iGniter: Interference-Aware GPU Resource Provisioning for Predictable DNN Inference in the Cloud

Fei Xu, *Member, IEEE*, Jianian Xu, Jiabin Chen, Li Chen, *Member, IEEE*, Ruitao Shang, Zhi Zhou, *Member, IEEE*, Fangming Liu, *Senior Member, IEEE*

Abstract—GPUs are essential to accelerating the latency-sensitive deep neural network (DNN) inference workloads in cloud datacenters. To fully utilize GPU resources, *spatial sharing* of GPUs among co-located DNN inference workloads becomes increasingly compelling. However, GPU sharing inevitably brings *severe performance interference* among co-located inference workloads, as motivated by an empirical measurement study of DNN inference on EC2 GPU instances. While existing works on guaranteeing inference performance service level objectives (SLOs) focus on either *temporal sharing* of GPUs or *reactive* GPU resource scaling and inference migration techniques, how to *proactively* mitigate such severe performance interference has received comparatively little attention. In this paper, we propose *iGniter*, an *interference-aware* GPU resource provisioning framework for cost-efficiently achieving predictable DNN inference in the cloud. *iGniter* is comprised of two key components: (1) a *lightweight* DNN inference performance model, which leverages the system and workload metrics that are practically accessible to capture the performance interference; (2) A *cost-efficient* GPU resource provisioning strategy that *jointly* optimizes the GPU resource allocation and adaptive batching based on our inference performance model, with the aim of achieving predictable performance of DNN inference workloads. We implement a prototype of *iGniter* based on the NVIDIA Triton inference server hosted on EC2 GPU instances. Extensive prototype experiments on four representative DNN models and datasets demonstrate that *iGniter* can guarantee the performance SLOs of DNN inference workloads with practically acceptable runtime overhead, while saving the monetary cost by up to 25% in comparison to the state-of-the-art GPU resource provisioning strategies.

Index Terms—Cloud-based DNN inference, predictable performance, GPU resource provisioning, performance interference

1 Introduction

With the proliferating artificial intelligence applications, deep neural network (DNN) inference workloads are becoming increasingly commonplace in cloud datacenters [1]. While DNN models are getting more complex and thus consuming more computation and memory resources, GPUs have served as the *key* accelerator to reduce the inference latency and meet the service level objective (SLO) [2]. Hence, modern internet companies like Google, Alibaba, and JD are increasingly adopting GPUs for serving DNN inference in their latency-critical products such as voice assistants [3], recommendation systems [4], and video analysis [5]. To cut down the inference budget and facilitate cloud-based DNN inference, most cloud providers have recently launched commercial cloud AI platforms such as AWS SageMaker [6] and Google Vertex AI [7]. As reported by Omdia, NVIDIA GPUs held an 80.6% market share of AI processors in cloud datacenters in 2020 and expect to reach 37.6 billion in revenue worldwide by 2026 [8].

To improve the utilization of GPU resources, *temporal sharing* [9] and *spatial sharing* [10] are two common GPU resource multiplexing techniques. Many existing works (*e.g.*, Cocktail [11], Clockwork [12]) leverage temporal sharing of GPUs to optimize the DNN inference performance and reduce the monetary cost. However, a recent study [13] has shown that temporal sharing of GPUs to execute DNN inference workloads can intrinsically result in GPU resource wastage. To *fully exploit* the computation and memory resources of GPUs, NVIDIA has recently developed the multi-process service (MPS) [14] technique, which allows multiple inference workloads to *spatially* share the GPU resources with a limited percentage [15] (*e.g.*, 50%).

Though MPS can configure an amount of GPU resources for each inference workload, there exists *noticeable performance interference* among the DNN inference workloads *co-located* on a GPU device. As evidenced by our motivation experiments in Sec. 2.2, the DNN inference latency can be prolonged by around 35% with only 5 co-located workloads on a GPU device. Such severe performance interference makes inference workloads easily suffer from unexpected SLO violations, which mainly originate from the shared *resource contention* in three aspects: (1) the *increased scheduling delay of kernels* by the GPU scheduler, and (2) the *severe*

- *Fei Xu, Jianian Xu, Jiabin Chen, Ruitao Shang are with the Shanghai Key Laboratory of Multidimensional Information Processing, School of Computer Science and Technology, East China Normal University, 3663 N. Zhongshan Road, Shanghai 200062, China. Email:* fxu@cs.ecnu.edu.cn.
- *Li Chen is with the School of Computing and Informatics, University of Louisiana at Lafayette, 301 East Lewis Street, Lafayette, LA 70504, USA. E-mail:* li.chen@louisiana.edu.
- *Zhi Zhou is with the Guangdong Key Laboratory of Big Data Analysis and Processing, School of Computer Science and Engineering, Sun Yat-sen University, 132 E. Waihuan Road, Guangzhou 510006, China. E-mail:* zhouzhi9@mail.sysu.edu.cn.
- *F. Liu is with Peng Cheng Laboratory, and Huazhong University of Science and Technology, China. E-mail:* fangminghk@gmail.com.

Manuscript received January XX, 2022; revised April XX, 2022.

