This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

A Stochastic Approach for Scheduling AI Training Jobs in GPU-based Systems

Federica Filippini, Jonatha Anselmi, Danilo Ardagna, Bruno Gaujal

***Abstract*—In this work, we optimize the scheduling of Deep Learning (DL) training jobs from the perspective of a Cloud Service Provider running a data center, which efficiently selects resources for the execution of each job to minimize the average energy consumption while satisfying time constraints. To model the problem, we first develop a Mixed-Integer Non-Linear Programming formulation. Unfortunately, the computation of an optimal solution is prohibitively expensive, and to overcome this difficulty, we design a heuristic STochastic Scheduler (STS). Exploiting the probability distribution of early termination, STS determines how to adapt the resource assignment during the execution of the jobs to minimize the expected energy cost while meeting the job due dates. The results of an extensive experimental evaluation show that STS guarantees significantly better results than other methods in the literature, effectively avoiding due date violations and yielding a percentage total cost reduction between 32% and 80% on average. We also prove the applicability of our method in real-world scenarios, as obtaining optimal schedules for systems of up to 100 nodes and 400 concurrent jobs requires less than 5 seconds. Finally, we evaluated the effectiveness of GPU sharing, i.e., running multiple jobs in a single GPU. The obtained results demonstrate that depending on the workload and GPU memory, this further reduces the energy cost by 17-29% on average.**

***Index Terms*—Deep Learning, GPU cluster, Scheduling, Average energy consumption minimization, GPU sharing, Job Tardiness.**

I. Introduction

Nowadays, different classes of problems are tackled with Deep Learning (DL) algorithms. The adoption of GPUs as accelerators highly benefits the training process, providing an execution speedup of about 5-40x with respect to CPUs [1], and thus extending the set of applications that can be tackled in reasonable computing time. However, high-performance GPU-based servers are cost-prohibitive (about 200k USD for high-end systems like NVIDIA DGX A100 [2]). As a consequence, the demand for GPU-accelerated cloud servers increased dramatically in recent years, dictating the necessity for Cloud Service Providers (CSPs) to exploit effective resource management strategies.

This work aims to optimize the resource selection and scheduling of DL training jobs from the perspective of a CSP running a data center, efficiently selecting resources for the execution of each job to minimize the energy consumption costs while meeting the applications' due dates. To the best of our knowledge, methods available in the literature tackle this problem by proposing (i) simple job scheduling mechanisms, such as Earliest-Deadline-First (EDF) or First-in-First-Out (FIFO) [3]–[6], possibly coupled with effective resource selection algorithms, or (ii) more elaborate heuristics [7]–[9], sometimes even coupling the resource selection and scheduling problem as in our previous works [10], [11]. Albeit achieving good-quality solutions, these approaches only consider the worst-case execution times of DL applications in searching for an optimal schedule. However, DL training jobs usually exploit termination criteria based not only on a predetermined number of epochs but also on reaching desired levels of accuracy. The framework we propose in this work exploits the stochastic information about the expected training times, designing a solution that does not optimize only the worst-case execution, but considers the probability of terminating the training after a smaller number of epochs.

Our reference scenario includes the CSP cluster, composed of a set of nodes characterized by different types and numbers of GPUs, and a list of jobs, each associated with a due date and a priority for fulfilling it. Jobs are submitted continuously over time, and there is no information about future arrivals. Therefore, our framework aims to minimize the energy consumption costs and the penalties associated to due date violations in an online setting, determining a new optimal schedule every time a new job is submitted or terminates its execution. Depending on the cluster configuration and the characteristics of the jobs (in terms of due dates, priorities, memory requirements and expected training times, estimated through suitable Machine Learning models), our framework selects the best GPU type to execute each application and determines the amount of resources it needs. Jobs can be executed alone on a dedicated node (exploiting one or more GPUs) or concurrently on the same node, sharing the resources with other jobs. While resource partitioning inevitably introduces some overheads, these can usually be neglected when jobs run on dedicated GPUs in the same machine (see, e.g., [11]–[15]). The interference induced by GPU sharing is more significant, nevertheless, the GPUs available on the market are increasingly performing and have a large amount of memory, which enables multiple mini-jobs to be trained simultaneously, optimizing the energy consumption and resource usage albeit slightly increasing the execution times [16].

In this work, we propose a novel Mixed-Integer Non-Linear Programming formulation (MINLP) to model the Resource Selection and Job Scheduling (RS-JS) problem. Due to the stochasticity of the parameters and the combinatorial nature of job scheduling, this problem is NP-hard and it is too

F. Filippini and D. Ardagna are with Politecnico di Milano. Milan, Italy
Email: {name.lastname}@polimi.it
J. Anselmi and B. Gaujal are with Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France.
Email: {name.lastname}@inria.fr

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

demanding to be solved directly. Therefore, we develop a heuristic method, called STochastic Scheduler (STS). Based on a stochastic model considering a single DL job, initially proposed in [17], STS determines a dynamic schedule by minimizing the expected value of energy costs while meeting the applications' due dates. STS optimizes the costs and resource usage by starting with low-power configurations and progressively increasing the assigned resources to recover potential delays. An extensive experimental evaluation based on simulation proves the effectiveness of this approach, guaranteeing an average total cost reduction between 32% and 80% against the EDF and Randomized Greedy (RG) methods proposed in [11] and a Dynamic Programming-based method adapted from the literature [4], while completely avoiding due dates violations in the analyzed scenarios. Furthermore, we show that exploiting GPU sharing yields an additional reduction between 17 and 29%. Finally, a scalability analysis demonstrates how systems with up to 100 nodes and 400 concurrent jobs are tackled in less than 5 seconds, assessing the applicability of STS to practical scenarios.

Albeit related to the work in [11], this paper cannot be considered as an extension. The RS-JS problem we tackled in [11] is similar, but we addressed a different context, while the MINLP formulation and heuristic algorithm we propose here are novel. The main contributions of this paper are:

- We tackle the RS-JS problem in a physical system, focusing on energy cost minimization, while the work in [11] considered a virtualized environment.
- We propose a novel MINLP formulation to consider GPU sharing, which was not introduced in [11].
- We consider stochastic models for the jobs training times, while the previous work assumed deterministic estimates based on executing a fixed number of epochs (a less realistic assumption in real-life scenarios).
- We develop an original heuristic method, STS, which incorporates GPU sharing and leverages the solution model proposed in [17] to tackle the inherent stochasticity of the optimal schedule.
- We design a simulator to test the quality of solutions determined by STS, capable of dealing with training times stochasticity.

Furthermore, this work extends [17], which was focused on determining the optimal schedule for a single job in a context where resource requirements can always be satisfied, using it as a starting point to determine good-quality heuristic solutions for the whole RS-JS problem in a more realistic setting.

The rest of the paper is organized as follows: a brief overview of other literature proposals relevant to our field is reported in Section II. Section III describes the reference framework of our work, introducing the parameters used to characterize the DL training jobs and the system architecture. The mathematical formulation we propose to tackle our RS-JS problem is reported in Section IV, while the single-job stochastic model considered as starting point for STS is described in Section V. Section VI describes the STS method we developed in this work, and Section VII outlines the experimental evaluation and the results we obtained with our algorithms. Finally, conclusions are drawn in Section VIII.

II. Related Work

The effective management of GPU-accelerated clusters, which is crucial to exploit at best the available resources, introduces many challenges. In this section, we provide an overview of some relevant literature proposals in this field, addressing resource selection and job scheduling problems from different perspectives. Among these, note that GPU sharing is currently gaining remarkable interest, due to the large capacity of the most recent hardware. In particular, a *time sharing* [18] or *space sharing* [19] pattern can be considered, related to the fact that jobs either exploit the entire GPU for a limited time fraction or spatially share the GPU resources with a limited percentage. While we focus on space sharing in our work, both strategies are considered in this section.

Hydra [9] is one of the literature proposals closest to our work. The authors consider heterogeneous GPU clusters and their solution adopts a quantitative cost-comparison approach where the cost is defined as the sum between the job completion times and a dynamic penalty calculated from the total tardiness. The optimal solution is determined via an optimized branch&bound algorithm. However, Hydra considers jobs executed on a single GPU, running jobs are never re-evaluated, a time-sharing pattern is considered, and the scheduling is performed in batches. Similarly, SchedTune [5] and DASH [8] are some of the few literature proposals considering heterogeneous GPU resources. SchedTune exploits Random Forest-based models to predict the job memory demand and completion times when exploiting GPU sharing. DL applications are executed in FIFO order, and deployed on the nodes that guarantee lower completion times while satisfying the memory demand and a co-location threshold to limit the interference issues. DASH is a GPU cluster scheduler designed to match different DL workloads and GPU types in multi-generational GPU environments. It promotes jobs that are likely to observe large performance improvements, using a checkpoint-restart mechanism. The system is re-optimized every time a new job arrives, establishing a protection period to avoid too frequent promotions/demotions. Note that, while the scheduler switches between generations, the GPU type and the number of GPUs assigned to each job are fixed by the user.

Differently, Pollux [7] models how the *goodput* (a custom metric encompassing throughput and training efficiency) changes by adding or removing resources in homogeneous GPU-based systems. It profiles each job and dynamically tunes the batch size, learning rate and number of assigned GPUs. It is relevant to note that, as in our work, the number of GPUs to be assigned to each job is automatically determined but the impact of different GPU types is not considered.

HiveD [20] is focused on guaranteeing the affinity requirements of jobs sharing resources in a GPU-accelerated cluster: instead of reserving a GPU quota to a job, corresponding to a given number of GPUs, users receive cells including affinity requirements. Since, a job receiving, e.g., 8 nodes with 8 GPUs experiences better performance than receiving 64 nodes with one GPU each, HiveD introduces a hierarchical reservation

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

mechanism based on cells where a multi-level structure is considered to capture the different levels of affinity that a group of GPUs could satisfy. The proposed cell structure naturally forms a hierarchy similar to a typical GPU cluster where GPU assignment follows the underlying system performance (e.g., a job is assigned first to multiple GPUs attached to a PCIe switch, then GPUs connected to a CPU socket, then additional GPUs in a rack, and so on). HiveD relies on local GPU schedulers but the proposed reservation mechanism guarantees more efficient placements. With similar arguments, ASTRAEA [21] deploys jobs on as few nodes as possible with the aim of guaranteeing long-term GPU-time fairness both at the job and at the user level. The fact that deploying jobs on a few nodes improves their performance, together with the choice of considering relatively small applications requiring at most 8 GPUs, strengthened our decision to consider single-node-only placements.

Scrooge [19] proposes a Mixed-Integer Linear Programming formulation to find the minimum-cost GPU-accelerated Virtual Machine in the cloud, meeting performance objectives. Similarly to our work, the problem is solved whenever a new job leaves or joins. However, Scrooge tackles inference DL workloads. Igniter [22], focused also on inference jobs, is an interference-aware resource provisioning framework that allows spatial GPU sharing and jointly optimizes resource allocation and scheduling. The authors model the performance interference analytically, and Igniter determines the optimal batch size, a lower bound for GPU resources and a greedy placement minimizing the interference. Note that, differently from our proposal, the optimization is executed periodically and the resource assignment is static. Similarly, Nexus [18] tackles the problem of distributing a Deep Neural Network-based video analysis tasks workload on a GPU-accelerated cluster, leveraging bin packing techniques and spatial GPU sharing to achieve high throughput under latency constraints.

PickyMan [23] and Lucid [24] address the issues related to preemption overheads. Preemption is allowed by PickyMan, which minimizes it by predicting the execution times using network traffic and historical data, and by greedily choosing the appropriate job to stop, while it is not allowed by Lucid. This is based on interpretable models and includes a job profiler, an indolent packing strategy, and a resource orchestration mechanism based on jobs priority.

