that harness their power) is Deep Learning (DL) on Neural Networks (NNs), which nowadays fuels a plethora of applications like voice and face recognition, or self-driving cars [25], [28].

The importance of DL training jobs is evidenced by the fact that the latest hardware generations provide purpose-built components (e.g., Nvidia Tensor cores) that further reduce training time. However, despite hardware improvements, constant efforts to enhance the efficiency of neural models, and the performance boost due to the transition to GPU as AI accelerator (5 to 40 times faster than using the CPU alone [2], [15]), DL applications training is still computationally burdensome.

Despite all the benefits mentioned above [5], the adoption of GPU acceleration on a large scale is still limited by their high cost [4], which is only affordable for large organizations. Indeed, high-end GPU-based servers like NVIDIA DGX-2 cost up to 500 thousand USD [16], whereas in public clouds, GPU-based Virtual Machines (VMs) time unit cost is 5-8x higher than high-end CPU-only VMs [17]. Given the relevance that DL applications are acquiring and the significant training costs involved, the efficient use of GPUs is, therefore, especially important and challenging and calls for an integrated solution that fosters resource sharing and virtualization as major enablers. In particular, we envision a scenario where multiple DL training jobs are continuously submitted for execution on a cluster of virtual machines (in this work, the terms VM and node are used interchangeably). Individual nodes can be configured from a variety of VM types available from the cloud provider's catalog and each type features, possibly, several GPUs. More than one job can run on the same node, and, in this case, available resources are partitioned and statically allotted to avoid interference. Each job is characterized by a due date, a tardiness cost, i.e., a penalty cost proportional to the difference between the job completion time and the due date, and its priority. The set of jobs to be scheduled is not known in advance: new

jobs are submitted with different characteristics, deadlines, and tardiness without any repetition scheme resulting in an online problem. Finally, job preemption is allowed to manage higher priority submissions. The resulting online resource allocation and scheduling problem aims at minimizing the overall job execution costs (the sum of the costs incurred to run the VMs plus the tardiness penalty costs) over the considered time horizon. Online decisions concerns the selection of the VM type for each node, the order in which the jobs are executed, and how resources are partitioned and assigned to each job.

This paper extends our previous work [10] which proposed a preliminary formulation and a solution applicable, due to scalability issues, only to problem instances with a small number of nodes and jobs. In this paper, we propose a hierarchical approach coupled with a novel Mixed Integer Linear Programming (MILP) formulation to overcome such limitations. The experiments show that the efficiency of GPU-based systems, evaluated in terms of costs, can be improved by 50-70% compared to first-principle methods (i.e., first-in-first-out, earliest-deadline-first, and priority). Scalability results assess the feasibility to the novel approach for practical scenarios, since instances with up to 100 nodes are handled in less than one minute on average.

The rest of the paper is organized as follows. Section II reviews the related work. Section III describes the proposed novel hierarchical framework whose MILP formulation is detailed in Section IV. Section V presents the experimental setup and the results of comparing the proposed method against first-principle approaches and our previous solution [10]. Section VI draws conclusions and outlines future works.

II. Related work

Considering the relatively slow increase of per-core computation power witnessed in the last decade, the natural way to get a substantial computation speed up is to resort to a higher degree of parallelism like the one achievable by using one or more GPUs. Yet, while the use of GPU farms delivers unprecedented computing power, such potential is still difficult to harness [23] and new challenges arise in GPU resource management. Among others, in this context, the job scheduling problem on multiple GPUs in virtualized environments calls for both a robust theoretical framework and viable practical solutions [24]. To the best of our knowledge, our work represents the first attempt to tackle the problem of online DL job scheduling on multiple virtualized GPUs. Therefore, we briefly review previous work on the GPU scheduling problem in cloud computing, High-Performance Computing (HPC) as well as in virtualized environments.

Considerable attention has been devoted to GPU scheduling in HPC systems to improve load balance and performance of CPU and GPUs (see e.g. [13], [26]) but there are a handful of solutions targeting specifically DL training applications. A scheduling algorithm based on collocating CPU-only jobs with GPU-assisted jobs is presented in [26]. GPU scheduling in HPC is also considered in [21] where the utilization is improved using an optimized scheduling algorithm allowing multiple applications to share the same GPU.

Scheduling problems are often tackled by defining policies concerning the allocation of a resource budget to tasks. The common budget is time budget: in [12], different time budgets are assigned to GPU tasks with different priorities. A resource budget-based policy is implemented in [11] where GPU processing cores are assigned only to high priority tasks. Another budget-constrained scheduler is proposed in [18] to schedule a large bag of tasks on clouds that have various CPU performance and cost. The information about completing time of the jobs is either known a priori or estimated at run-time. Bag of tasks scheduling is also considered in [3] which proposed a deadline-aware greedy method that minimizes the resource renting costs estimating the real task execution time. In [29] the authors explore a scenario where there exists a linear dependency between compute-intensive, stochastic, and deadline-constrained multi-stage jobs. 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 [9] where full management of cloudified GPUs in a public cloud environment is provided. GPUs virtualization can lead to a significant increase in utilization [9], which can be further improved by providing remote GPU access to applications that are executed on different cluster nodes with reduced overhead [20]. A successful middleware to implement this approach is rCUDA [22], which enables the concurrent remote usage of CUDA-enabled devices transparently. See [8] for an extensive survey for GPU virtualization techniques and scheduling methods.