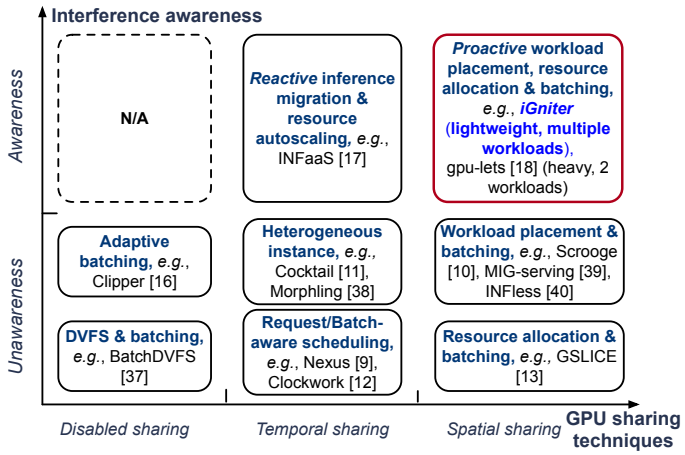

Fig. 1: *iGniter* positioning in the literature context of predictable DNN inference serving on GPUs.

contention of GPU L2 cache space, as well as (3) the *reduced GPU frequency due to limited power cap*. Accordingly, it is essential to explicitly consider performance interference when provisioning GPU resources to DNN inference workloads, in order to meet the stringent performance SLOs for users.

To guarantee the performance SLOs of DNN inference workloads, many research efforts have been devoted to batch size configuration (*e.g.*, Clipper [16]), request scheduling (*e.g.*, Clockwork [12]), resource autoscaling (*e.g.*, Cocktail [11]), and GPU resource allocation (*e.g.*, GSLICE [13]), as summarized in Fig. 1. However, they are *oblivious* to the severe performance interference among inference workloads, which is likely to cause resource under-provisioning and thus trigger *frequent reactive* adjustment of GPU resources. There have also been recent works on mitigating such performance interference through *reactive* inference migration (*e.g.*, INFaaS [17]) or characterizing the performance interference of *two* co-located workloads using a linear regression model (*e.g.*, gpu-lets [18]). Nevertheless, such an interference model requires a large number (*i.e.*, thousands) of workload profiling and cannot readily be applied to multiple co-located inference workloads. As a result, there has been scant research attention paid to achieving predictable DNN inference by characterizing the performance interference in a *lightweight* manner and *proactively* mitigating such interference for inference workloads.

To fill this gap, in this paper, we design and implement *iGniter*, an *interference-aware* GPU resource provisioning framework to achieve predictable performance [19] (*i.e.*, latency and throughput) of DNN inference workloads while minimizing the inference budget in the cloud. To the best of our knowledge, *iGniter* is the first attempt to demonstrate how to *characterize the performance interference of DNN inference on GPUs in a lightweight manner, and cost-efficiently provision GPU resources for inference workloads by jointly optimizing the GPU resource allocation and adaptive batching*. Specifically, we make the following contributions in *iGniter* as below.

▷ First, **we build a *lightweight* analytical performance model to explicitly capture the performance interference among DNN inference workloads** (Sec. 3). It empirically leverages a set of key system and workload metrics (*e.g.*, the GPU L2 cache utilization, the number of kernels) to characterize the severe contention of GPU scheduler, GPU L2 cache space, and GPU power consumption, as identified by our motivation experiments in Sec. 2.2.

▷ Second, **we propose a *cost-efficient* GPU resource provisioning strategy to guarantee the performance SLOs of DNN inference workloads** (Sec. 4.1). Given the DNN models with their performance SLOs, *iGniter* first leverages our inference performance model to calculate the appropriate batch size and lower bound of allocated GPU resources. It then greedily identifies the GPU device for placement with the minimum performance interference and allocates GPU resources for each inference workload.

▷ Finally, **we implement a prototype[1] of *iGniter* based on the NVIDIA Triton inference server** [20] with three pieces of modules, including an *inference workload placer* and a *GPU resource allocator* as well as an *inference performance predictor* (Sec. 4.2). We conduct prototype experiments on a cluster of 10 p3.2xlarge GPU instances with 12 representative inference workloads on Amazon EC2 (Sec. 5). Experiment results show that *iGniter* delivers predictable performance to DNN inference workloads with acceptable runtime overhead, while reducing the monetary cost by up to 25% compared with the state-of-the-art GPU resource provisioning strategies.

2 Background and Motivation

In this section, we first seek to analyze the severity of performance interference among co-located DNN inference workloads and identify the key factors that cause such interference. Next, we present an illustrative example to show how to adequately provision GPU resources for workloads to achieve predictable DNN inference.

2.1 Multi-Process Service of NVIDIA GPUs

To provide powerful computing ability, the NVIDIA GPU has been equipped with a number of Streaming Multiprocessors (SMs), and accordingly, GPUs are currently widely used for hosting DNN inference workloads in the cloud [12]. To improve the resource utilization of GPUs, NVIDIA MPS [14] has been developed to share GPU resources (*i.e.*, SMs) among multiple inference workloads executed on a single GPU device. One process commonly hosts one inference workload. However, an uncontrollable allocation of GPU resources can degrade the Quality-of-Service (QoS) of DNN inference workloads. To deal with such a performance issue, MPS provisions each DNN inference workload with an amount of limited GPU resources (*i.e.*, a set of SMs), starting from the NVIDIA Volta architecture [15]. In general, the batch size of DNN inference also requires tuning to improve the GPU resource utilization, without violating the performance SLOs of inference workloads [2].