MISO [6] and DISC [25] consider GPU space and time sharing. MISO exploits the Multi-Instance GPU capability of the latest NVIDIA datacenter GPUs (e.g., A100). It monitors a FIFO queue and schedules jobs on the least-used GPU to minimize disruptions and help with load balancing. As in our work, users can specify the memory requirement of each job, but migrations are not allowed to reduce the problem size and complexity. DISC is an adaptive and efficient heuristic algorithm for hyperparameter tuning aiming to improve GPU utilization and model accuracy. GPU time sharing is modeled as a fair multi-armed bandit problem using Nash social welfare, while space sharing follows memory usage patterns.

Our previous work in [11] jointly optimizes the resource selection and scheduling of DL jobs in virtualized clusters, finding low-cost solutions by applying several heuristics based on Randomized Greedy and Path Relinking. In this paper, we rely on the same performance models to estimate job execution times for a specific GPU assignment. The analysis in [11] of the performance achieved on real clusters demonstrated that the difference between the total costs (based on jobs performance) obtained by our models and the measured costs on the real system is below 13% (significantly lower than the savings, in 32-80% range, we achieved with respect to other methods as it will be demonstrated in Section VII). Finally, the paper in [17], which serves as the foundation for the development of our stochastic scheduler, is included in Section V for comprehensive coverage. This paper presents an analytical model for the ideal scheduling of individual Deep Learning jobs within a system based on GPUs.

III. Problem Description

The aim of this work is to tackle the Resource Selection and Job Scheduling (RS-JS) problem for DL training jobs from the perspective of a CSP running a data center. The CSP goal is to minimize the energy consumption costs and the penalties for due date violations. It is worth noting that the principles underlying our solution are also applicable to scenarios involving batch inference jobs when the processing times are considerable. However, mixed-workloads including training and inference jobs are left as part of our future work.

The scenario we model includes two main components: the cluster, identified by a set of nodes $\mathcal{N}$ characterized by their type and number of GPUs, and the queue of submitted jobs $\mathcal{J}$, each associated with a due date d_j and a priority for fulfilling it. Each job is trained for a maximum number of epochs denoted by $w_j^{\max}$ and suitable Machine Learning models can be used to estimate the corresponding execution time on all the candidate resources [26]. However, to characterize termination criteria based on desired levels of accuracy, we introduce the probability of concluding the training after $W_j \leq w_j^{\max}$ epochs. This corresponds to considering stochastic execution profiles since the training time of job j depends on the number of epochs it executes, which is defined as a random variable.

The following sections detail the reference framework we tackle in our work, as well as the GPUs power consumption model we consider to guide the optimal resource selection.

A. Reference Framework

The RS-JS process is exemplified in Figure 1. In each scheduling interval, the optimizer combines the information about the submitted jobs and the system description to (i) determine the optimal configuration (type and number of GPUs) for each job, and (ii) appropriately partition the set of available computational resources. Jobs can run on a single dedicated node or share the resources (the node or even the GPU) with other jobs, provided that they are guaranteed at least the minimum amount of memory m_j they need to run. If the available resources do not allow to execute all jobs concurrently, some jobs can be postponed to the following scheduling interval. Note that, while the interference generated by node partitioning can be neglected (see, e.g., [11]–[15]), this is not generally true for GPU sharing. Nevertheless, we

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

decided to consider this scenario, which is currently gaining popularity in the literature [19], [22], since the large memory of the most recent GPUs usually allows to simultaneously execute multiple mini-jobs, freeing resources for more demanding jobs. Moreover, GPU sharing is essential for effectively supporting the training of DL models for edge computing systems, where the available resources are limited. The DL models considered in this technology setting are usually small so that multiple training jobs can fit in memory when adopting high-end GPUs. To consider the performance degradation, we define the possible GPU fractions $f \in \mathcal{F}$ that jobs may be assigned to and introduce corresponding time-inflation coefficients b_f.

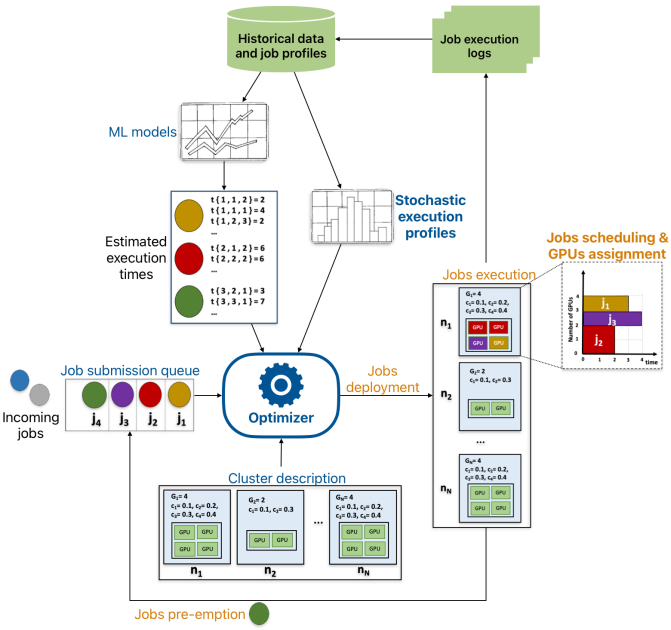


Fig. 1: Reference framework. In the example, we consider four submitted jobs j_1, j_2, j_3, j_4. Nodes n_1 to n_N are equipped with a variable number of GPUs of possibly different types and have different energy costs. Jobs j_1, j_2 and j_3 are deployed on n_1, running on 1 GPU and 2 GPUs, respectively. Job j_4 is sent back to the queue, and no other nodes are selected.

The RS-JS problem is tackled in an online scenario: no information is available in advance about new job submissions and their characteristics. Whenever a job enters the system, the schedule is re-evaluated, and the resources assigned to the running applications may change to accommodate the new one, favoring high-priority jobs to minimize penalty violations. As in other literature proposals [13], [27], [28], jobs can be preempted and/or migrated to nodes with a different type or number of GPUs. Note that the online setting makes it impossible to find a schedule that optimizes the training of all jobs over a long time horizon including multiple submissions. Therefore, as in zero-data problems [29], we are constrained to find a *local* minimum in each scheduling interval, reconfiguring the system whenever (i) a job is submitted or (ii) terminates the training, or (iii) after a fixed time H. Focusing on a single interval may adversely affect the overall schedule, thus we need suitable mechanisms to guarantee that the solution is robust against future, unpredictable events that may reduce the amount of available resources.

At every rescheduling point, the queue $\mathcal{J}$ includes both jobs that have just been submitted and jobs that were previously postponed. Moreover, since all assignments are reevaluated, the scheduler virtually adds to $\mathcal{J}$ the already running jobs[1]. The priority of each job j is characterized by the tardiness weight η_j, used to compute the penalty for due date violations. Each cluster node $n \in \mathcal{N}$ is instead characterized by a specific number of GPUs, denoted by K_n, of a fixed type $i \in \mathcal{I}$ (we define a coefficient G_{ni} equal to 1 if node n hosts GPUs of type i), and by an hourly cost c_{nk} related to the number of used GPUs k. Moreover, we denote by M_n the memory of each GPU on node n (note that, for the sake of simplicity, we consider the GPUs on a node to be homogeneous). Finally, we introduce a set $\mathcal{G}$ to enumerate all the GPUs in the cluster (i.e., $|\mathcal{G}| = \sum_{n \in \mathcal{N}} K_n$), and we introduce a binary parameter Γ_{gn} that is 1 if GPU $g \in \mathcal{G}$ is located on node n. As discussed in the next section, this will be relevant to properly define the placement problem when multiple jobs share a single GPU.

Execution logs of the running jobs are used to periodically re-train the ML models used to estimate execution time and to update the statistics on the number of training epochs.

B. Power-Consumption Model

Discussing how the power consumption of the GPUs assigned to a given job j is related to the job processing speed is crucial to identify the optimal solution to our problem.

Consider a job j, executed on k GPUs of type i on any node n. Since the job, the GPU type and the node are fixed, we will omit here the corresponding indices, denoting by $K = K_n$ the maximum possible value of k.

Let s_k be the processing speed of job j when k GPUs are in use. The function s_k is the speed-up function of the job and measures its degree of parallelism. We can assume s_k to be sublinear in the number of GPUs k (see, e.g., [26]). More precisely, we assume that $(s_k)_k$ is non-decreasing and that $s_{k+1} - s_k$ is non-increasing. These "diminishing returns" assumption is natural to model the overheads induced by parallel computations [26], [30]. Moreover, $s_0 = 0$.

We assume, as in [10], [31], that the (instantaneous) power consumption when operating at speed s_k takes the form:

$$P(s_k) := \begin{cases} kP_{\text{on}} + (K-k)P_{\text{idle}} & \text{if } k > 0 \\ 0 & \text{if } k = 0, \end{cases} \tag{1}$$

where P_{on} and P_{idle} are the (instantaneous) power consumption of an in-use and idle GPU, respectively. Note that $P(0) = 0$, i.e., the assumption that the system is at complete rest when no GPU is active, holds because we can prescribe that nodes are switched off when no GPUs are used, to avoid resource wastage. In the right-hand-side of (1), k is the inverse function of s_k. Since we have put mild assumptions on the sequence $(s_k)_k$, the power consumption function in (1), seen as a function of the processing speed, is in fact quite general. We assume that $P(s)$ is non-decreasing and convex.

[1]Note that they are actually stopped and migrated only at the end of the scheduling process if their resource assignment needs to be modified

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

TABLE I: RS-JS Problem Parameters and Variables

Parameters	
$\mathcal{J}$	set of submitted jobs
$\mathcal{N}$	set of cluster nodes
$\mathcal{G}$	set of all the cluster GPUs
$\mathcal{I}$	set of GPU types
$\mathcal{F}$	set of GPU fractions
K_n	total number of GPUs on node n
M_n	memory of a GPU on node n
G_{ni}	1 if node n hosts GPUs of type i, 0 otherwise
Γ_{gn}	1 if GPU g is located on node n, 0 otherwise
m_j	minimum amount of memory needed to execute job j
$w_j^{\max}$	maximum number of training epochs for job j
W_j	number of epochs executed by job j
t_{jik}	training time of job j on k GPUs of type i
b_f	time-inflation coefficient when a GPU fraction f is used
H	maximum scheduling time interval
$\bar{T}_j$	maximum execution time of job j
d_j	due date of job j
η_j	tardiness weight of job j
c_{nk}	time-unit energy cost of node n when k GPUs are used
ρ	penalty coefficient for postponed jobs
Variables	
ν_n	1 if node n is chosen, 0 otherwise
ξ_{jink}	1 if job j is executed on node n with k GPUs of type i, 0 otherwise
σ_{jg}	1 if GPU g is assigned to job j, 0 otherwise
φ_{jf}	1 if job j is executed on a GPU fraction f, 0 otherwise
ζ_{jn}	1 if job j is executed on node n, 0 otherwise
γ_{nk}	1 if k is the total number of used GPUs on node n, 0 otherwise
τ_j	tardiness of job j
$\bar{\tau}_j$	worst-case tardiness of job j
π_{jn}	energy-consumption cost of job j on node n
α_{jn}	1 if j is the first-ending job on node n, 0 otherwise

IV. Resource Selection–Job Scheduling Problem

This section proposes the mathematical formulations we developed to describe the RS-JS problem introduced in Section III. The goal is to minimize the expected energy consumption costs and penalties for due date violations, over a long time horizon including multiple job submissions and executions. As described in Section III, all the assignment decisions are reevaluated at each rescheduling point, occurring after (i) a new job arrival, (ii) a job completion, or (iii) a maximum time interval H. Since the problem is tackled in an online setting, future job submissions are unpredictable, and therefore we consider only events (ii) and (iii) in our MINLP formulation. Moreover, here we focus on determining the optimal solution for the *local* problem considering a single scheduling interval, while the complete horizon is tackled by the model described in Section V and the heuristic method presented in Section VI.