As regards the DL training, the work in [27] proposes 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. A seminal work on scheduling multi-GPUs among competing jobs on high-end servers is [1]. The paper proposes a topology-aware scheduling policy for DL jobs in cloud environments, which provides a placement strategy able to satisfy workload requirements preventing also application interference. Finally, Optimus, a Kubernetes scheduler especially designed to manage DL jobs on a shared distributed containerized environment, is presented in [19]. The proposed approach aims at minimizing job training time exploiting online resource-performance models to estimate job execution times.

III. Proposed framework

This paper proposes a hierarchical framework and a new MILP formulation for the management of DL training jobs on a GPU-based virtual machine cluster. The same problem was addressed in our previous work [10] in a

centralized framework, which however presented scalability limitations that prevented its usage for instances with more than 40 nodes. The hierarchical approach proposed here aims to overcome these scalability issues, enabling the method to tackle real-world larger instances. The efficient GPU use, the right VM type selection and jobs scheduling are the main issues in this context. We consider a system where multiple jobs are submitted and run concurrently. Each job is associated with a due date and a penalty is incurred in case it is exceeded. Each job can be allocated on a single node, which in turn can execute different jobs at once. Several types of GPU-powered VMs are available and the number of nodes simultaneously in use in the cluster is bounded. Moreover, jobs can be preempted.

Incoming jobs (see Figure 1) are submitted to a central queue, that forwards them to local queues associated with cluster partitions according to a Round Robin (RR) policy. Each local queue is managed by a local controller, which can also boot up and power down VMs of its partition. Each VM can be configured according to different VM types, which feature, possibly, multiple GPUs. Each local controller k has three main goals: i) selecting, from the provider catalog, the most suitable VM type for each node, ii) determining which jobs must be executed and which must wait in the queue, iii) partitioning, at node level, the available GPUs among the running jobs.

Each local controller solves, therefore, a joint Capacity Allocation (CA) and Jobs Scheduling (JS) problem in an online setting. This problem is solved both periodically and every time a new job is submitted or one of the running jobs completes. Moreover, since training operations are long-running batch applications, when a job ends, or a new one enters the local queue, the running jobs: i) can continue their execution with the same or a different number of GPUs on the same VM, on another VM already running, or on a newly instantiated VM that replaces one among the previously running; ii) can be preempted and pushed back in the local queue to be resumed at one of the following decision points.

The reference system is shown in Figure 1: we assume that up to N nodes can be started and can be individually configured with a VM type v from the provider's catalog $\mathcal{V}$. Overall, K controllers manage one local queue and node partition with N/K VM instances possibly of different types. Each VM type $v \in \mathcal{V}$ has distinctive characteristics such as the number of GPUs G_v, and time unit cost c_v. At each decision point the system needs to run a collection of available jobs $\mathcal{J}$. Each job $j \in \mathcal{J}$ is characterized by a submission time, a due date d_j and a tardiness weight ω_j. Jobs are never rejected and can be delayed. The job tardiness is denoted by τ_j.

Job execution time, across different VM types and number of GPUs g assigned, can be reliably estimated (e.g., our previous work [6] proposed machine learning models to predict the training time of DL applications with an average percentage error below 11%). In the following, t_{jvg} denotes the execution time of job j when it is running on a node of VM type v with g GPUs. Moreover, to take into account performance prediction variations, the problem is also solved periodically every H time unit.

Figure 1 shows a driving example featuring six DL training jobs (j_1-j_6). The N nodes (n_1-n_N) can be configured with VMs of three types. Two types (v_1 and v_2) have four GPUs while v_3 has 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 blue box and are obtained by relying on machine learning models [6], whose main inputs are the number of iterations to run and the job batch size. The local controller k schedules jobs j_2 and j_5 to run on the same node of VM type v_2, with 4 GPUs. Job j_5 ends first; the problem is solved again by the local controller: job j_2 can be preempted or can continue in the next time slot, either with the same configuration (same VM type but possibly changed number of GPUs) or with a different VM type, or can be pushed back to the queue to be resumed in a future time slot. The same happens when a new job joins the local queue.

Each local controller k solves a MILP to minimize the total operation costs of its node partition taking into account leasing and tardiness related costs. The new proposed model is described in detail in the next section.

IV. Problem formulation

The novel MILP formulation we propose in this paper extends our previous work [10]. A challenging aspect of the problem that was not satisfactorily delt with in [10] is the evaluation of the deployment costs, trying to capture the online nature of the problem. In an online context, the MILP is solved every time a job is completed or a new job is submitted, but no information about the next submissions are available, so that the completion of already running jobs is the only predictable event determining a rescheduling. Therefore, here we bind the execution costs to those of the first-ending job on each node. Although this approach might seem short sighted, as it optimizes the resources allocated to the shortest job while the others receive a best-effort assignment, we shall see in Section V that it turns out to be effective, since the joint CA and JS problem is repeatedly solved and resources are reallocated every time a job terminates. In particular, the novel MILP formulation we propose in this paper not only overcomes the scalability issues of the model in [10] but also achieves a slight improvement in the total cost.