The execution of a DNN inference workload on a GPU device mainly has three phases: *First*, the host CPU transmits the inference input data to the GPU device over the PCIe interconnect. *Second*, the GPU device executes the DNN inference query. *Finally*, the inference result is transmitted back to the host CPU via the PCIe interconnect. To improve the GPU resource utilization, the mainstream DNN inference servers (*e.g.*, NVIDIA Triton [20]) have developed

1. https://github.com/icloud-ecnu/igniter

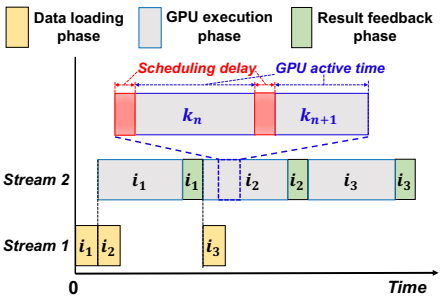

Fig. 2: CUDA streams mechanism overlaps the execution of different DNN inference queries (*i.e.,* i_1, i_2, i_3) in an inference workload, and the kernels (*e.g.,* k_n) are scheduled onto SMs during the GPU execution phase.

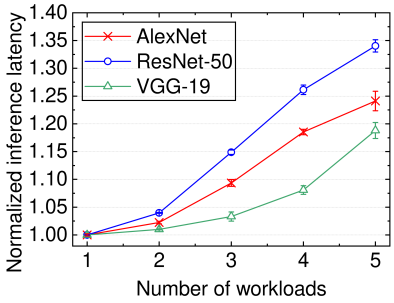

Fig. 3: Normalized inference latency of AlexNet, ResNet-50, and VGG-19 achieved on a V100 GPU, as the number of co-located inference workloads varies from 1 to 5, with respect to the workloads running alone.

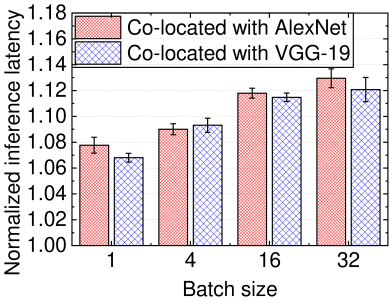

Fig. 4: Normalized inference latency of ResNet-50 when co-located with AlexNet or VGG-19 on a V100 GPU, as the batch sizes of AlexNet and VGG-19 vary from 1 to 32, with respect to ResNet-50 running alone.

the CUDA *streams* to *overlap* the data loading phase and the GPU execution phase of different DNN inference queries in an *asynchronous* manner. As shown in Fig. 2, the DNN inference queries (*i.e.,* i_1, i_2, i_3) are launched in two different streams which can be executed concurrently. Specifically, Stream 1 (*i.e.,* the data loading phase of i_2 and i_3) overlaps with Stream 2 (*i.e.,* the GPU execution phase of i_1 and i_2). In particular, an inference query consists of a number of kernels (*e.g.,* k_n) which require *scheduling* onto SMs [21], leading to a moderate amount of *scheduling delay* of kernels in the GPU execution stream.

2.2 Performance Interference among Co-located DNN Inference Workloads

Though MPS facilitates the spatial GPU resource sharing among co-located inference workloads, it still brings *non-negligible* performance interference. To examine the *severity* of such interference, we conduct two motivation experiments using p3.2xlarge EC2 instances [22] equipped with NVIDIA V100 GPUs. We use AlexNet [23], ResNet-50 [24], and VGG-19 [25] models executed on the NVIDIA TensorRT [26] framework as our DNN inference workloads. Specifically, we *first* launch 1 to 5 identical inference workloads concurrently and each is allocated 20% of GPU resources. *Second,* we launch two DNN inference workloads on a GPU, and each is allocated 50% of GPU resources. We vary the batch size of one workload from 1 to 32 while fixing the batch size of the other workload as 16. In particular, we measure the *average* DNN inference latency by excluding the inference batching delay. We illustrate the experimental results with error bars of standard deviation by repeating each experiment three times.

As shown in Fig. 3 and Fig. 4, the DNN inference latency increases from 0.83% to 34.98%, as the number of co-located workloads increases from 2 to 5 and the batch size of co-located inference workloads varies from 1 to 32. The experiment results indicate that the performance interference is *not uncommon* for MPS even with limited GPU resources (*i.e.,* GPU spatial sharing [14]). Our observation above is consistent with the findings in a more recent work [18]. Through an in-depth analysis, we find that such severe performance interference among DNN inference workloads is mainly caused by the following three factors.

Increased Scheduling Delay of Kernels. Each kernel of a DNN inference workload needs to be scheduled onto SMs by the GPU scheduler. As shown in Fig. 5, we observe that: *First,* the scheduling delay shows a roughly linear increase as the number of co-located workloads increases from 2 to 5. We conjecture that the GPU scheduler requires scheduling the kernels from different inference workloads onto SMs in a round-robin manner. *Second,* the scheduling delay of ResNet-50 increases much faster than AlexNet. This is simply because the number of kernels of ResNet-50 is bigger than that of AlexNet.