The scheduling decisions addressed in the RS-JS problem are characterized through a list of parameters and variables that are reported for clarity in Table I. The main binary variables introduced to represent the schedule are ν_n, which are 1 if node $n \in \mathcal{N}$ is used, ζ_{jn}, which are 1 if job $j \in \mathcal{J}$ runs on node n, ξ_{jink}, which are 1 if job j is executed on node n with k GPUs of type $i \in \mathcal{I}$, σ_{jg}, which are 1 if job j is deployed on a GPU $g \in \mathcal{G}$, and φ_{jf}, which are 1 if job j runs on a GPU fraction $f \in \mathcal{F}$. Note that the set of fractions is defined as $\mathcal{F} = \{1/2, 1/4, \cdots, 1\}$, where $f = 1$ is assigned when a job runs on a dedicated GPU (with, of course, no time inflation: $b_1 = 1$). Finally, we consider binary variables α_{jn}, which are 1 if j is the first job that terminates on node n, and γ_{nk}, which are 1 if k is the total number of used GPUs on node n. We introduce eight families of placement constraints, which are defined in the following. The first group (Constraints (2)-(5)) imposes that a job can be deployed on a specific configuration (node n, GPU type i and number of GPUs k) only if that configuration is selected in the schedule:

$$\xi_{jink} \leq \zeta_{jn} \qquad \forall j \in \mathcal{J},\ i \in \mathcal{I},\ n \in \mathcal{N},\ k = 1 : K_n \tag{2}$$

$$\sum_{i \in \mathcal{I}} \sum_{k=1}^{K_n} \xi_{jink} \leq \nu_n \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N} \tag{3}$$

$$\sum_{n \in \mathcal{N}} \zeta_{jn} \leq 1 \qquad \forall j \in \mathcal{J} \tag{4}$$

$$\zeta_{jn} \leq \nu_n \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N}. \tag{5}$$

Specifically, inequalities (2) specify that each job j can run on node n using k GPUs of type i only if it is assigned to node n in the current time period, i.e., $\zeta_{jn} = 1$. The sum at the left-hand-side of Constraints (3) corresponds to the number of pairs (i, k) selected for each job j on node n. This must be 0 when node n is not in use ($\nu_n = 0$). When instead $\nu_n = 1$, at most one of the variables ξ_{jink} can be equal to 1 (if the job is actually deployed on the node, as specified by the previous constraints). Finally, by Constraints (4) and (5), each job must be assigned to at most one node, and only if the node is in use. Note that, while deciding to deploy jobs on single nodes may seem a limitation, recent DL workloads analysis [32] highlighted how over 50% of jobs require a single GPU, while jobs exploiting more than 8 GPUs (which we will consider as the maximum node size in our experimental evaluation) are less than 10%. Moreover, enforcing GPU locality yields over $10\times$ speed-up [16].

The second family (Constraints (6) and (7)) is related to the GPU type selected for job j:

$$\xi_{jink} \leq G_{ni} \qquad \forall j \in \mathcal{J},\ i \in \mathcal{I},\ n \in \mathcal{N},\ k = 1 : K_n \tag{6}$$

$$\sum_{k=1}^{K_n} \xi_{jink} = \zeta_{jn} G_{ni} \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N},\ i \in \mathcal{I}. \tag{7}$$

In particular, the former prescribes that a job can be assigned to node n with GPU type i only if this is the type hosted by the node ($G_{ni} = 1$). The right-hand side of equalities (7) is 1 if job j is deployed on node n and this hosts GPUs of type i. If this is the case, the constraints impose that a specific number of GPUs is assigned to the job.

The third group (Constraints (8)-(10)) predicates on the GPU fraction f assigned to job j:

$$\sum_{f \in \mathcal{F}} \varphi_{jf} \leq 1 \qquad \forall j \in \mathcal{J} \tag{8}$$

$$\sum_{\substack{j \in \mathcal{J} \\ f \in \mathcal{F}}} f \varphi_{jf} \sigma_{jg} \leq 1 \qquad \forall g \in \mathcal{G} \tag{9}$$

$$\sum_{\substack{i \in \mathcal{I} \\ n \in \mathcal{N}}} \sum_{k=2}^{K_n} \xi_{jink} \leq \varphi_{j1} \qquad \forall j \in \mathcal{J}. \tag{10}$$

Specifically, at most one fraction in $\mathcal{F}$ can be assigned to each job (see (8)), the sum of the fractions assigned to all jobs in a specific GPU g must not exceed 1 (see (9)), and the fraction $f = 1$ must be assigned to all jobs that are executed on more than one GPU (identified by the sum at left-hand side of Constraints (10)).

The fourth family (Constraints (11) and (12)) predicates on the GPU(s) assigned to each job:

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

$$\sum_{g\in\mathcal{G}} \sigma_{jg} = \sum_{\substack{i\in\mathcal{I}\\ n\in\mathcal{N}}} \sum_{k=1}^{K_n} k\xi_{jink} \qquad \forall j \in \mathcal{J} \tag{11}$$

$$\sigma_{jg} \leq \zeta_{jn}\Gamma_{gn} \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N},\ g \in \mathcal{G}. \tag{12}$$

By summing over all GPUs $g \in \mathcal{G}$, Constraints (11) compute the number of GPUs assigned to job j; this must be equal to the number k identified by variables ξ_{jink}. Moreover, by Constraints (12), a specific GPU g can be assigned to j only if it is on the same node n where the job is deployed.

The fifth group (Constraints (13) and (14)) is related to the number of active GPU in each node:

$$\sum_{\substack{j\in\mathcal{J}\\ g\in\mathcal{G}}} \sigma_{jg}\Gamma_{gn} = k\gamma_{nk} \qquad \forall n \in \mathcal{N},\ k = 1 : K_n \tag{13}$$

$$\sum_{k=1}^{K_n} \gamma_{nk} = \nu_n \qquad \forall n \in \mathcal{N}. \tag{14}$$

Constraints (13) prescribe that, for each node n, the variable γ_{nk} is equal to 1 if k is the total number of used GPUs on the node, given by the sum at the left-hand side. Moreover, Constraints (14) impose that, if the node is selected ($\nu_n = 1$), exactly one γ_{nk} is equal to 1 (the number of used GPUs on the node must be unique), while all the γ_{nk} variables are zero if the node is not used.

Constraints (15) check the memory requirement of each job j, prescribing that it cannot exceed the fraction of available GPU memory on the selected node:

$$\sum_{\substack{g\in\mathcal{G}\\ f\in\mathcal{F}}} (M_n f - m_j)\,\Gamma_{gn}\varphi_{jf}\sigma_{jg} \geq 0 \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N}. \tag{15}$$

Constraint (16) concerns the total number of active nodes:

$$\sum_{n\in\mathcal{N}} \nu_n = \min\{|\mathcal{J}|, |\mathcal{N}|\}. \tag{16}$$

In particular, it requires that the number of used nodes is equal to the minimum between the queue length $|\mathcal{J}|$ and the total number of nodes $|\mathcal{N}|$, guaranteeing that as many submitted jobs as possible are executed. Running the jobs as soon as resources are available contributes to minimizing the costs, because postponed jobs approach their due date and thus require more (and possibly more expensive) resources to be completed without delays. Moreover, preemption guarantees that, if higher-priority jobs are submitted in the future, the resources can be reallocated, thus the current decisions do not undermine the effectiveness of the solution in the long run.

Finally, the eighth family (Constraints (17) and (18)) prescribes that exactly one job will be the one that completes first on each open node n:

$$\sum_{j\in\mathcal{J}} \alpha_{jn} = \nu_n \qquad \forall n \in \mathcal{N} \tag{17}$$

$$\alpha_{jn} \ \leq\ \zeta_{jn} \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N}. \tag{18}$$

To characterize the system costs, we introduce continuous variables representing the execution delays with respect to the jobs due dates (τ_j and $\bar{\tau}_j$) and the energy-consumption costs (π_{jn}). In particular, the tardiness τ_j is defined by:

$$\sum_{\substack{n\in\mathcal{N}\\ i\in\mathcal{I}}} \left(t_{ji1}\xi_{jin1} \sum_{f\in\mathcal{F}} b_f\varphi_{jf} + \sum_{k=2}^{K_n} t_{jik}\xi_{jink} \right) \leq d_j + \tau_j, \tag{19}$$

where the first term represents the total execution time of job j if it is executed on a single GPU using a fraction $f \in \mathcal{F}$, and the second term defines the execution time if it is executed on multiple, dedicated GPUs.

The worst-case tardiness $\bar{\tau}_j$ is defined by prescribing:

$$(H + \bar{T}_j)\left(1 - \sum_{n\in\mathcal{N}} \zeta_{jn}\right) \leq d_j + \bar{\tau}_j, \tag{20}$$

where $\bar{T}_j$ is the maximum execution time of job j on all possible configurations. Indeed, by definition of ζ_{jn}, the sum at the left-hand side is 0 if the job is postponed (no node is selected for the execution), so that $\bar{\tau}_j$ denotes the worst possible delay occurring when job j is not executed in the current scheduling point and runs with the slowest possible configuration afterward.

Finally, the energy cost π_{jn} is defined by:

$$\sum_{i\in\mathcal{I}} \left(t_{ji1}\xi_{jin1} \sum_{f\in\mathcal{F}} b_f\varphi_{jf} + \sum_{k=2}^{K_n} t_{jik}\xi_{jink} \right) \left(\sum_{\bar{\kappa}=1}^{K_n} c_{n\bar{\kappa}}\gamma_{n\bar{\kappa}} \right) \leq \pi_{jn}, \tag{21}$$

where the first parenthesis corresponds to the total execution time of job j with the current configuration (defined, as in Constraints (19), by summing the contribution obtained if j is deployed on a GPU fraction and the one obtained if it is deployed on dedicated GPUs). The second parenthesis is the energy cost of node n, since $\gamma_{n\bar{\kappa}}$ is 1 when a total of $\bar{\kappa}$ GPUs are used on node n. Note that, with this definition, job j pays the energy cost due to the entire node usage, even if the $\hat{k}$ GPUs are partitioned among many jobs. However, this does not create issues since, in each scheduling interval, only one job will have $\alpha_{jn} = 1$ in each node, thus the cost terms are effectively considered only once.

Our optimization problem reads:

$$\min \mathbb{E}\left[\sum_{j\in\mathcal{J}} (\eta_j\tau_j + \rho\eta_j\bar{\tau}_j) + \sum_{\substack{j\in\mathcal{J}\\ n\in\mathcal{N}}} \alpha_{jn}\pi_{jn} \right] \tag{P1.1}$$

subject to: (2)-(21) and

$$\nu_n \in \{0,1\} \qquad \forall n \in \mathcal{N} \tag{P1.2}$$

$$\xi_{jink} \in \{0,1\} \qquad \forall j \in \mathcal{J},\ i \in \mathcal{I},\ n \in \mathcal{N}, k = 1 : K_n \tag{P1.3}$$

$$\sigma_{jg} \in \{0,1\} \qquad \forall j \in \mathcal{J},\ g \in \mathcal{G} \tag{P1.4}$$

$$\varphi_{jf} \in \{0,1\} \qquad \forall j \in \mathcal{J},\ f \in \mathcal{F} \tag{P1.5}$$

$$\zeta_{jn} \in \{0,1\} \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N} \tag{P1.6}$$

$$\gamma_{nk} \in \{0,1\} \qquad \forall n \in \mathcal{N},\ k = 1 : K_n \tag{P1.7}$$

$$\alpha_{jn} \in \{0,1\} \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N} \tag{P1.8}$$

$$\pi_{jn} \geq 0 \qquad \forall j \in \mathcal{J},\ n \in \mathcal{N} \tag{P1.9}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathcal{J} \tag{P1.10}$$

$$\bar{\tau}_j \geq 0 \qquad \forall j \in \mathcal{J}. \tag{P1.11}$$

The first term of the objective function (P1.1) represents the sum over all jobs $j \in \mathcal{J}$ of the penalties for due date violations. Tardiness and worst-case tardiness are multiplied by the weight η_j, defined so that violating the due date of high-priority jobs implies a higher penalty. Since job postponements may induce unforeseen delays if the submission of new jobs

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

TABLE II: Stochastic Model Parameters and Variables

Parameters	
s_k	job processing speed when using k GPUs
P_{on}	instantaneous power consumption of a GPU in-use
P_{idle}	instantaneous power consumption of an idle GPU
Variables	
$w(t)$	number of epochs (amount of work) executed at time t
$s(w)$	processing speed when w epochs have been completed
$Q(s)$	power consumption per epoch when operating at speed s
$P(s)$	instantaneous GPU power consumption at speed s
x_k	number of epochs after which we move from s_k to s_{k+1}

prevents from assigning them all the required resources, the worst-case tardiness is further penalized by a coefficient $\rho > 1$.