We consider four input sets: the set of candidate jobs $\mathcal{J}$, the set of nodes $\mathcal{N}$ ($|\mathcal{N}| = N$), the set of VM types $\mathcal{V}$, and the set of GPU partitions for each VM type v, denoted by $\mathcal{G}_v$. In particular, assuming homogeneous GPUs $\forall v \in \mathcal{V},\ \mathcal{G}_v = \{1, \ldots, G_v\}$, where G_v is the total number of GPUs available on VM type v. Since a single VM type can be selected for each node, the set $\mathcal{N}$ is often referred to, in the following, as set of assignable VMs. Jobs are partitioned across the local controllers according to

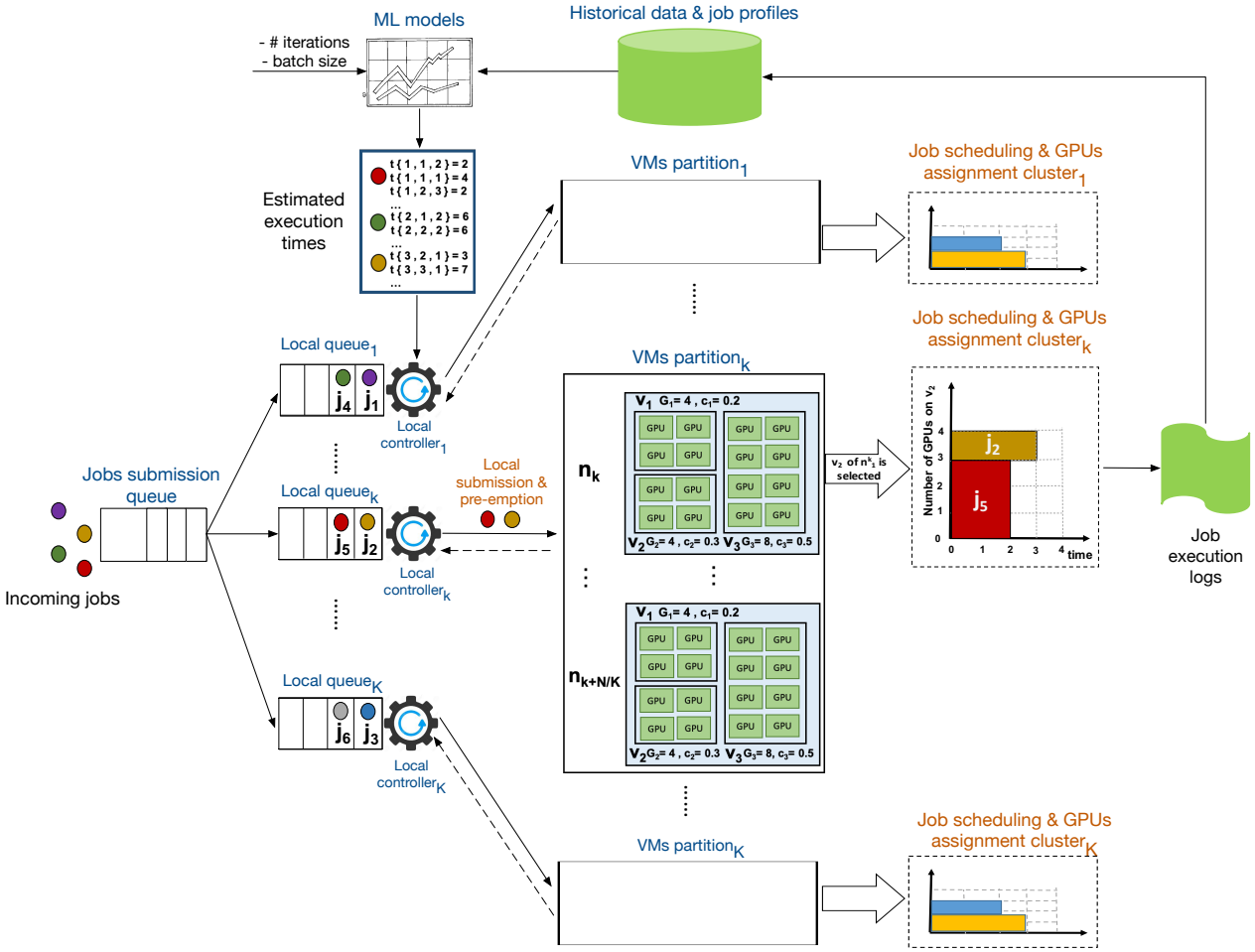

Figure 1: Reference framework.

the RR policy. The index sets for job and node partitions assigned to the local controller k are denoted by $\mathcal{J}_k$ and $\mathcal{N}_k$, respectively. The notation is summarized in Table I.

For each job $j \in \mathcal{J}$, M_j denotes its maximum execution time, i.e., $M_j = \max_{v,g} t_{jvg}$, over every possible VM type v and GPU partition $g \in \mathcal{G}_v$, while $\widehat{M}_j$ denotes its maximum execution cost, i.e., $\widehat{M}_j = \max_{v,g} t_{jvg} c_v$. The constant parameter H represents the periodic scheduling time interval.

The proposed MILP formulation for the local controller k is as follows:

$$\min \sum_{j\in\mathcal{J}_k} \omega_j \left(\tau_j^k + \rho\widehat{\tau}_j^k\right) + \mu \sum_{\substack{n\in\mathcal{N}_k \\ v\in\mathcal{V}}} \left(G_v y_{nv}^k - \sum_{\substack{j\in\mathcal{J}_k \\ g\in\mathcal{G}_v}} g x_{jnvg}^k\right) + \sum_{\substack{j\in\mathcal{J}_k \\ n\in\mathcal{N}_k}} \alpha_{jn}^k \pi_{jn}^k \tag{P1a}$$

subject to:

$$\sum_{v\in\mathcal{V}} y_{nv}^k = w_n^k \quad \forall n \in \mathcal{N}_k \tag{P1b}$$