Severe Contention of GPU L2 Cache Space. Though MPS can partition GPU resources, the GPU L2 cache space is still shared by co-located DNN inference workloads [27]. To characterize the severity of such L2 cache contention on a GPU device, we simply adopt a system metric, *i.e.,* the L2 cache request hit ratio. As shown in Fig. 6, we observe that the GPU active time (*i.e.,* GPU execution latency - GPU scheduling delay, as depicted in Fig. 2) of ResNet-50 is inversely related to the GPU L2 cache hit ratio. As the number of co-located workloads increases, the severer cache contention leads to a smaller L2 cache hit ratio, which in turn increases the GPU active time of an inference workload.

Reduced GPU Frequency due to Limited Power Cap. Reduction of GPU frequency brings performance degradation to GPU workloads [28]. As shown in Fig. 7, we observe that: *First,* the GPU frequency starts to decrease once the GPU power reaches its upper limit value. This is because more inference workloads consume a larger amount of power on a GPU device, while the GPU has to maintain the upper limit of GPU power through frequency reduction. *Second,* the GPU power of VGG-19 and ResNet-50 shows a roughly linear relationship to the number of inference workloads, as long as the GPU power is below its upper limit value.

Based on our analysis above, we further explain why the batch size of co-located workloads (*i.e.,* AlexNet, VGG-19) can *moderately affect* the DNN inference performance (*i.e.,* ResNet-50) by 6.36% – 13.93%, as shown in Fig. 4. Such performance interference can mainly be attributed to the resource contention of GPU L2 cache space and GPU power. As the batch sizes of AlexNet and VGG-19 increase from 1 to 32, the GPU L2 cache utilization of the two workloads increases from 11.1% to 18.4% and from 16.9% to 22.0%, respectively. Similarly, the GPU power of AlexNet and VGG-19 also increases from 108 W to 156 W and from 139 W to 179 W, respectively, thereby causing GPU frequency reduction. Accordingly, such severe contention of the GPU L2 cache space and GPU power from co-located inference

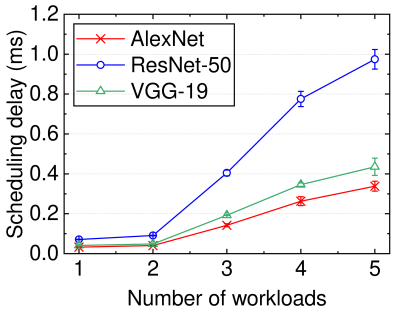

Fig. 5: Scheduling delay of AlexNet, ResNet-50, and VGG-19 with different numbers of workloads executed on a V100 GPU.

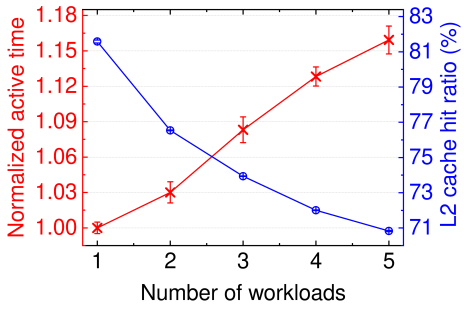

Fig. 6: GPU active time and L2 cache request hit ratio of ResNet-50 with different numbers of workloads executed on a V100 GPU.

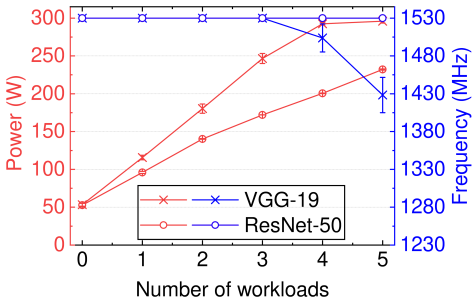

Fig. 7: GPU power and GPU frequency of VGG-19 and ResNet-50 with different numbers of workloads executed on a V100 GPU.

TABLE 1: Comparison of GPU resource provisioning plans and SLO violations achieved by the gpu-lets, GSLICE and our *iGniter* strategies for three representative DNN models (*i.e.*, AlexNet (A), ResNet-50 (R), VGG-19 (V)).

Approaches	Resource provisioning plans GPU: model(#resource, #batch)	Violations
GSLICE [13]	GPU1 : A(37.5%, 18), R(30%, 8), V(40%, 6)	2 models (A, R)
gpu-lets [18]	GPU1 : A(40%, 23) GPU2 : R(60%, 18), V(40%, 6)	2 models (A, R)
iGniter	GPU1 : A(10%, 4), R(30%, 8), V(37.5%, 6)	None

workloads inevitably prolongs the DNN inference latency.

Summary. *First*, the performance interference among DNN inference workloads cannot be overlooked. We identify the main factors that cause such interference as the severe contention of the GPU scheduler, GPU L2 cache space, and GPU power consumption among co-located inference workloads on a GPU device. *Second*, explicitly considering the performance interference is compelling when provisioning GPU resources to DNN inference workloads, so as to guarantee the performance of DNN inference workloads.