The second term corresponds to the total energy cost: for each node $n \in \mathcal{N}$, we add to the sum the energy cost π_{jn} related to the execution of the first job that terminates on node n under the current resource selection and scheduling, characterized by $\alpha_{jn} = 1$. This cost definition is reasonable because the system is fully reconfigured when a job terminates, thus the resources assigned to all jobs may change.

Note that all the terms in the objective function (P1.1) depend, through the definitions in Constraints (19), (20) and (21), on the jobs execution times. These are random variables because the number of epochs executed during the training is stochastic, therefore the total cost is optimized in the expected value. Moreover, all reconfiguration costs are neglected in our model, since the corresponding overheads are orders of magnitude smaller than DL training times (a few minutes against several hours or days [32]).

V. Stochastic Model for a Single DL Job

The RS-JS problem needs to be solved online within a short computing time (in the order of a few minutes). However, the MINLP formulation presented above is not solvable in such a short time, even using state-of-the-art MINLP solvers. Indeed, in our previous work [33], an instance including 40 nodes in a simplified problem setting where GPU sharing was neglected and a deterministic number of epochs were considered required about 15 min execution time making the adoption of state-of-the-art commercial solvers impractical. Therefore, we propose the STS heuristic method to obtain good solutions in a reasonable computing time.

The main idea of our heuristic is to focus on a single DL job j and, for every GPU type i, to compute the best allocation of GPUs that respects its due date d_j while minimizing the resulting expected energy cost.

In this section, we develop a stochastic model for the execution of a single job and identify structural properties that will yield the definition of an efficient algorithm for the computation of an optimal speed profile. Note that the model relies on the assumption that the number of available GPUs in the cluster is always enough to execute the considered job with its optimal configuration, i.e., that the cluster can allocate all GPUs to each job. While this is not true in practice, unless the load is very light, we will show in the next sections that the solution found under this theoretical setting can be effectively exploited to guide the choices of our STS method. The list of parameters and variables is reported in Table II.

A. Problem Formulation

The model presented in this section was first introduced in a previous work [17]. In Section VI, we will rely on this model to solve the global optimization problem heuristically by using the optimal GPU allocation for each job in isolation and then by constructing a global schedule by superposing each individual schedule. Therefore, since here we consider a fixed job j and GPU type i, we simplify the notation and drop indices j and i in the rest of this section. Let K represent the total number of GPUs and d the job due date. The number of epochs (or *work*) required to complete the job is a random variable W upper bounded by $w^{\max}$ and with complementary probability distribution function $F^c(w) := \mathbb{P}(W > w)$.

Now, let us define the following quantities:

- $w := w(t)$, the number of epochs executed at time t;
- $s := s(w) \in \{s_1, \dots, s_K\}$, the speed used when w epochs have been executed, $w \in [0, w^{\max}]$;
- $Q(s)$, the power consumption *per epoch* when operating at speed s, defined as:

$$Q(s) := P(s)\frac{\mathrm{d}t}{\mathrm{d}w} = \frac{P(s)}{s}, \tag{22}$$

  where derivatives are always intended as right derivatives and P follows the power-consumption model defined in Section III-B. Since $P(0) = 0$ and P is non-decreasing and convex, then Q is non-decreasing.

We aim to find a speed profile $s : [0, w^{\max}] \to \{s_1, \dots, s_K\}$ that minimizes the mean energy consumption to execute the given job constrained by the due date d. The mean energy consumption under speed profile s is defined by:

$$\mathcal{E}(s) = \int_0^{w^{\max}} F^c(w)\, Q(s(w))\, \mathrm{d}w.$$

Our objective is to solve the optimization problem:

$$\min_{s:[0,w^{\max}]\to\{s_1,\dots,s_G\}} \quad \int_0^{w^{\max}} F^c(w) Q(s(w))\, \mathrm{d}w \tag{P2.1}$$

subject to:

$$\int_0^{w^{\max}} \frac{\mathrm{d}w}{s(w)} \le d. \tag{P2.2}$$

Note that the constraint in (P2.2) simply states that the job has to be completed before the due date d. To see this, let $t(w)$ denote the time to execute the job up to w. Then,

$$\frac{\mathrm{d}t(w)}{\mathrm{d}w} = \frac{1}{s(w)}$$

and integrating both sides and using that $t(0) = 0$ and that $t(w^{\max}) \le d$, we obtain (P2.2).

B. A Convex Programming Solution

We show that the problem (P2) can be solved via convex programming, which implies that an optimal solution can be computed efficiently by using existing algorithms as the interior point method. The following Lemma and Propositions are reported here to make this paper self-contained; the corresponding proofs can be found in [17].

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

Lemma 1. *There exists an optimal speed schedule $s^*(w)$ of* (P2) *that is non-decreasing almost everywhere.*

The proof is based on Lusin's theorem for measurable functions and a compactness argument. Any non-decreasing speed schedule can be described by K (possibly empty) time intervals: Speed s_1 is used in $[0, x_1)$ (i.e., one GPU is used), speed s_2 is used in $[x_1, x_2)$ (i.e., the job is assigned two GPUs), and so forth up to s_K in $[x_{K-1}, x_K)$ with $x_K = w^{\max}$. This implies that the infinite-dimensional optimization problem (P2) can be recast into a discrete one:

$$\min_{x_k,\, k=1,\ldots,K-1} \quad \sum_{k=1}^{K} Q(s_k) \int_{x_{k-1}}^{x_k} F^c(w)\mathrm{d}w \tag{P3.1}$$

subject to:

$$\sum_{k=1}^{K} \frac{x_k - x_{k-1}}{s_k} \le d \tag{P3.2}$$

$$x_k \le x_{k+1}, \quad \forall k = 0, \ldots, K-1, \tag{P3.3}$$

where $x_0 = 0$ and $x_K = w^{\max}$. Here, the new decision variable x_k is interpreted as the number of epochs (i.e., the amount of work) after which the speed changes from s_k to s_{k+1}. At this point, one more GPU is allocated to the job.

The following proposition gives properties about (P3).

Proposition 1. *The optimization problem* (P3) *is convex, with a differentiable objective function when F^c is continuous. In addition, if F^c is strictly decreasing, then it is strictly convex.*

This means that a solution of (P3) can be computed using interior point methods [34] or even derivative-free optimization algorithms [35]. Note that if W is discrete, one can define a non-linear interpolation scheme to construct a continuous and strictly decreasing complementary probability distribution function that approximates the original one arbitrarily well.

C. Efficient Computation

By Proposition 1, an optimal speed profile may be directly computed by applying standard algorithms for convex programming. However, this approach does not fully exploit the particular structure of our problem as these algorithms essentially rely on convexity only. We now go beyond convexity and investigate further structural properties possessed by an optimal speed profile when the objective is strictly convex and regular. These will be used to develop Algorithm 1, which will find an optimal solution much faster than general-purpose algorithms from convex programming. These are given in Propositions 2 and 3 below and will lead us to the design of an algorithm (Algorithm 1) that will significantly speed up the computation of an optimal speed profile.

The first result says that only *consecutive* speeds are used. Specifically, if the job starts under speed, e.g., s_2 and then moves to speed s_3, either it runs with speed s_3 until it terminates or it moves to s_4, but there is no possibility to move directly from s_3 to any s_k with $k \ge 5$.

Proposition 2. *Assume that F is continuous and P is strictly convex. Then, the optimal schedule uses a consecutive set of speeds and always uses the maximal speed s_K.*

Now, given $m \in \{1, \ldots, K\}$, let us construct the speed profile $v := v_m : [0, w^{\max}] \to \{s_m, \ldots, s_K\}$ as follows:

$$v(w) = s_k \quad \text{if and only if} \quad w \in [y_{k-1}, y_k) \tag{23}$$

for all $k = m, \ldots, K$, where $y_0 = \cdots = y_{m-1} = 0$, $y_K = w^{\max}$ and the speed change vector point $(y_m, \ldots, y_{K-1}) \in [0, w^{\max}]^{K-m}$ satisfies:

$$y_k = (F^c)^{-1}\left(\frac{\Phi_m}{\Phi_k} F^c(y_m)\right), \quad m+1 \le k \le K-1, \tag{24}$$

where $(F^c)^{-1}(y) := \sup\{z : F^c(z) = y\}$,

$$\Phi_k := \frac{s_k P(s_{k+1}) - s_{k+1} P(s_k)}{s_{k+1} - s_k}$$

and

$$\sum_{k=m}^{K} \frac{y_k - y_{k-1}}{s_k} = d. \tag{25}$$

The expression for the y_k's in (24) is closely related to the KKT conditions of the optimization problem (P3).

The next proposition gives a property on v.

Proposition 3. *Assume that F is continuous and P is strictly convex. Then, there exists a unique optimizer that solves* (P2)*. In addition, if such optimizer uses all the speeds $s_1, \ldots, s_K$, then it is given by v with $m = 1$.*

By combining the previous propositions, we can design an algorithm that computes the optimal schedule *even when some speeds are not used in the optimal solution*; see Algorithm 1.

Algorithm 1 Dichotomy over the set of speeds for the optimal schedule computation (single job).

Input: The set of speeds $s_1, \ldots, s_K$, the power function P, the deadline d and the complementary probability distribution F^c.
Output: The optimal subset of speeds $s_U, \ldots, s_K$, and schedule $y_U, \ldots y_K$.
$U := K$; $L := 1$;
while $U > L$ **do**
  $m := \lfloor (U + L)/2 \rfloor$;
  Solve (25) for y_m using speeds $s_m, \ldots, s_K$, where $y_{m+1}, \ldots, y_K$ are given by (24);
  if $y_m \le 0$ **then**
    $L := m$;
  else
    $U := m$;
  end if
end while

We have the following result (see the proof in [17]):

Proposition 4. *Assume that F is continuous and P is strictly convex. Then, Algorithm 1 computes the (unique) optimal schedule. The complexity of this algorithm is $\log_2(K)$ times the complexity of solving the one-dimensional equation* (25)*.*

Let us comment on the computational complexity of Algorithm 1. First, it is clear that it halts after no more than $\log_2(K)$ iterations. Then, for each iteration, a one-dimensional equation needs to be solved, i.e., (25). Rearranging terms and using (24), this equation can be rewritten as:

$$d = \frac{w^{\max}}{s_K} + \sum_{k=m}^{K-1} y_k S_k = \frac{w^{\max}}{s_K} + \sum_{k=m}^{K-1} (F^c)^{-1}\left(\frac{\Phi_m}{\Phi_k} F^c(y_m)\right) S_k, \tag{26}$$

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

where $S_k := \frac{1}{s_k} - \frac{1}{s_{k+1}}$. We notice that (26) admits a unique solution and that F^c is differentiable almost everywhere because it is increasing. If derivatives can be computed, one can solve (26) by using (efficient) standard root-finding algorithms such as the Newton–Raphson or the secant methods.

VI. STochastic Scheduler

As previously mentioned, tackling the MINLP formulation presented in Section IV is impractical for state-of-the-art solvers, especially considering the stochastic nature of the parameters and, consequently, the objective function.

Therefore, in this section we illustrate the STS heuristic to determine high-quality solutions in a reasonable computing time. This method leverages the single-job stochastic model presented in Section V, under the assumption that energy costs increase linearly with the number of used GPUs (assumption that in practice holds see, e.g., the Cloud Providers pricing models in [36], [37]).

The STS method is reported in Algorithm 2 and includes three stages, which are described in the following sections.