$$x_{jnvg}^k \le y_{nv}^k \quad \forall j \in \mathcal{J}_k \; \forall n \in \mathcal{N}_k \; \forall v \in \mathcal{V} \; \forall g \in \mathcal{G}_v \tag{P1c}$$

$$x_{jnvg}^k \le z_{jn}^k \quad \forall j \in \mathcal{J}_k \; \forall n \in \mathcal{N}_k \; \forall v \in \mathcal{V} \; \forall g \in \mathcal{G}_v \tag{P1d}$$

$$\sum_{v\in\mathcal{V}} \sum_{g\in\mathcal{G}_v} x_{jnvg}^k \le w_n^k \quad \forall j \in \mathcal{J}_k \; \forall n \in \mathcal{N}_k \tag{P1e}$$

$$\sum_{n\in\mathcal{N}_k} \sum_{v\in\mathcal{V}} \sum_{g\in\mathcal{G}_v} x_{jnvg}^k = \sum_{n\in\mathcal{N}_k} z_{jn}^k \quad \forall j \in \mathcal{J}_k \tag{P1f}$$

$$\sum_{j\in\mathcal{J}_k} \sum_{g\in\mathcal{G}_v} g x_{jnvg}^k \le G_v \quad \forall n \in \mathcal{N}_k \; \forall v \in \mathcal{V} \tag{P1g}$$

$$\sum_{n\in\mathcal{N}_k} \sum_{v\in\mathcal{V}} \sum_{g\in\mathcal{G}_v} t_{jvg} x_{jnvg}^k \le d_j + \tau_j^k \quad \forall j \in \mathcal{J}_k \tag{P1h}$$

$$(H + M_j)(1 - \sum_{n\in\mathcal{N}_k} z_{jn}^k) \le d_j + \widehat{\tau}_j^k \quad \forall j \in \mathcal{J}_k \tag{P1i}$$

$$\sum_{v\in\mathcal{V}} \sum_{g\in\mathcal{G}_v} t_{jvg} c_v x_{jnvg}^k \le \pi_{jn}^k \quad \forall j \in \mathcal{J}_k \; \forall n \in \mathcal{N}_k \tag{P1j}$$

$$\sum_{j\in\mathcal{J}_k} \alpha_{jn}^k = w_n^k \quad \forall n \in \mathcal{N}_k \tag{P1k}$$

$$\alpha_{jn}^k \le z_{jn}^k \quad \forall j \in \mathcal{J}_k \; \forall n \in \mathcal{N}_k \tag{P1l}$$

$$\sum_{n\in\mathcal{N}_k} z_{jn}^k \le 1 \quad \forall j \in \mathcal{J}_k \tag{P1m}$$

$$\sum_{n\in\mathcal{N}_k} w_n^k = min\{|\mathcal{N}_k|, |\mathcal{J}_k|\} \tag{P1n}$$

Table I: Notation of the MILP model

Problem parameters	
c_v	time unit cost of a type v VM
d_j	due date of job j
ω_j	tardiness weight of job j
M_j	maximum execution time of job j
t_{jvg}	execution time of job j when running on type v VM with g GPUs
$\widehat{M_j}$	maximum possible execution cost for job j
H	scheduling time interval
μ	a penalty coefficient for unused GPUs
ρ	a penalty coefficient for postponed job
Local controller k variables	
w_n^k	1 if node $n \in \mathcal{N}_k$ is allocated and 0 otherwise
y_{nv}^k	1 if node $n \in \mathcal{N}_k$ is of type v and 0 otherwise
z_{jn}^k	1 if job $j \in \mathcal{J}_k$ is executed on node $n \in \mathcal{N}_k$ and 0 otherwise
x_{jnvg}^k	1 if job $j \in \mathcal{J}_k$ is executed on node $n \in \mathcal{N}_k$ of type v on g GPUs, and 0 otherwise
τ_j^k	tardiness of job $j \in \mathcal{J}_k$
$\widehat{\tau}_j^k$	worst-case tardiness of job $j \in \mathcal{J}_k$ if it is postponed
π_{jn}^k	cost of node n to execute job $j \in \mathcal{J}_k$
α_{jn}^k	1 if job $j \in \mathcal{J}_k$ is the first ending job on node $n \in \mathcal{N}_k$, and 0 otherwise

$$y_{nv}^k \in \{0,1\}, z_{jn}^k \in \{0,1\}, \alpha_{jn}^k \in \{0,1\} \quad \forall n \in \mathcal{N}_k \ \forall v \in \mathcal{V} \tag{P1o}$$

$$x_{jnvg}^k \in \{0,1\} \quad \forall j \in \mathcal{J}_k \ \forall n \in \mathcal{N}_k \ \forall v \in \mathcal{V} \ \forall g \in \mathcal{G}_v \tag{P1p}$$

$$\tau_j^k \geq 0, \widehat{\tau}_j^k \geq 0 \quad \forall j \in \mathcal{J}_k \tag{P1q}$$

$$\pi_{jn}^k \geq 0 \quad \forall j \in \mathcal{J}_k \ \forall n \in \mathcal{N}_k. \tag{P1r}$$