2.3 An Illustrative Example

To achieve predictable DNN inference performance and cost-efficient GPU resource provisioning, we propose *iGniter* in Sec. 4 and illustrate its effectiveness by conducting another motivation experiment with AlexNet, ResNet-50, and VGG-19 models. We set the latency SLOs (ms) and request arrival rates (req/s) for the three inference workloads as 15, 40, 60 and 500, 400, 200, respectively. We define the P99 latency of an inference workload exceeding its latency SLO as a violation.

As shown in Table 1, GSLICE [13] and gpu-lets [18] require 1 GPU and 2 GPUs, respectively. Unfortunately, they make two DNN models violate their SLOs. In contrast, our *iGniter* strategy provisions 1 GPU for hosting the three models appropriately and it guarantees their SLOs. Specifically, we find that GSLICE and gpu-lets tend to provision more GPU resources and larger batch sizes to AlexNet and ResNet-50 than *iGniter*. This is because the two strategies aim to maximize the request throughput while guaranteeing latency SLOs. In addition, GSLICE [13] is an *interference-unaware* strategy, which tunes the allocated GPU resources for inference workloads *separately*. Accordingly, the total allocated resources can exceed the maximum resources (*i.e.*, 100%) of a GPU device which inevitably leads to the contention of SMs, causing high long-tail inference latency.

Though gpu-lets [18] explicitly considers the performance interference, it works *only for two inference workloads* on a GPU device. Also, gpu-lets only considers the interference for the *newly-arrived* inference workload (*i.e.*, VGG-19), and it does not change the allocated GPU resources and batch size of the *originally-placed* workload (*i.e.*, ResNet-50) on the GPU. Accordingly, the inference latency of ResNet-50 exceeds its latency SLO due to the interference impact from VGG-19. Moreover, gpu-lets first provisions an *efficient* amount of GPU resources and then sets the batch size as large as possible for inference workloads. However, a large batch size cannot fully utilize the GPU resources at a low request arrival rate. It can cause SLO violations due to long batching latency. In contrast, *iGniter* sets an appropriate batch size for inference workloads that *just meet* their latency SLOs and request arrival rates. It further provisions GPU resources by explicitly considering the interference among multiple (more than 2) inference workloads to guarantee the DNN inference performance in a cost-efficient manner.

3 Modeling DNN Inference Performance on GPUs

In this section, we first build an analytical model to predict the DNN inference performance in the cloud. We explicitly consider the performance interference among DNN inference workloads with different batch sizes and allocated GPU resources. We next formulate the GPU resource provisioning problem to minimize the monetary cost while guaranteeing inference performance SLOs. The key notations in our performance model are summarized in Table 2.

3.1 Predicting DNN Inference Performance with GPU Resources

We consider a set of *constantly-arrived* DNN inference workloads denoted by $\mathcal{I} = \{i_1, i_2, ..., i_m\}$ over a period of time (*e.g.*, several minutes). A set of GPU devices to be allocated is denoted by $\mathcal{J} = \{j_1, j_2, ..., j_g\}$ with *a given GPU type*. As elaborated in Sec. 2.1, the execution of DNN inference on the GPU can be divided into three sequential steps: *data loading*, *GPU execution*, and *result feedback*. Accordingly, the DNN inference latency t_{inf}^{ij} of a workload i executed on a

TABLE 2: Key notations in our DNN inference performance model.

Notation	Definition
$\mathcal{I}, \mathcal{J}$	Sets of DNN inference workloads and allocated GPUs
t_{inf}^{ij}	DNN inference latency of an inference workload i on a GPU j
h^{ij}	Throughput of an inference workload i on a GPU j
t_{load}^{i}, $t_{feedback}^{i}$	DNN inference data loading latency and result feedback latency of an inference workload i
t_{gpu}^{ij}	GPU execution latency of an inference workload i on a GPU j
t_{sch}^{ij}, t_{act}^{ij}	Scheduling delay and GPU active time of an inference workload i on a GPU j
f^j	Actual frequency of a GPU j
p_{demand}^{j}	Total power demand of a GPU j
k_{act}^{i}	GPU active time of an inference workload i when running alone on a GPU device
p^i, c^i	Power consumption and L2 cache utilization of an inference workload i when running alone on a GPU device
r^{ij}, v^{ij}	GPU resource allocation and placement of an inference workload i on a GPU j
b^i	Batch size of an inference workload i

GPU device j can be calculated by summing up the data loading latency t_{load}^{i}, the GPU execution latency t_{gpu}^{ij}, and the result feedback latency $t_{feedback}^{i}$, which is given by

$$t_{inf}^{ij} = t_{load}^{i} + t_{gpu}^{ij} + t_{feedback}^{i}. \tag{1}$$

As discussed in Sec. 2.1, the data loading phase *overlaps* with the GPU execution and result feedback phases in the mainstream DNN inference servers (*e.g.,* Triton [20]) to improve the GPU resource utilization. Accordingly, we estimate the DNN inference throughput h^{ij} as

$$h^{ij} = \frac{b^i}{t_{gpu}^{ij} + t_{feedback}^{i}}, \tag{2}$$

where $b^i \in \mathcal{N}^+$ denotes the batch size of an inference workload $i \in \mathcal{I}$.