Algorithm 2 STochastic Scheduler

1: **Input:** $\mathcal{J}$, Δ, $\mathcal{N}$, $\mathcal{S}_{old}$
2: **Output:** $\mathcal{S}$, $\mathcal{J}_p$
$\mathcal{J}$: list of submitted jobs; Δ: pressure indices of all jobs; $\mathcal{N}$: set of cluster nodes; $\mathcal{S}_{old}$: solution of the previous scheduling step
$\mathcal{S}$: new schedule; $\mathcal{J}_p$: list of postponed jobs
3: $\mathcal{S} \leftarrow \emptyset$
4: $\mathcal{J}_p \leftarrow \emptyset$
5: $\mathcal{N}_A \leftarrow \mathcal{N}$
6: $\mathcal{J}_s \leftarrow$ SORT_JOBS_LIST($\mathcal{J}$, Δ)
7: **while** $\mathcal{N}_A \neq \emptyset$ **do**
8: **for all** $j \in \mathcal{J}_s$ **do**
9: $\mathcal{D}_j \leftarrow$ SELECT_BEST_CONFIGURATIONS(j)
10: (assigned, n_j, D_j) $\leftarrow$ ASSIGN($j, \mathcal{D}_j, \mathcal{N}$)
11: **if** assigned **then**
12: $\mathcal{S} \leftarrow \mathcal{S} \cup (j, D_j, n_j)$
13: **if** n_j was saturated **then**
14: $\mathcal{N}_A \setminus \{n_j\}$
15: **end if**
16: **else**
17: $\mathcal{J}_p \leftarrow \mathcal{J}_p \cup \{j\}$
18: $\mathcal{S} \leftarrow \mathcal{S} \cup (j, \cdot, \cdot)$
19: **end if**
20: $\mathcal{J}_s \leftarrow \mathcal{J}_s \setminus \{j\}$
21: **end for**
22: **end while**
23: $\mathcal{S} \leftarrow$ POSTPROCESSING($\mathcal{S}, S_{old}$)
24: **return** $\mathcal{S}$, $\mathcal{J}_p$

A. Preprocessing Stage

The list of submitted jobs $\mathcal{J}$ is sorted (line 6) by assigning to all jobs a pressure index Δ_j that denotes how close they are to the corresponding due date. This step is based on the assumption that, in our framework, the penalties for due date violations are significantly larger than the energy costs. Since, moreover, for large systems (such as the ones that are usually tackled by Cloud Service Providers) the available resources may not be enough to execute all jobs concurrently, it is reasonable to prioritize jobs that have a higher risk of violation. Δ_j is defined as:

$$\Delta_j = T_c + \min_{i \in \mathcal{I}, n \in \mathcal{N}, k=1:K_n} \{t_{jik}\} - d_j, \tag{27}$$

where T_c denotes the current time and the minimum identifies the smaller possible execution time of job j with all the existing configurations. Note that this index can be further updated by multiplying it by η_j when it is strictly positive, which means prioritizing more jobs that are not only violating the due date but also subject to higher penalties.

It is worth noting that first-principle scheduling methods often considered in the literature, as EDF and FIFO, exploit modified versions of this pressure index defined as $\Delta_j^{EDF} = d_j$ or $\Delta_j^{FIFO} = \Theta_j$, respectively, where Θ_j is the submission time of job j.

B. Optimization and Assignment Stage

The goal of this stage (corresponding to the loop in lines 7–22) is to determine a high-quality schedule $\mathcal{S}$, defined by a set of tuples characterizing the configuration selected for each job j and the node where it is executed (if any). Specifically, we denote the configuration as $D_j = (i_j, k_j, f_j)$, i.e., as the type, number and fraction of GPUs assigned to job j. If n_j is the node where j is deployed, the corresponding element in the scheduler will thus be (j, D_j, n_j). Note that, when job j is postponed, this is denoted in the schedule as $(j, \cdot, \cdot)$.

For each job j in the sorted queue $\mathcal{J}_s$ defined at the previous step, the set $\mathcal{D}_j$ of optimal configurations is selected following the procedure reported in Algorithm 3, which is developed by adapting and iteratively solving the model presented in Section V. It is based on the assumption that, since jobs frequently terminate before their maximum number of epochs, it is reasonable to start executing them with low-power configurations, that are less expensive, and then progressively increase the resources as the due dates approach. This minimizes costs for jobs that terminate early while guaranteeing that the due dates are still met even for jobs that have a slow improvement rate and terminate at the maximum number of epochs. Note that the stochastic model of Section V was defined to cover a simpler scenario, where each job is deployed on a dedicated node, GPUs are homogeneous and there are infinite resources. This entailed that, on one hand, postponements never occur, and, on the other hand, the optimal resource profile assigned to each job can be determined upon submission. Indeed, since GPUs are infinite, future submissions do not affect the resource availability. This is not true in our setting, where the optimal resource selection needs to be dynamically adjusted over time. Therefore, we need to perform some adaptations:

- After solving the model for each job and GPU type separately to determine the best configurations, we demand the deployment to a different component, so that multiple jobs can be packed on a single node or GPU if needed.
- We translate the model over time so that if T_c is the current scheduling step, the due dates become $d_j - T_c$.
- Only the first configuration of the optimal resource profile, i.e., the number of GPUs k to be used at the beginning of the current scheduling step, is considered, and the stochastic model is periodically reevaluated.

Note that performing the last adjustment requires the following considerations. First of all, we need to determine the stochastic profile of partially executed jobs. For each job j, if w_c is the number of already executed epochs, we redefine the support of w as $\left[w_c, w_j^{\max}\right]$, we impose $x_0 = w_c$ and we

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

replace the probability measure $P(W)$ with $P(W \mid W > w_c)$ by exploiting the conditional probability theory. Furthermore, the model determines not only the optimal configuration for job j, but also the time T_j at which the resource assignment needs to be improved (i.e., the processing speed for the considered job should change from s_k to s_{k+1}). Therefore, other than solving the optimization problem every time a job terminates, we need to determine a new optimal schedule at each T_j. To guarantee that the optimal configuration selected by the model in two scheduling steps is consistent (i.e., the number of GPUs assigned to a job does not decrease over time), we set the minimum number of GPUs that can be selected for a job j, denoted as k_0, to $k+1$ if $T_c = T_j$, and to k otherwise, where k is the number selected in the previous schedule (we set $k_0 = 0$ if j was never executed).

The model is solved (line 5 of Algorithm 3) by the procedure described in Algorithm 1, keeping, as previously mentioned, only the first configuration. The list of all configurations selected for each GPU type $i \in \mathcal{I}$ is sorted (line 9) according to the value of the objective function (P3.1).

Once the procedure is complete and the set $\mathcal{D}_j$ is available, the assignment process starts (lines 10–19 of Algorithm 2). The *assign* function selects the node n_j where job j is scheduled for execution, exploiting the best configuration $D_j \in \mathcal{D}_j$ compatible with the available resources, if any, according to a best-fit approach with the aim to saturate the resources currently in use. If no resources are available, job j is returned to the queue (lines 17–18). Note that the job allocation always aims to saturate first the resources that are already partially used, since this contributes to minimizing the energy costs. During the assignment process, if the selected node n_j is saturated after having deployed job j, it is removed from the set $\mathcal{N}_A$ of available resources.

Algorithm 3 Optimization procedure

```
1: function SELECT_BEST_CONFIGURATIONS(j)
   I : GPU types; M_j : instance of stochastic model for j; P_ji : profile solution of M_j with GPU type i; T_c : current scheduling time
2:     D_j ← ∅
3:     for all i ∈ I do
4:         M_j ← DEFINE_MODEL(j, i, T_c)
5:         P_ji ← OPTIMAL_SOLUTION_FOR_SINGLE_JOB(M_j)
6:         D_ji ← GET_INITIAL_CONFIGURATION(P_ji)
7:         D_j ← D_j ∪ D_ji
8:     end for
9:     D_j ← SORT(D_j)
10:    return D_j
11: end function
```

C. Postprocessing Stage

Once the schedule $\mathcal{S}$ is complete, we try to minimize the amount of idle resources by increasing the number of GPUs or the GPU fractions assigned to the jobs when these remain unused. Moreover, if the type and number of GPUs assigned to a job do not change between $\mathcal{S}_{old}$ and $\mathcal{S}$, we attempt to allocate it to the same node, to avoid unnecessary migrations.

VII. Experimental Analysis

This section describes the experimental evaluation we designed to assess the quality of solutions obtained by the STS, comparing the energy costs and the due date violation penalties with the outcomes of other heuristic approaches by means of extensive simulations. Specifically, Section VII-A introduces the setup of the experiments and the approach we followed in the comparisons, while the results are reported in Section VII-B. Finally, a scalability analysis is discussed in Section VII-C. All the analyses are based on simulation. The code and all results presented in this section are available on Zenodo.[2]

A. Experimental Setup and Methodology

This section illustrates the experimental settings for evaluating the quality of the STS solutions. In particular, Section VII-A1 presents the methods we used to compare the obtained results, as well as the metric we defined to denote the solution quality. Section VII-A2 describes the experiment we performed to derive the stochastic profiles for job training times. Section VII-A3 describes how we generated the problem instances we tackled with the different methods, and Section VII-A4 presents the parameters we considered to investigate the impact of GPU sharing. Finally, Section VII-A5 describes the software and hardware setting of our tests.

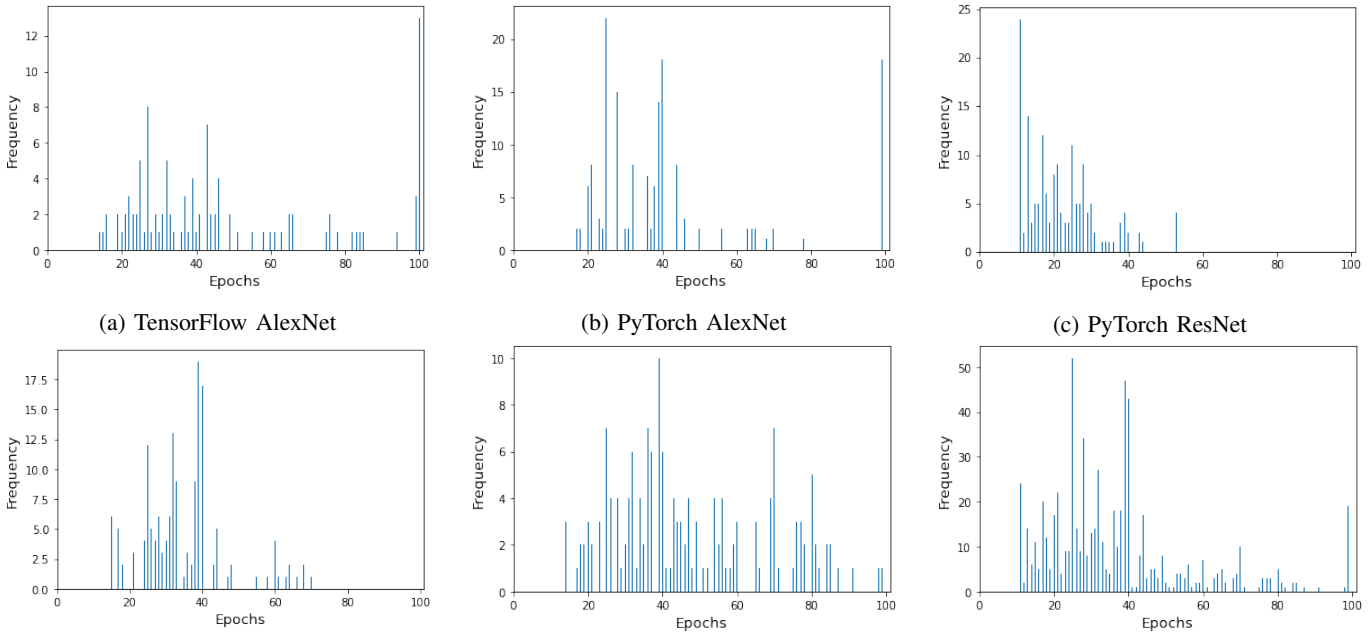
1) Alternative Methods: We compared STS with:

- Earliest-Deadline-First: a first-principle scheduling method often used for comparison in the literature [3], which we consider as our benchmark. It assigns resources to jobs giving higher priority to those closer to the due date. It is relevant to note that the configuration chosen for each job is never reevaluated, so that, if higher-priority jobs are submitted after long-running jobs with lower priority, they have to wait for them to be completed before receiving resources. Note that we chose EDF over other heuristics such as FIFO because it proved to be more effective in similar scenarios [11].
- Random Greedy: a heuristic method proposed in [11], which orders jobs according to the same pressure index defined in Equation (27) and selects the best configuration for each job j as (i) the cheapest configuration such that j completes before the due date, if possible, or (ii) the fastest available configuration if j is going to violate the due date with any type and number of GPUs. RG algorithm had to be adapted because it was developed to deal with cluster nodes that can be equipped on demand with different Virtual Machines, while for us the GPU type on each node is fixed.
- Dynamic Programming (DP): a heuristic method initially proposed in [4], that leverages dynamic programming to determine at each reconfiguration interval Δt the optimal batch size and number of GPUs required by all jobs, and then schedules them in FIFO order. As mentioned in [11], we extended this method by proposing alternative proxy functions, with the goal of better addressing the issue of due date violations and thus obtaining a more fair comparison with STS. In this work, we discuss the results obtained by considering a proxy function $\mathcal{F}_{WCT}$ defined, by adapting the corresponding equation from [11], as:

[2]https://doi.org/10.5281/zenodo.7438625

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540



(a) TensorFlow AlexNet (b) PyTorch AlexNet (c) PyTorch ResNet

(d) PyTorch VGG16 (e) PyTorch MobileNetV2 (f) PyTorch all networks

Fig. 2: Number of jobs ending at a specific epoch for different combinations of AI frameworks and neural networks

$$\sum_{j\in\mathcal{J}} \eta_j \tau_j + \sum_{n\in\mathcal{N}} \sum_{k=1}^{\Gamma_n} \gamma_{nk} c_{nk} \Delta t + \sum_{j\in\mathcal{J}} \left(1 - \sum_{n\in\mathcal{N}} \zeta_{jn}\right) \bar{\tau}_j. \quad (28)$$

The first term penalizes due date violations as the corresponding sum in Equation (P1.1). The second term defines the energy costs paid in the current scheduling step Δt. The last term adds a penalty corresponding to the worst-case tardiness $\bar{\tau}_j$ to all jobs whose execution is postponed. To further avoid due date violations, $\mathcal{F}_{WCT}$ is coupled with the choice of executing each job with the fastest available configuration.

Note that all these heuristics consider the worst-case job execution times, i.e., the expected training times when the jobs are executed up to the maximum number of epochs. Therefore, to extend the comparison we modify them to consider the average expected training times, derived from the distributions described in Section VII-A2 (they will be denoted as $avgEDF$, $avgRG$, and $avgDP$ respectively). In this case, whenever a job is trained for longer than the average time without terminating, the heuristic reverts to considering the worst-case execution time as in the original version.

In comparing the listed methods, we consider two different metrics. First of all, we define the *cost ratio* between any method m and a method b used as baseline:

$$cost\ ratio = \frac{C_m}{C_b}, \quad (29)$$

where C can denote either the total cost or the energy cost of the methods m and b, respectively. Note that considering as a metric the cost ratio instead of the absolute cost C_m allows to better capture the distance among methods and, moreover, makes the plots more readable if the cost of one method is significantly far from the others (due, e.g., to high tardiness penalties). Finally, we define the percentage cost-reduction (PCR) obtained by a *target* method with respect to any *other* approach as:

$$PCR = \frac{C_o - C_t}{C_o} \cdot 100, \quad (30)$$

where C_t is the cost obtained with the target method and C_o is the cost of the other approach, so that it is positive when the target method achieves lower costs than the other one. Also in this case, the PCR is computed considering either the total cost or the energy cost of the methods.

2) Stochastic Execution Profiles: To evaluate our STS method, we had to model the probability distributions for the number of epochs required to complete the training of a Deep Learning job. For this purpose, we collected data on the execution of DL tasks of various kinds using different neural networks and AI frameworks.

We considered a subset of the ImageNet competition dataset [38], including 10 classes with 1300 images each. We trained jobs exploiting ResNet[39], VGG16 [40], AlexNet [41] and MobileNetV2 [42], varying the batch size (16, 32, 64) and the optimizer (Adam, SGD). We set to 100 the maximum number of epochs for all jobs and record after how many epochs the jobs terminate using a stopping criterion based on patience. This corresponds to fixing a given number of epochs (we considered 10) after which the training is stopped if no improvement is observed on the validation accuracy. We performed a total number of 648 runs with PyTorch and 237 with TensorFlow, obtaining the results reported in Figure 2. The probabilities used in the simulations are obtained from these distributions to be comparable with real-life scenarios. Note that these probablities distributions are heterogeneous: those for AlexNet are skewed to the right with jobs ending at the maximum number of epochs, those for ResNet and

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

VGG16 are skewed to the left, and for MobileNetV2 we have a centered distribution.

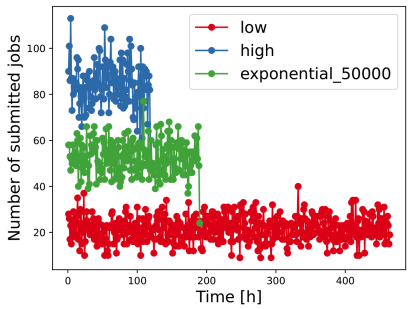


Fig. 3: Number of submitted jobs for every distribution

3) Randomly Generated Instances: We randomly generated a set of problem instances using the parameters described in the following. We varied the number $|\mathcal{N}|$ of cluster nodes between 10 and 100 and selected $|\mathcal{J}| = 10|\mathcal{N}|$ jobs from the application types listed in Section VII-A2. The set of GPU types includes NVIDIA K80, M60 and A100. Due to the recent high variability of energy prices in Europe, which did not allow to have stable estimates, we decided to follow a more conservative approach and consider as energy costs the market prices of the three GPU types in the cloud [37]. The values are reported in Table III.

TABLE III: Energy cost $[\$/h]$ for number and type of GPUs

#GPUs	1	2	3	4	5	6	7	8
K80	0.90	1.80	2.70	3.60	4.50	5.40	6.30	7.20
M60	1.14	2.28	3.42	4.56	5.70	6.84	7.98	9.12
A100	3.67	7.35	11.02	14.69	18.37	22.04	25.71	29.38

Jobs inter-arrivals are generated following three distributions, as proposed in [3], [4], [11]:

- an *exponential* distribution with mean $50000s/|\mathcal{N}|$; dividing by the number of nodes $|\mathcal{N}|$ guarantees a nearly constant per-node workload.
- two Poisson distribution with different rates, denoted as *high* and *low*. The former is set to $\lambda_h = \varepsilon\kappa_{max}\lambda$ while the latter is $\lambda_l = \lambda_h/4$, where λ is the reciprocal of the minimum expected completion time given the configurations available in the catalog, κ_{max} is the number of cluster nodes multiplied by the maximum number of GPUs that can be assigned to each job, and the parameter ε is tuned to match real-life scenarios (0.4 in our tests).

Sample workloads generated with the three distributions in a scenario featuring 1000 cluster nodes are reported in Figure 3.

The due dates d_j were sampled uniformly in the range:

$$\left[\min_{i,k} t_{jik}, \min\left(3\min_{i,k} t_{jik}, \max_{i,k} t_{jik}\right)\right], \tag{31}$$

while the tardiness weights η_j are sampled uniformly from the range $(0.0254, 0.0444)$ $\$/s$, so that the penalty for a time-unit due date violation is about 10 times larger than the time-unit energy cost.

Three problem instances were generated for each value of cluster size and each possible inter-arrival distribution, varying the random seed. Overall we considered 900 experiments. In the following, we report the average of the evaluated costs.

4) GPU Sharing Parameters: As mentioned in Section III, when enabling GPU sharing among multiple jobs, the corresponding execution times increase due to interference. The time-inflation coefficients b_f introduced to characterize this phenomenon were selected at random from a uniform distribution between 1.01 (i.e., 1% increase) and a $b_{f,max}$ that was set depending on the chosen fraction and literature models [43]. The list of admissible fractions and corresponding parameters for each GPU type is reported in Table IV, where m_j is the memory required to run job j. The NVIDIA K80 has 24 GB of memory and allows four jobs with $m_j \leq 6GB$ or two with $6GB < m_j \leq 12GB$ to run simultaneously. In contrast, the NVIDIA M60 is slightly more limiting, having only 16 GB of memory. Finally, most recent GPUs as the NVIDIA A100 have significantly larger memory (up to 80GB). This entails that the number of jobs that can be co-located increases, inducing better resource usage and, consequently, a more substantial cost reduction but introducing slightly higher performance variability.

TABLE IV: Possible GPU fractions and their parameters

GPU type	Memory m_j $[GB]$	Fraction f	$b_{f,max}$
K80	6	0.25	1.11
	12	0.5	1.05
M60	4	0.25	1.11
	8	0.5	1.05
A100	8	0.1	1.17
	12	0.15	1.15

We investigated the impact of GPU sharing by enabling or disabling it when running the STS method, considering the same random instances described in Section VII-A3. Since the execution times increase when co-locating jobs on the same GPUs, however, we adapted the due dates in this setting, increasing by 20% the right extremum of interval (31).

5) Software and Hardware Settings: Both the STS method and the heuristics used for comparison (EDF, RG and DP) are implemented in C++, to guarantee good scalability properties and fast execution times. We performed the experiments with 3 different random seeds for each job trace, and we repeated 10 times each run with the RG method. The server time required to complete the experiments (on an Ubuntu 18.04 VM based on a dual Intel Xeon Silver 4114 CPU at 2.20GHz with overall 40 cores and 64GB of memory) is about one week.

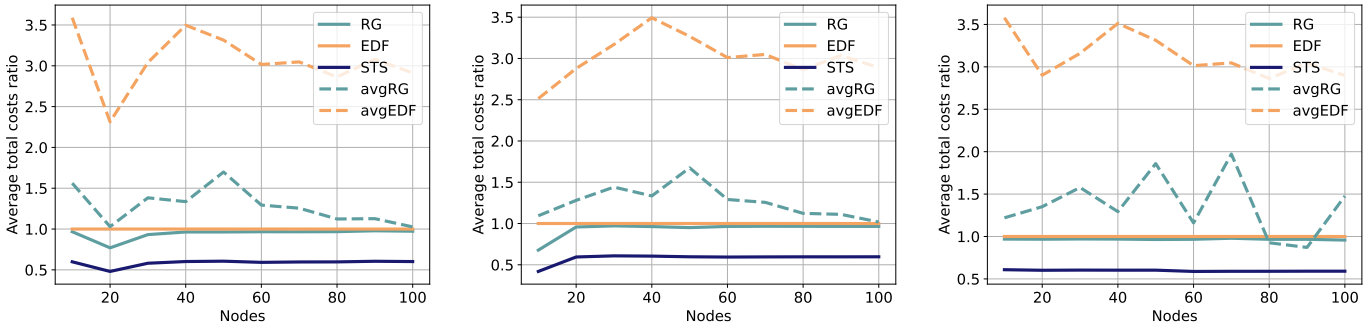
B. Experimental Results

In this section, we report the results of the experiments described in Section VII-A1. The methods are compared by considering the two metrics defined in Equations (VII-A1) and (VII-A1), namely the total or energy cost ratio and the PCR. The considered costs are always obtained by averaging over the three problem instances considered for each scenario (see Section VII-A3).