Constraints (P1b) enforce that, for each selected node $n \in \mathcal{N}_k$, exactly one VM type v is chosen. Constraints (P1c) ensure that only deployments on the chosen node are feasible, while Constraints (P1d) ensure that only jobs that will be executed will be deployed. Constraints (P1e) bind the allocation of jobs to selected nodes. Moreover, Constraints (P1f) enforce the association of exactly one deployment choice to every executed job. Constraints (P1g) bind the number of allocated GPUs to the available capacity of the selected node: g is the number of GPUs selected by variable x_{jnvg}^k for each job, thus their sum must not exceed the number of available GPUs (G_v) of the selected VM. Constraints (P1h) try to enforce every job due date and define the tardiness τ_j in case of violation. Constraints (P1i) define the worst-case tardiness $\widehat{\tau}_j$ for postponed jobs (characterized by $\sum_{n \in \mathcal{N}_k} z_{jn}^k = 0$). The worst-case tardiness will be equal to zero for executed jobs (with $\sum_{n \in \mathcal{N}_k} z_{jn}^k = 1$) and will be equal to $(H + M_j - d_j)$ otherwise. Constraints (P1j) compute the VM cost for executing each job. Constraints (P1k) enforce that, for each node $n \in \mathcal{N}_k$, only one job on VM n will have $\alpha_{jn}^k = 1$, while the others will have $\alpha_{jn}^k = 0$. As discussed later, because of the objective function, $\alpha_{jn}^k = 1$ if and only if job j is the first to finish on node n. In this way, constraints (P1l) link the α_{jn}^k and z_{jn}^k variables and for each node the cost of execution will be limited to consider the the first ended job on the node. Constraints (P1m) ensure that each job is allocated to at most one node. Constraints (P1n) ensure that the number of selected nodes is equal to the minimum between the number of available nodes and of available jobs. This enforces the execution of jobs as soon as resources are available. If there are idle resources, not running the jobs in the current time slot and postponing them to the next one cannot reduce their execution cost. Instead, since jobs can be preempted, postponing their execution will only make their due dates stricter, possibly requiring additional resources and imposing higher costs. Constraints (P1o)-(P1r) define the decision variables domain.

Concerning the objective function (P1a), in the first term, $\sum_{j \in \mathcal{J}_k} \omega_j \tau_j^k$ corresponds to the weighted tardiness of all running jobs, while $\sum_{j \in \mathcal{J}_k} \rho \omega_j \widehat{\tau}_j$ to the worst-case weighted tardiness of the jobs that are postponed. The $\rho > 1$ term increments the penalty of the postponed jobs forcing the system to not put off jobs that would violate their due date.

The second term of the objective function corresponds to the difference between the number of used GPUs and the number of available GPUs from each selected VM type. Since the objective function is minimized, all available resources tend to be used (in this way multi-GPU VMs do not have idle resources, μ is a positive constant acting as a Lagrange multiplier). The third term of the objective function $\sum_{j \in \mathcal{J}_k} \sum_{n \in \mathcal{N}_k} \alpha_{jn}^k \pi_{jn}^k$ corresponds to the total execution costs. For each node $n \in \mathcal{N}_k$, the execution cost is given by the time unit cost of its chosen VM type multiplied by the execution time of the first job that will complete on it. Due to the objective function minimization, only the first job to complete on each node will have $\alpha_{jn}^k = 1$ while the other jobs will be characterized by $\alpha_{jn}^k = 0$ as the fastest job has the smallest π_{jn}^k. Finally, note that we are neglecting re-configuration costs of running nodes since this would require a few minutes while DL training jobs run for several hours (or days).

The formulation (P1) is clearly non-linear because of the bilinear terms $\alpha_{jn}^k \pi_{jn}^k$ in the objective function. Since current non-linear optimization solvers that support integer variables are much less effective than MILP solvers, the model (P1) is linearized at the end of this section. The linearized version is denoted in the following as (P2).

As pointed out at the beginning of this section, (P1) focuses on identifying the best resources for the first job to complete on each node while the remaining jobs receive a best effort assignment. Since this approach might seem short sighted, we investigated also other variants. We developed a second MILP model which focuses on earliness rather than on tardiness and tries to allocate the right amount of resources to jobs so that they end as close as possible to their due dates. This second formulation includes some approximations since the execution of a full set of jobs is considered in the objective function conservatively, while the re-optimization performed after the first job ends on any node will possibly provide a different GPU assignment. We obtained a third MILP model through

a convex combination between the objective function of the second model and the one of (P2), trying to identify the right balance between a selfish assignment to jobs ending first and the cost upper bound provided by the second model earliness formulation. However, experiments demonstrated that in general such variants provide worse results than those obtained with (P2).

Finally, it is worth noticing that our previous MILP formulation in [10] was based on an approximated evaluation of (P1) costs, assuming that the time unit costs of VMs with the same type of GPUs are linear in the GPUs number. Indeed, the costs of the allocated GPUs are computed without considering that the GPUs come in lots. This is in line with the current cloud providers pricing models since the economy of scale does not provide any real benefit. Such an approximation was adopted to reduce the number of variables and constraints as well as to keep the model linear. Here, however, the scalability issues are fully overcame by the adoption of a hierarchical framework; moreover, since each local controller solves an instance with limited size, the high number of variables and constraints in the MILP formulation is less challenging than in [10]. In Section V, we will show that the hierarchical approach combined with the new MILP formulation enables us to improve our previous work [10] by not only reducing the computing time significantly but also decreasing the total costs by 3% on average..

As mentioned above, the optimization problem (P1) is non-linear because of the bi-linear terms $\alpha^k_{jn}\pi^k_{jn}$ in the objective function. To have a linear formulation, we replace them with new variables ξ^k_n that account for the minimum deployment cost across all jobs using six new constraints. Moreover, for each job, we introduce an auxiliary non-negative variable γ^k_{jn} that corresponds to the deployment cost to execute job j completely.