Data Loading and Result Feedback Phases. As discussed in Sec. 2.1, the inference input and result data are transmitted between the CPU and GPU devices via the PCIe. In general, both the inference input data size and result data are linear to the batch size b^i. We calculate the data loading latency t_{load}^{i} and the result feedback latency $t_{feedback}^{i}$ as

$$t_{load}^{i} = \frac{d_{load}^{i} \cdot b^i}{B_{pcie}} \quad \text{and} \quad t_{feedback}^{i} = \frac{d_{feedback}^{i} \cdot b^i}{B_{pcie}}, \tag{3}$$

respectively, where d_{load}^{i} and $d_{feedback}^{i}$ are the input data size and result data size, respectively, when $b^i = 1$. B_{pcie} denotes the available PCIe bandwidth of a GPU device.

GPU Execution Phase. Each DNN inference workload is executed with an amount of allocated GPU resources denoted by $r^{ij} \in [0, r_{max}], \forall i \in \mathcal{I}, j \in \mathcal{J}$, which are actually mapped to a set of SMs [14]. In general, r_{max} is set as 1. As depicted in Fig. 2, the GPU execution phase consists of GPU scheduling and kernels running on the allocated SMs (*i.e.,* r^{ij}). Moreover, the GPU execution phase can be prolonged by the GPU frequency reduction due to the workload co-location, as evidenced by Sec. 2.2. Accordingly, we formulate the GPU execution latency t_{gpu}^{ij} as

$$t_{gpu}^{ij} = \frac{t_{sch}^{ij} + t_{act}^{ij}}{\frac{f^j}{F}}, \tag{4}$$

where t_{sch}^{ij} and t_{act}^{ij} denote the total scheduling delay of kernels and the GPU active time of an inference workload i executed on a GPU device j, respectively, *without any GPU frequency reductions.* f^j and F denote the actual and maximum GPU frequency, respectively, on a GPU device j.

In the following, we first model the *scheduling delay* t_{sch}^{ij} of DNN inference workloads. Intuitively, t_{sch}^{ij} is roughly linear to the number of kernels n_k^i for a DNN inference workload i, which can be estimated as

$$t_{sch}^{ij} = (k_{sch}^{i} + \Delta_{sch}^{j}) \cdot n_k^i, \tag{5}$$

where k_{sch}^{i} denotes the scheduling delay when the workload i is running alone on a GPU device. Δ_{sch}^{j} is the increased scheduling delay caused by the interference on the GPU resource scheduler, which is relevant to the number of co-located inference workloads as evidenced by Sec. 2.2. Accordingly, we estimate the increased scheduling delay as

$$\Delta_{sch}^{j} = \begin{cases} 0 & \sum_{i \in \mathcal{I}} v^{ij} \leq 1, \\ \alpha_{sch} \cdot \sum_{i \in \mathcal{I}} v^{ij} + \beta_{sch} & \text{otherwise}, \end{cases} \tag{6}$$

where α_{sch} and β_{sch} are the coefficients to characterize the increased scheduling delay on a given GPU type. $\sum_{i \in \mathcal{I}} v^{ij}$ denotes the number of co-located inference workloads on a GPU device j. v^{ij} denotes whether an inference workload i is running on a GPU device j, which is given by

$$v^{ij} = \begin{cases} 1 & \text{a workload } i \text{ runs on a GPU } j \ (r^{ij} > 0), \\ 0 & \text{otherwise } (r^{ij} = 0). \end{cases} \tag{7}$$

We next model the *GPU active time* t_{act}^{ij} of an inference workload i executed on a GPU device j. As evidenced by Sec. 2.2, the GPU active time is inversely proportional to the GPU L2 cache hit ratio. We simply leverage a system metric called *GPU L2 cache utilization* to characterize the workload *demand* on the GPU L2 cache space. Given a fixed *supply* of L2 cache space on a GPU device, a higher GPU L2 cache utilization (*i.e., demand*) indicates severer contention on the GPU L2 cache space, thereby causing a longer GPU active time. Accordingly, we estimate t_{act}^{ij} as

$$t_{act}^{ij} = k_{act}^{i} \cdot \Big(1 + \alpha_{cache}^{i} \cdot \sum_{i \in \mathcal{I} \setminus i} \big(c^i \cdot v^{ij}\big)\Big), \tag{8}$$

where α_{cache}^{i} denotes the coefficient to characterize the prolonged GPU active time due to L2 cache contention for an inference workload i. k_{act}^{i} and c^i are the GPU active time and L2 cache utilization, respectively, when an inference workload i is running alone on a GPU device.

Finally, we model the *GPU frequency* f^j on a GPU device j. As evidenced by Sec. 2.2, the GPU frequency decreases dramatically as the total GPU power *demand* p_{demand}^{j} of workloads exceeds the upper limit of GPU power *supply*

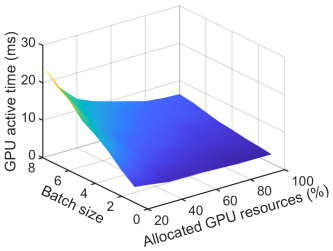

Fig. 8: GPU active time of ResNet-50 with different batch sizes and allocated GPU resources.

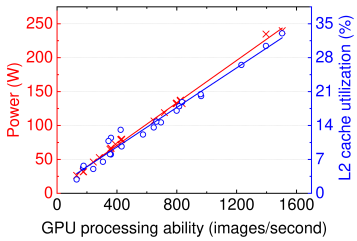

Fig. 9: Power consumption and L2 cache utilization of ResNet-50 with different GPU processing abilities.