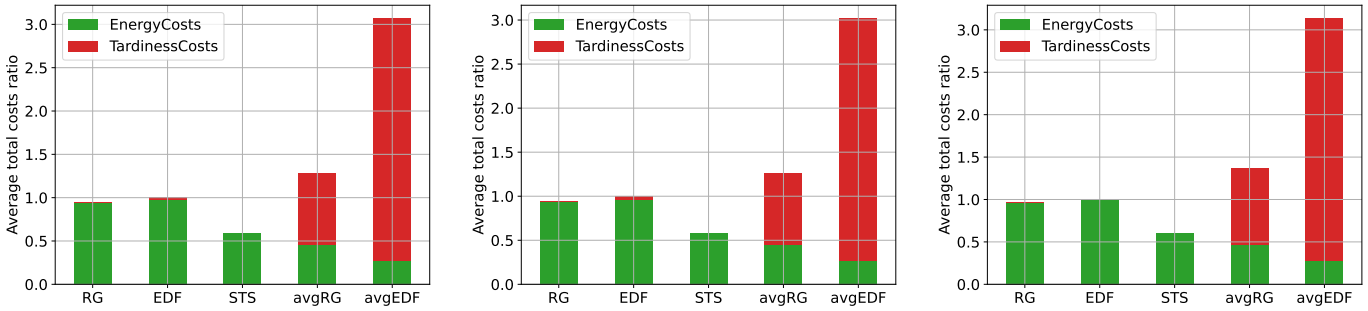
Figure 4 reports the comparison between STS, EDF, RG and the modified versions considering the average execution times, i.e., avgEDF and avgRG, when GPU sharing is not allowed. In particular, Figure 4a plots the average total cost ratio across the three random instances (computed according to Equation (VII-A1) with C_m equal to the total cost of STS,

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

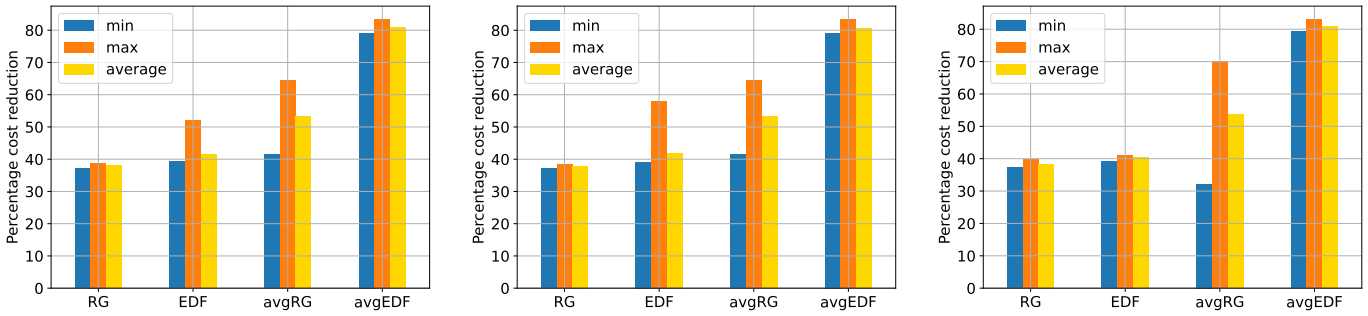
This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540



(a) Average over all random instances of the total cost ratio between each method and EDF, varying the number of nodes $|\mathcal{N}|$ (Exponential inter-arrival times; High rate; Low rate)



(b) Average across all the considered instances with different numbers of nodes of the total cost ratio between each method and EDF, separating the contribution of energy costs and due dates violation penalties (Exponential inter-arrival times; High rate; Low rate)



(c) PCR of STS against the other methods (Exponential inter-arrival times; High rate; Low rate)

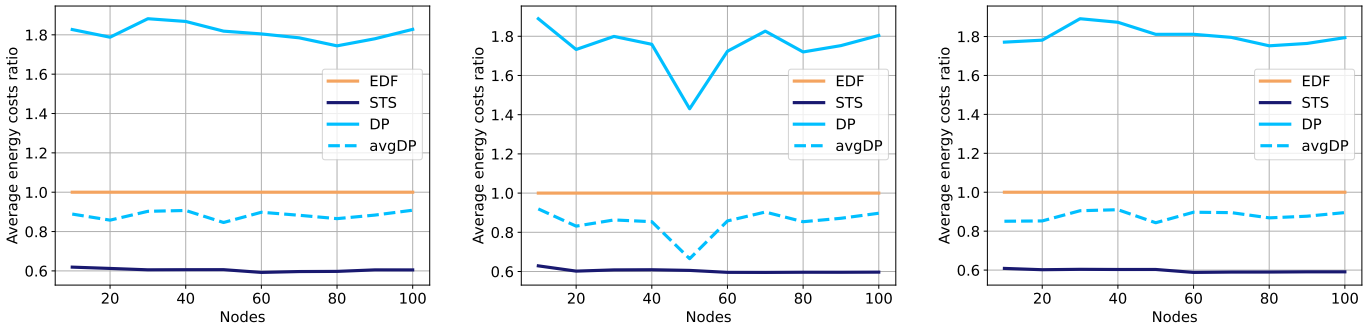
Fig. 4: Average total costs ratio and percentage cost reductions of STS against RG and EDF (no GPU sharing)

EDF, RG, avgEDF or avgRG, and C_b being the total cost of EDF) while increasing the number of cluster nodes $|\mathcal{N}|$. The total cost includes the sum of energy costs and penalties for due date violations. Each column in the plots of Figure 4b corresponds to the total cost ratio of each method averaged across all the problem instances with different cluster sizes $|\mathcal{N}|$. With respect to the previous case, we highlight the contribution to the total cost of energy costs and tardiness penalties, to verify whether the different methods effectively avoid due dates violations. Finally, Figure 4c reports the percentage cost reduction of STS with respect to the other methods (RG, EDF, avgRG and avgEDF), i.e., the PCR computed according to Equation (VII-A1) by considering STS as the *target* method.

From the first set of plots (Figure 4a), we note that the total cost ratio of STS over EDF is very stable with respect to the number of nodes $|\mathcal{N}|$ in the cluster. This is significant since it shows that our method performs well both for small and large systems. From the second set of plots (Figure 4b), we see that the energy costs achieved by avgRG and avgEDF are lower than those of STS. However, these methods do not manage to effectively avoid execution delays, which result in very significant penalties. Indeed, considering the average execution times, avgRG and avgEDF tend to assign minimum-cost configurations to the jobs; if these terminate earlier than expected, they get excellent results; otherwise, having to balance the slow initial training, frequently they are not able to recover the delay, and the jobs are led into tardiness. Since our priority is to prevent due date violations, these results show how the avgRG and avgEDF cannot be used in practice and confirm that consistent heuristics with respect to the worst-case execution times need to be developed to effectively minimize the energy cost by exploiting stochastic execution profiles. On the other hand, the energy costs delivered by STS are significantly lower than the ones obtained by both EDF and RG. Overall, we observe an average PCR of about 40% (and up to 80%) in all the considered scenarios (see Figure 4c).

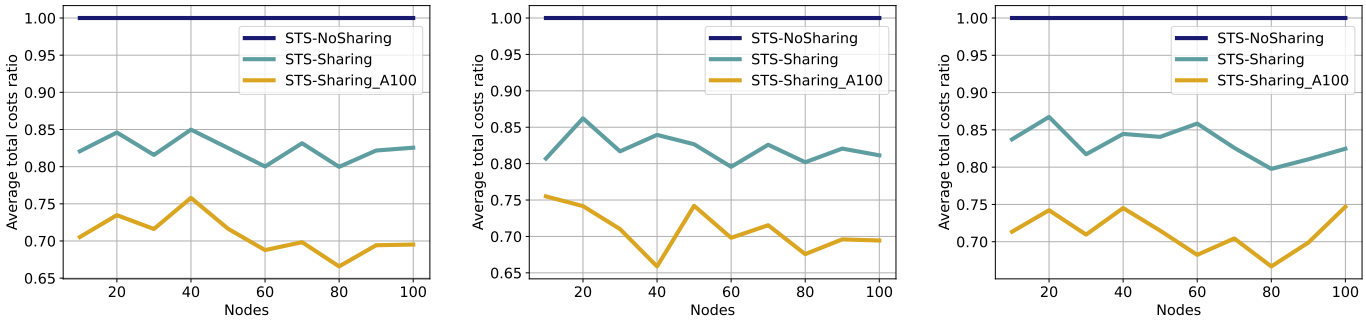
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

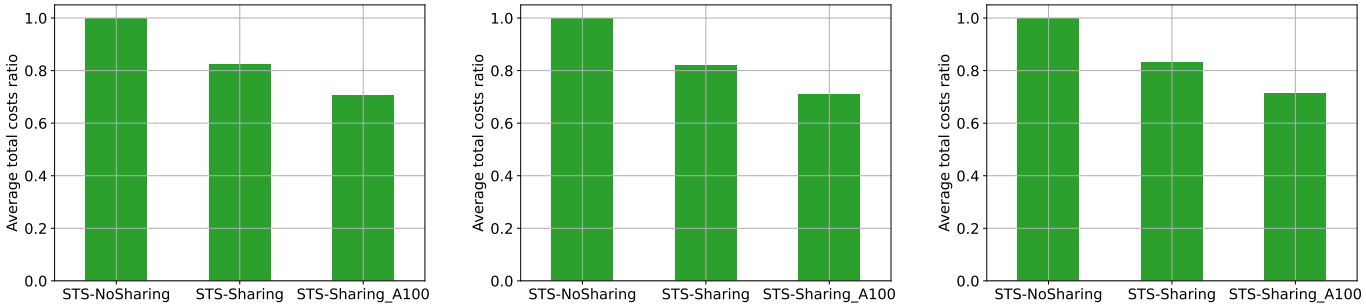


(a) Exponential inter-arrival times (b) High rate (c) Low rate

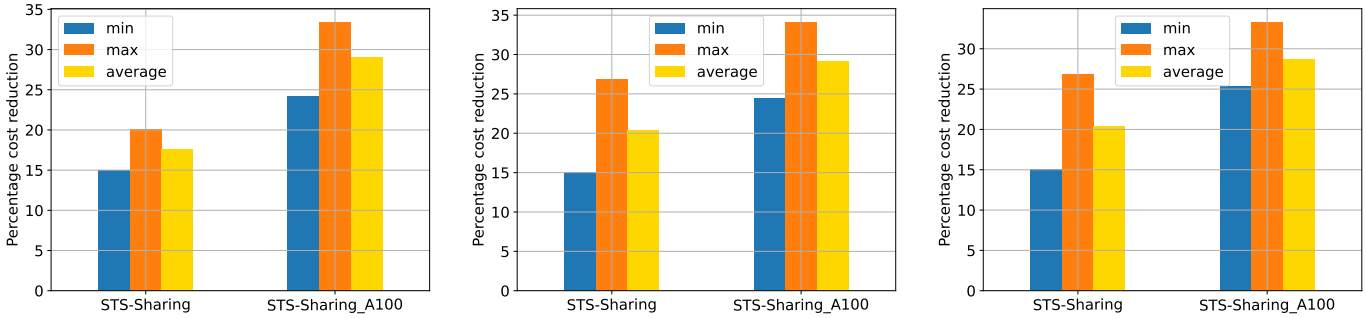
Fig. 5: Average over all random instances of the energy cost ratio computed by dividing the energy cost of STS, EDF and DP by the energy cost of EDF, varying the number of nodes $|\mathcal{N}|$ (no GPU sharing)



(a) Average over all random instances of the total cost ratio between STS exploiting GPU sharing and STS without GPU sharing, varying the number of nodes $|\mathcal{N}|$ (Exponential inter-arrival times; High rate; Low rate)



(b) Average across all the considered instances with different numbers of nodes of the total cost ratio between STS exploiting GPU sharing and STS without GPU sharing (Exponential inter-arrival times; High rate; Low rate)



(c) PCR of STS exploiting GPU sharing against STS without GPU sharing (Exponential inter-arrival times; High rate; Low rate)

Fig. 6: Average total costs ratio and percentage cost reductions of STS considering GPU sharing

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

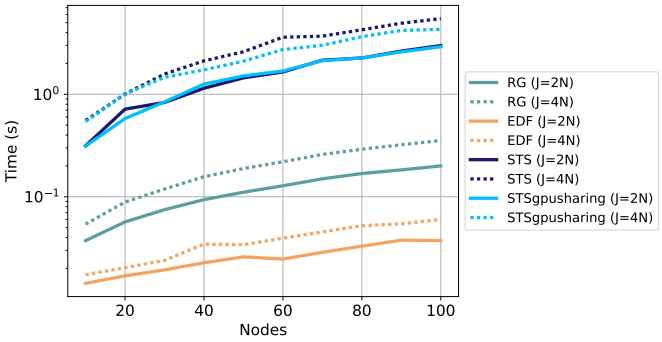
Similar considerations can be drawn from the comparison reported in Figure 5, where we plot the energy cost ratio computed according to Equation (VII-A1) (where C_m is the energy cost of STS, EDF, DP and avgDP, and C_b is the energy cost of EDF) averaged across the three problem instances defined while increasing the cluster size. The plots highlight how STS achieves better performance than all the others. The PCR computed by Equation (VII-A1) when considering STS as the *target* and DP as the *other* method is of 67%, while the PCR when avgDP is the *other* method is 32%.