The linearized formulation (P2) is as follows:

$$\min \sum_{j\in\mathcal{J}_k} \omega_j \left(\tau^k_j + \rho\widehat{\tau^k_j}\right) + \mu \sum_{\substack{n\in\mathcal{N}_k \\ v\in\mathcal{V}}} (G_v y^k_{nv} - \sum_{\substack{j\in\mathcal{J}_k \\ g\in\mathcal{G}_v}} g x^k_{jnvg}) + \sum_{n\in\mathcal{N}_k} \xi^k_n, \tag{P2a}$$

subject to:

(P1b) - (P1r) and:

$$\xi^k_n \geq \sum_{j\in\mathcal{J}_k} \gamma^k_{jn} \qquad \forall n \in \mathcal{N}_k \tag{P2b}$$

$$\gamma^k_{jn} \leq \pi^k_{jn} \qquad \forall j \in \mathcal{J}_k\ \forall n \in \mathcal{N}_k \tag{P2c}$$

$$\gamma^k_{jn} \leq \widehat{M_j}\alpha^k_{jn} \qquad \forall j \in \mathcal{J}_k\ \forall n \in \mathcal{N}_k \tag{P2d}$$

$$\pi^k_{jn} - \widehat{M_j}(1-\alpha^k_{jn}) \leq \gamma^k_{jn} \qquad \forall j \in \mathcal{J}_k\ \forall n \in \mathcal{N}_k \tag{P2e}$$

$$\xi^k_n \geq 0 \qquad \forall n \in \mathcal{N}_k \tag{P2f}$$

$$\gamma^k_{jn} \geq 0 \qquad \forall j \in \mathcal{J}_k\ \forall n \in \mathcal{N}_k. \tag{P2g}$$

Given the constraints (P2b)-(P2g), it is easy to verify that if $\alpha^k_{jn} = 1$, then ξ^k_n equals the deployment cost of the first jobs $j \in \mathcal{J}_k$ that will end under the selected GPUs assignment. Indeed:

- If $\alpha^k_{jn} = 1$, then the corresponding constraint (P2c) is more restrictive than the constraints (P2d) $\gamma^k_{jn} \leq \widehat{M_j}$ since $\pi^k_{jn} \leq \widehat{M_j}$. Moreover, from the corresponding constraint (P2e) $\gamma^k_{jn} \geq \pi^k_{jn}$, we get $\gamma^k_{jn} = \pi^k_{jn}$. Finally, since we have a minimization problem also constraints (P2b) hold as equalities and γ^k_{jn} is equal to the cost due to execute job $j \in \mathcal{J}_k$ completely under the selected GPUs assignment.
- If $\alpha^k_{jn} = 0$, then constraint (P2d) entails $\gamma^k_{jn} \leq 0$. By constraint (P2g) we get $\gamma^k_{jn} = 0$. Moreover, constraint (P2e) becomes $\gamma^k_{jn} \geq \pi^k_{jn} - \widehat{M_j}$, which is a non-positive number and hence is always satisfied.

Since we have a minimization problem also constraints (P2b) hold as equalities and ξ^k_n is equal to the only γ^k_{jn} corresponding to $\alpha^k_{jn} = 1$, i.e., it equals the cost to execute completely the first job $j \in \mathcal{J}_k$ that will end under the selected GPUs assignment on node $n \in \mathcal{N}_k$.

V. Experimental results

We have evaluated the proposed approach in a large set of randomly generated scenarios as described in Section V-A. Two aspects are assessed in Section V-B: the quality of results, and the efficiency. For what concerns the quality of results, the approach proposed in this paper is compared against first-principle methods and our previous work [10]. On the other side, for what concerns the efficiency aspects, a scalability analysis is presented. The proposed approach has also been evaluated in a prototype system deployed on Microsoft Azure, achieving a deviation between the expected and real costs (including VMs and tardiness costs) below 7%. Details can be found in [10].

A. Experimental setup

As representatives of long-running DL training jobs, we selected the training of some neural networks (i.e., Alexnet, Resnet, VGG, and DeepSpeech) implemented with different deep learning frameworks (i.e., PyTorch and Tensorflow) with a significant heterogeneity in terms of resource usage. Indeed, while VGG performance is heavily related to available computational power, Alexnet and DeepSpeech performance are mainly determined by disk-access efficiency and by the GPU memory size and speed. Finally, Resnet is characterized by a balanced type of workload. For each pair network-framework, several application instances have been created varying the epochs number and the batch size.

The considered VM catalog (reported in Table II) is composed of 9 different types. Six of them (NC6, NC12, NC24, NV6, NV12, NV24) are based on Nvidia K80 and M60 and are available on Microsoft Azure. The others are based on in-house servers (Quadro P600 and GTX 1080Ti) and they have been added to increase the complexity of the problems by enlarging the set of possible candidates.