P of a GPU device. As the GPU frequency is highly relevant to the GPU power [28], we estimate f^j as

$$f^j = \begin{cases} F & p^j_{demand} \leq P, \\ F + \alpha_f \cdot (p^j_{demand} - P) & p^j_{demand} > P, \end{cases} \quad (9)$$

where α_f denotes the coefficient to characterize the relationship between the GPU power and frequency on a GPU device. In addition, we estimate the total power demand of a GPU device j by summing up the power consumption p^i of all workloads and the *idle* power p_{idle} of a GPU device, which is given by

$$p^j_{demand} = p_{idle} + \sum_{i \in \mathcal{I}} (p^i \cdot v^{ij}). \quad (10)$$

In particular, we obtain p^i by running an inference workload i alone on a GPU device of the given type.

Obtaining Model Coefficients. Based on the above, we have 8 *workload-specific* coefficients (*i.e.*, d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache}) and 7 *hardware-specific* coefficients (*i.e.*, P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch}) in our performance model. Specifically, four workload-specific coefficients (*i.e.*, d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}) are obtained by profiling the workload *only once* using the `Nsight Systems` [29]. The available PCIe bandwidth B_{pcie} is measured by transferring data from the main memory to GPU memory. Given a GPU type, three hardware-specific coefficients (*i.e.*, P, F, p_{idle}) are obtained using the `nvidia − smi` [30]. The GPU frequency coefficient α_f and scheduling coefficients (α_{sch}, β_{sch}) as well as cache coefficient α^i_{cache} are obtained by launching multiple (*e.g.*, 2 to 5) inference workloads concurrently. Moreover, we obtain the GPU active time k^i_{act}, power consumption p^i, and the L2 cache utilization c^i of an inference workload i *running alone on a GPU device* as follows.

Specifically, as depicted in Fig. 8, the GPU active time k^i_{act} shows a roughly inverse proportion to the amount of allocated GPU resources r^{ij}. Also, the GPU active time increases fast with the batch size b^i, which can be formulated by a quadratic function. Accordingly, we formulate k^i_{act} as

$$k^i_{act} = \frac{k^i_1 \cdot (b^i)^2 + k^i_2 \cdot b^i + k^i_3}{r^{ij} + k^i_4} + k^i_5, \quad (11)$$

where k^i_1, k^i_2, k^i_3, k^i_4, k^i_5 denote the model coefficients for an inference workload i. In addition, Fig. 9 shows that both the power consumption p^i and L2 cache utilization c^i (measured by `Nsight Compute` [31]) of an inference workload i grow *linearly* with the GPU processing ability (*i.e.*, $\frac{b}{k_{act}}$). This is because a stronger GPU processing ability commonly leads to higher GPU resource utilization and power consumption. Accordingly, we estimate p^i and c^i as

$$p^i = \alpha^i_{power} \cdot \frac{b^i}{k^i_{act}} + \beta^i_{power},$$
$$c^i = \alpha^i_{cacheutil} \cdot \frac{b^i}{k^i_{act}} + \beta^i_{cacheutil},$$

where α^i_{power}, β^i_{power} and $\alpha^i_{cacheutil}$, $\beta^i_{cacheutil}$ denote the model coefficients to characterize the relationship between the power consumption, L2 cache utilization and the GPU processing ability. Such model coefficients above can be obtained by fitting several (*e.g.*, more than 5) sets of profiled workload data using the *least squares method* [32]. In particular, we only require profiling each inference workload with 11 different configurations of allocated GPU resources and batch sizes, which is far less than the number (*i.e.*, $40 \times 32 = 1,280$) of all possible configurations of allocated GPU resources (*e.g.*, 40 choices) and batch sizes (*e.g.*, 32 choices) for each inference workload, even without considering performance interference.

3.2 Analyzing GPU Resource Provisioning Optimization Problem

Based on our DNN inference performance model above, we proceed to define the optimization problem of GPU resource provisioning as follows: *Given the inference performance SLOs in terms of the request arrival rate R^i and latency SLO T^i_{slo}, how can we provision GPU resources r^{ij} and configure batch size b^i for each inference workload i, to achieve predictable DNN inference performance while minimizing the monetary cost C of allocated GPU resources?* Accordingly, our online optimization problem can be formulated as

$$\min_{b^i, r^{ij}} \quad C = \sum_{j \in \mathcal{J}} u^j \quad (12)$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} h^{ij} \cdot v^{ij} \geq R^i, \quad \forall i \in \mathcal{I} \quad (13)$$

$$\sum_{j \in \mathcal{J}} t^{ij}_{inf} \cdot v^{ij} \leq \frac{T^i_{slo}}{2}, \quad \forall i \in \mathcal{I} \quad (14)$$

$$\sum_{i \in \mathcal{I}} r^{ij} \leq r_{max}, \quad \forall j \in \mathcal{J} \quad (15)$$

$$\sum_{j \in \mathcal{J}} v^{ij} = 1, \quad \forall i \in \mathcal{I} \quad (16)$$

where u^j denotes the unit price of each GPU device j, and Eq. (12) defines our objective function which minimizes the monetary cost C of GPU resource provisioning, subject to the following four constraints. Specifically, Constraint (13) guarantees that the throughput of each inference workload can meet its arrival rate R^i. Constraint (14) guarantees the inference latency of each inference workload below its objective latency $\frac{T^i_{slo}}{2}$. This is because the batch inference latency cannot exceed half of the SLO [9] by excluding the performance impact of request batching and queueing. Constraint (15) denotes that the allocated GPU resources of each GPU device should be no more than the maximum GPU resources r_{max}. Constraint (16) denotes that each inference workload can only be placed on one GPU device.