Finally, the average total cost ratio obtained by STS when enabling GPU sharing are reported in Figure 6, together with the PCR achieved in this scenario. Here the baseline we consider in defining the cost ratio is the STS method when GPU sharing is not enabled, since our goal is to analyze the impact of jobs co-location on the same GPU. It is relevant to note that larger GPUs (A100) allow co-locating even more jobs, increasing the PCR from around 17% to around 29% on average. The corresponding results are reported separately with respect to the outcomes of GPU sharing on K80 and M60 GPUs since there is a significant gap both in performance and costs among these resources.

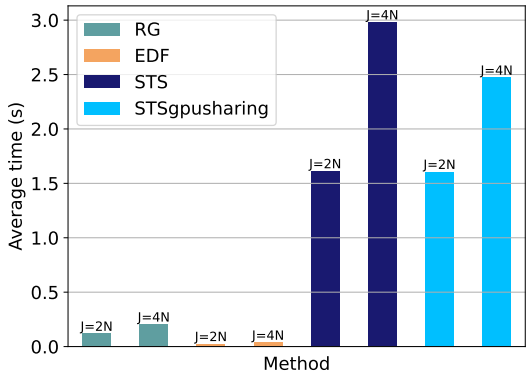
C. Scalability Analysis

To test the performance of our STS method, we performed a scalability analysis measuring the time required by our STS method, RG and EDF to solve a single problem instance involving a variable number of jobs and nodes. Specifically, we increased $|\mathcal{N}|$ from 10 to 100 and considered $|\mathcal{J}| = 2|\mathcal{N}|$ and $|\mathcal{J}| = 4|\mathcal{N}|$ to characterize light and heavy system loads. These values were chosen since we realized by inspecting the simulations reported in the previous sections that at most 400 jobs were concurrently in the queue $\mathcal{J}$ for problem instances including 100 cluster nodes.

The results are reported in Figure 7. In particular, Figure 7a reports the execution times of all methods while increasing the number of cluster nodes; Figure 7b reports the average execution times over all system sizes. We can note that the time required by STS to determine a solution, albeit higher than the one required by RG and EDF, is still considerably small, being lower than $5s$ even for systems involving 100 nodes and 400 concurrent jobs, and lower than $3s$ on average.



(a) Execution times with the different methods



(b) Average execution times with the different methods

Fig. 7: Scalability analysis changing the system load

VIII. Conclusion

In this paper, we presented a stochastic approach to model and tackle the optimal resource selection and scheduling problem for DL training jobs in GPU-based systems. We provided a mathematical formulation of the problem by defining the execution time as a stochastic variable, and we developed a stochastic heuristic that allocates jobs to the available resources to minimize the average energy costs while meeting the imposed due dates.

We set up an extensive experimental evaluation and performed simulations to test the quality of our method by comparing it with other literature approaches. The results confirm that our heuristic guarantees significantly better results than the existing ones, with a percentage total cost reduction between 32% and 80% on average in all the considered scenarios. Moreover, the results demonstrate that, depending on the workload and GPU memory, the possibility of co-locating multiple jobs on a single GPU can yield a percentage cost reduction between 17% and 29% on average.

Future developments of our work will include the support for multi-node executions, i.e., the possibility of running very demanding applications on GPUs hosted on different nodes involving, possibly, also mixed training and long inference workloads.

Acknowledgment

Federica Filippini and Danilo Ardagna's work has been funded by the European Commission under the H2020 grant N. 101016577 AI-SPRINT: AI in Secure Privacy pReserving computINg conTinuum.

References

[1] S. Madougou, A. Varbanescu, *et al.*, "The landscape of GPGPU performance modeling tools," *PC*, vol. 56, pp. 18–33, 2016.

[2] V. Anand. "'NVIDIA DGX A100,'" [Online]. Available: https://www.hardwarezone.com.sg/tech-news-nvidia-dgx-a100-supercomputer-super-performance-fight-covid-19 (visited on 01/09/2023).

[3] M. Amaral, J. Polo, *et al.*, "Topology-aware gpu scheduling for learning workloads in cloud environments," in *HPCNSA Proc.*, ACM, 2017.

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

[4] V. Saxena, K. R. Jayaram, and et al., "Effective elastic scaling of deep learning workloads," in *MASCOTS*, 2020, pp. 1–8.

[5] H. Albahar, S. Dongare, *et al.*, "Schedtune: A heterogeneity-aware gpu scheduler for deep learning," in *CCGrid Proc.*, 2022, pp. 695–705.

[6] B. Li, T. Patel, *et al.*, "Miso: Exploiting multi-instance gpu capability on multi-tenant gpu clusters," in *SoCC Proc.*, 2022, pp. 173–189.

[7] A. Qiao, S. K. Choe, *et al.*, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning.," in *OSDI*, vol. 21, 2021, pp. 1–18.

[8] B. Li, T. Patel, *et al.*, "Dash: Scheduling deep learning workloads on multi-generational gpu-accelerated clusters," in *IEEE HPEC*, 2022, pp. 1–7.

[9] Z. Yang, H. Wu, *et al.*, "Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous gpus," *IEEE Transactions on Computers*, pp. 1–13, 2023.

[10] F. Filippini, D. Ardagna, *et al.*, "ANDREAS: Artificial intelligence traiNing scheDuler foR accElerAted resource clusterS," in *FiCloud Proc.*, IEEE Computer Society, 2021, pp. 388–393.

[11] F. Filippini, M. Lattuada, *et al.*, "A path relinking method for the joint online scheduling and capacity allocation of dl training workloads in gpu as a service systems," *IEEE Trans. Serv. Comput.*, pp. 1–16, 2022.

[12] K. Mahajan, A. Balasubramanian, *et al.*, "Themis: Fair and efficient GPU cluster scheduling," in *USENIX (NSDI 20)*, 2020, pp. 289–304.

[13] S. Chaudhary, R. Ramjee, *et al.*, "Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning," in *EUROSYS*, 2020.

[14] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM*, 2019, pp. 505–513.

[15] G. Yeung, D. Borowiec, *et al.*, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE TPDS*, vol. 33, no. 1, pp. 88–100, 2022.

[16] Q. Weng, W. Xiao, *et al.*, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *USENIX NSDI*, 2022, pp. 945–960.

[17] J. Anselmi and B. Gaujal, "Energy Optimal Activation of Processors for the Execution of a Single Task with Unknown Size," in *MASCOTS Proc.*, 2022.

[18] H. Shen, L. Chen, *et al.*, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *ACM SOSP Proc.*, 2019, pp. 322–337.

[19] Y. Hu, R. Ghosh, and R. Govindan, "Scrooge: A cost-effective deep learning inference system," in *ACM SoCC Proc.*, 2021, pp. 624–638.

[20] H. Zhao, Z. Han, *et al.*, "HiveD: Sharing a GPU cluster for deep learning with guarantees," in *USENIX OSDI*, 2020, pp. 515–532.

[21] Z. Ye, P. Sun, *et al.*, "Astraea: A fair deep learning scheduler for multi-tenant gpu clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2781–2793, 2022.

[22] F. Xu, J. Xu, *et al.*, "Igniter: Interference-aware gpu resource provisioning for predictable dnn inference in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 812–827, 2023.

[23] C. Chen, Y. Chen, *et al.*, "Pickyman: A preemptive scheduler for deep learning jobs on gpu clusters," in *IEEE IPCCC*, 2022, pp. 120–129.

[24] Q. Hu, M. Zhang, *et al.*, "Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs," in *ASPLOS Proc.*, vol. 2, 2023, pp. 457–472.

[25] L. Liu, J. Yu, and Z. Ding, "Adaptive and efficient gpu time sharing for hyperparameter tuning in cloud," in *ACM ICPP Proc.*, 2023.

[26] M. Lattuada, E. Gianniti, *et al.*, "Performance prediction of deep learning applications training in GPU as a service systems," *Clust. Comput.*, vol. 25, no. 2, pp. 1279–1302, 2022.

[27] Y. Peng, Y. Bao, *et al.*, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *EUROSYS*, 2018.

[28] W. Xiao, R. Bhardwaj, *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *USENIX OSDI*, 2018.

[29] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," vol. 2, Jan. 2008, pp. 646–651.

[30] X. Zhan, Y. Bao, *et al.*, "Parsec3.0: A multicore benchmark suite with network stacks and splash-2x," *SIGARCH Comput. Archit. News*, vol. 44, no. 5, pp. 1–16, Feb. 2017.

[31] K. Kasichayanula, D. Terpstra, *et al.*, "Power aware computing on gpus," in *SAAHPC*, 2012, pp. 64–73.

[32] Q. Hu, P. Sun, *et al.*, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *SC21 Proc.*, 2021.

[33] F. Filippini, M. Lattuada, *et al.*, "Hierarchical Scheduling in on-demand GPU-as-a-Service Systems," in *SYNASC*, 2020, pp. 125–132.

[34] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, Mar. 2004.

[35] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Philadelphia, PA, USA: SIAM, 2009.

[36] "'Amazon ec2 on-demand pricing.'" (2023), [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls (visited on 01/25/2023).

[37] "'Azure cloud services pricing.'" (2023), [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing (visited on 01/25/2023).

[38] O. Russakovsky, J. Deng, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[39] K. He, X. Zhang, *et al.*, *Deep residual learning for image recognition*, 2015.

[40] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014.

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

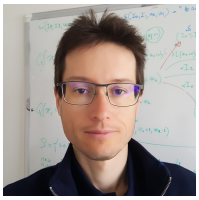
This article has been accepted for publication in IEEE Transactions on Cloud Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCC.2023.3336540

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[42] M. Sandler, A. Howard, *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018.

[43] G. Gilman and R. J. Walls, "Characterizing concurrency mechanisms for nvidia gpus under deep learning workloads," *Perform. Evaluation*, vol. 151, 2021.

Federica Filippini is a Ph.D. student at Politecnico di Milano where she received in April 2020 the Master degree in Mathematical Engineering. Her research interests include optimization problems applied to resource selection and scheduling in Cloud and distributed environments.



Jonatha Anselmi is a tenured researcher at the French National Institute for Research in Digital Science and Technology (Inria), since 2014. Prior to this, he was a full-time researcher at the Basque Center for Applied Mathematics and a postdoctoral researcher at Inria. He received his PhD in computer engineering at Politecnico di Milano (Italy) in 2009. His research interests focus on the performance evaluation and optimization of distributed systems.

Danilo Ardagna is Associate Professor at Politecnico di Milano, DEIB. He received a Ph.D. degree in computer engineering in 2004 from Politecnico di Milano. His work focuses on the design, prototype, and evaluation of optimization algorithms for resource management of cloud computing and big data systems.

Bruno Gaujal Bruno Gaujal is an Inria researcher. Till Dec. 2015, he has been the head of the large-scale computing team in Inria Grenoble-Alpes. He has held several positions in AT&T Bell Labs, Loria and École Normale Supérieure of Lyon. He obtained his PhD from University of Nice in 1994. He is a founding partner of a start-up company, RTaW, since 2007. His main interests are in performance evaluation, optimization and control of large discrete event dynamic systems with applications to telecommunication and large computing infrastructures.

© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.