To verify the effectiveness and generality of the proposed approach, several random problem instances are generated

Table II: Characteristics of the Target Nodes

VM type	GPU type	# GPU	Cost [$/h]
NC6	K80	1	0.56
NC12	K80	2	1.13
NC24	K80	4	2.25
NV6	M60	1	0.62
NV12	M60	2	1.24
NV24	M60	4	2.48
Custom1	Quadro P600	2	0.11
Custom2	GTX 1080Ti	8	1.13
Custom3	Quadro P600	8	0.44

using the parameters described in the following. We varied the number N of available nodes in the cluster from 5 to 100. The number of submitted jobs in each instance is set to $J = 10N$. The number of controllers K has been set to $N/5$, i.e., each local controller has to manage 5 VMs. As in other literature proposals, the jobs inter-arrival times have been generated according to a Poisson distribution [1] whose mean is equal to $45,000s$. This value is smaller than the execution time of shortest jobs, so that multiple jobs are loaded in the system at each time slot. For each value of the cluster size, three problem instances are built by changing the seed of the random distribution. The remaining parameters are set as follows. The periodic scheduling time interval H is set to one hour. The due date 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. In this way, for any tardy job, the average time unit delay is almost ten times larger than the time unit execution cost. The postponed job penalty ρ is set to 100 while the μ parameter is set equal to 1 (given the objective function adopted in the problem formulation, any positive value forces the use of all available GPUs).

The results of the proposed approach have been compared against those obtained with first principle methods such as First-in-First-out (FIFO), Earliest Deadline First (EDF), Priority Scheduling (PS), and the centralized approach described in [10]. All the methods in comparison have been implemented in Python and rely on Gurobi Optimizer 8.0. The relative mixed integer programming gap (the difference between the current upper and lower bounds of the MILP solver) has been set to 5%. All the results have been collected by running the implementation on a VM running on top of a server based on Intel Xeon E5-2640 exploiting 32 cores and 32 GB of memory. The collection of all the results for all methods and all instances required more than 60 days.

B. Comparative analysis

Figure 2 compares the average computation time required to solve the single instances in the centralized approach (which solves the MILP model presented in [10]) and the one proposed in this paper. The main difference

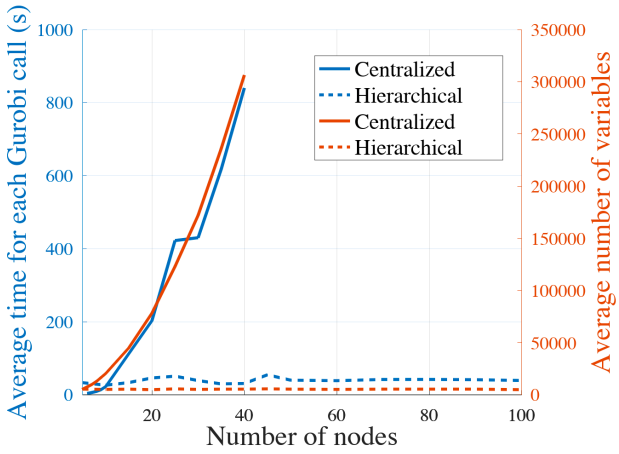

Figure 2: Average computation time and number of variables of the centralized and the hierarchical MILP models

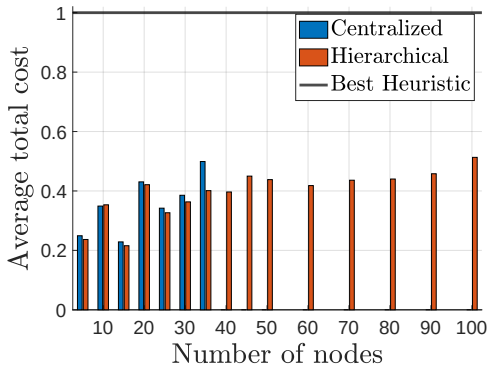

Figure 3: Fraction of the proposed approach and centralized MILP model cost w.r.t the best heuristic

is attributable to the different number of variables in the two MILP models, which are also reported in Figure 2.

In the centralized model, instances with 40 nodes lead to MILP models with more than 300,000 variables, leading Gurobi to memory failures. However, in the hierarchical framework, each local controller tackles an instance whose size is fixed, thus considering a MILP model involving 5,000 variables on average, with a required computation time of approximately $40s$. Small variations of the k subproblem computation time are due to the varying number of jobs in the local controller queues, but this did not introduce any significant increment in the computation time, making the approach scalable for very large problem instances including hundreds of jobs to be scheduled.

It is worth noticing that the new MILP formulation presented in this paper includes a larger number of variables and constraints than the one presented in [10], so that it can be exploited only in a hierarchical framework, where the dimension of the instances is fixed, while it would be infeasible in a centralized framework.

The results of the analyzed approaches are compared in Figure 3. The results of different experiments with the same number of nodes have been averaged and normalized with respect to the best heuristic. Despite the distributed

hierarchical approach, the proposed method not only does not produce worse results than the centralized one, but it slightly improves centralized solutions by 3% on average.

Finally, to determine the minimum number of servers needed to support the optimization runs by the K controllers without introducing any delay to computation time, we modeled the optimization server running Gurobi as a $M/G/1$ queue [14] and considered the largest simulation setting with 100 nodes and 1,000 jobs. As depicted in Figure 2, the average computation time of the k-th controller is about $40s$. For what concerns the frequency of execution of the optimization process, we observed that our simulations can be split into three parts: in the first part only job submissions happen in the system, in the second part job submissions are interleaved with job completions and in the third part every job has been already submitted and we observe only job completions. However, in any step completion rate is much lower than the submission rate (DL jobs are long-running), so as a first conservative approximation, to determine the number of optimization servers required, we considered only the second part of the job submission trace. On average, the rate of the optimization is equal to the rate of submission plus the rate of ending jobs: this sum can be approximated to two times the job submission rate (characterized by an average inter-arrival time of $450s$). Results have shown that a single server allows computing the optimal solution requested by all local controllers in about $48.65s$. Hence, a single optimization server is enough to handle a system with 100 nodes and 20 controllers.

VI. Concluding remarks

In this paper, we propose a hierarchical framework and a novel MILP formulation for the online joint capacity planning of on-demand VMs and DL training jobs scheduling in cloud deployments. The effectiveness of our approach has been assessed by performing an extensive simulation campaign. Results show how our approach achieves savings in the 45-80% range with respect to first principle scheduling methods and the framework can support systems including up to 100 nodes in less than one minute on average. Future work will investigate the impact of other policies besides the round-robin and will identify a criterion to right-size the number of local controllers given the incoming jobs load.

References

[1] Amaral, M., Polo, J., Carrera, D., Seelam, S., Steinder, M.: Topology-aware gpu scheduling for learning workloads in cloud environments. In: HPCNSA Proc. ACM (2017)

[2] Bahrampour, S., Ramakrishnan, N., Schott, L., Shah, M.: Comparative study of deep learning software frameworks. arXiv (2015)

[3] Cai, Z., Li, X., Ruiz, R., Li, Q.: A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. FGCS **71**, 57–72 (2017)

[4] Cao, M., Jia, W., Li, S., Li, Y., Zheng, L., Liu, X.: Gpu-accelerated feature tracking for 3d reconstruction. OLT **110**, 165–175 (2019)

[5] Cheng, J.R., Gen, M.: Accelerating genetic algorithms with gpu computing: A selective overview. CIE **128**, 514–525 (2019)

[6] Gianniti, E., Zhang, L., Ardagna, D.: Performance prediction of gpu-based deep learning applications. In: SBAC-PAD (2018)

[7] GlobalMarketInsights: Gpu as a service market size by product. [Online]. Available: https://www.gminsights.com/industry-analysis/gpu-as-a-service-market (visited on 03/07/2020)

[8] Hong, C.H., Spence, I., Nikolopoulos, D.S.: Gpu virtualization and scheduling methods: a comprehensive survey. ACM CSUR **50**(3), 35 (2017)

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

[10] Jahani, A., Lattuada, M., Ciavotta, M., Ardagna, D., Amaldi, E., Zhang, L.: Optimizing on-demand gpus in the cloud for deep learning applications training. In: ICCCS (2019)

[11] Kang, Y., Joo, W., Lee, S., Shin, D.: Priority-driven spatial resource sharing scheduling for embedded graphics processing units. JSA **76**, 17–27 (2017)

[12] Kato, S., Lakshmanan, K., Rajkumar, R., Ishikawa, Y.: Timegraph: Gpu scheduling for real-time multi-tasking environments. In: USENIX Proc. (2011)

[13] Kayiran, O., Nachiappan, N.C., Jog, A., Ausavarungnirun, R., Kandemir, M.T., Loh, G.H., Mutlu, O., Das, C.R.: Managing gpu concurrency in heterogeneous architectures. In: MICRO (2014)

[14] Kleinrock, L.: Queueing Systems, vol. I: Theory. Wiley Interscience (1975)

[15] Madougou, S., Varbanescu, A., de Laat, C., van Nieuwpoort, R.: The landscape of GPGPU performance modeling tools. PC **56**, 18–33 (2016)

[16] NVIDIA: Nvidia tesla gpu servers (gpx). [Online]. Available: https://www.thinkmate.com/systems/servers/gpx (visited on 03/07/2020)

[17] NVIDIA: Nvidia virtual gpu technology. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/technologies/virtual-gpu/ (visited on 03/07/2020)

[18] Oprescu, A.M., Kielmann, T.: Bag-of-tasks scheduling under budget constraints. In: CCTS (2010)

[19] Peng, Y., Bao, Y., Chen, Y., Wu, C., Guo, C.: Optimus: An efficient dynamic resource scheduler for deep learning clusters. In: EuroSys (2018)

[20] Reaño, C., Silla, F., Castelló, A., Peña, A.J., Mayo, R., Quintana-Ortí, E.S., Duato, J.: Improving the user experience of the rcuda remote gpu virtualization framework. CCPE **27**(14), 3746–3770 (2015)

[21] Reano, C., Silla, F., Nikolopoulos, D.S., Varghese, B.: Intra-node memory safe gpu co-scheduling. IEEE TPDS **29**(5), 1089–1102 (2017)

[22] Silla, F.: rcuda selected as one of the top 5 cuda. [Online]. Available: http://www.rcuda.net/index.php/130-rcuda-top5-blogs (visited on 03/07/2020)

[23] Steinberger, M.: On dynamic scheduling for the gpu and its applications in computer graphics and beyond. IEEE CGA **38**(3), 119–130 (2018)

[24] Tan, H., Tan, Y., He, X., Li, K., Li, K.: A virtual multi-channel gpu fair scheduling method for virtual machines. IEEE TPDS **30**(2), 257–270 (2019)

[25] Wang, W., Song, W., Chen, C., Zhang, Z., Xin, Y.: I-vector features and deep neural network modeling for language recognition. CS Proc. **147**, 36–43 (2019)

[26] Wu, J., Hong, B.: Collocating cpu-only jobs with gpu-assisted jobs on gpu-assisted hpc. In: CCGrid (2013)

[27] Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., et al.: Gandiva: Introspective cluster scheduling for deep learning. In: USENIX OSDI (2018)

[28] Zhong, L., Hu, L., Zhou, H.: Deep learning based multi-temporal crop classification. RSE **221**, 430–443 (2019)

[29] Zhu, J., Li, X., Ruiz, R., Xu, X.: Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources. IEEE TPDS **29**(6), 1401–1415 (2018)