Problem Analysis. According to Eq. (12), the monetary cost C is affected by the unit price u^j and set of allocated GPU devices $\mathcal{J}$, as the DNN inference models and requests *arrive constantly*. As u^j becomes a constant value u given a GPU type, the optimization problem can be reduced to minimizing the number $|\mathcal{J}|$ of provisioned GPU devices. To achieve such a goal, each inference workload requires to be allocated GPU resources that *just meet* the request arrival rate and latency SLOs.

Theorem 1. *Given a DNN inference workload with the arrival rate and latency SLO, the lower bound r^i_{lower} of allocated GPU resources (i.e., the allocated GPU resources that DNN inference workloads are running alone on a GPU device) and the appropriate batch size b^i_{appr} can be calculated as*

$$b^i_{appr} = \left\lceil \frac{T^i_{slo} \cdot R^i \cdot B_{pcie}}{2 \cdot (B_{pcie} + R^i \cdot d^i_{load})} \right\rceil, \quad (17)$$

$$r^i_{lower} = \left\lceil \frac{\gamma^i}{\delta^i \cdot r_{unit}} - \frac{k^i_4}{r_{unit}} \right\rceil \cdot r_{unit}. \quad (18)$$

where $\gamma^i = k^i_1 \cdot (b^i_{appr})^2 + k^i_2 \cdot b^i_{appr} + k^i_3$ and $\delta^i = \frac{T^i_{slo}}{2} - \frac{(d^i_{load}+d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch} \cdot n^i_k$. r_{unit} denotes the allocation unit of GPU resources, which can be empirically set as 2.5% (i.e., around 2 SMs) for NVIDIA V100 GPUs.

The proof can be found in Appendix A. Our selected *appropriate* batch size b^i_{appr} can guarantee the request arrival rate by letting $t^{ij}_{gpu} = \frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback}$. Accordingly, Constraint (13) and Constraint (14) can be combined as one constraint. The original optimization problem in Eq. (12) can be simplified as

$$\min_{r^{ij}} \quad \frac{u}{r_{max}} \cdot \left(\sum_{i \in \mathcal{I}} r^i_{lower} + \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} r^{ij}_{inter} + \sum_{j \in \mathcal{J}} r^j_f \right) \quad (19)$$

$$\text{s.t.} \quad \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} + \sum_{j \in \mathcal{J}} t^{ij}_{gpu} \leq \frac{T^i_{slo}}{2}, \ \forall i \in \mathcal{I}$$

$$(15), \ (16),$$

where $r^{ij}_{inter} = r^{ij} - r^i_{lower} \cdot v^{ij}$ is the *increased* GPU resources caused by the interference of co-located inference workloads. $r^j_f = r_{max} - \sum_{i \in \mathcal{I}} r^{ij}$ denotes the *unallocated* GPU resource fragments on a GPU device j. Accordingly, given the fixed lower bound r^i_{lower} of GPU resources, our optimization problem can be transformed into minimizing the *GPU resource fragmentation* and the *increased GPU resources* caused by the performance interference. Suppose that there is no performance interference among the inference workloads (*i.e.*, $r^{ij}_{inter} = 0$), our problem can be reduced to a classic *bin packing problem* which is already shown to be NP-hard [33]. Obviously, our original optimization problem is more *complicated* than such a bin packing problem. Accordingly, we turn to devising a heuristic algorithm to acquire an appropriate (*i.e., sub-optimal*) solution to our GPU resource provisioning problem.

4 Design of *iGniter*: Guaranteeing Performance of DNN Inference Workloads

Based on the analysis of our DNN inference performance model and the optimization problem defined in Sec. 3, we further present *iGniter* in Alg. 1, a *simple yet effective* GPU resource provisioning strategy to provide predictable performance (*i.e.,* guarantee the latency SLO and request arrival rate) for inference workloads, while minimizing the monetary cost of provisioned GPU resources in the cloud.

Algorithm 1: *iGniter:* Cost-efficient GPU resource provisioning strategy for achieving predictable performance of DNN inference workloads.

Input: The latency SLO T^i_{slo} and the request arrival rate R^i of each inference workload $i \in \mathcal{I}$.
Output: Cost-efficient resource provisioning plan, including the provisioned GPU resources r^{ij} and the appropriate batch size b^i_{appr} as well as the number of allocated GPUs g.

Acquire *hardware-specific* coefficients P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch} for a given GPU type, and obtain *workload-specific* coefficients d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache} through profiling each workload $i \in \mathcal{I}$;
Initialize: the appropriate batch size $b^i_{appr} \leftarrow$ Eq. (17), the lower bound of GPU resources $r^i_{lower} \leftarrow$ Eq. (18), and $r^{ij} \leftarrow 0, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}$, as well as $g \leftarrow 1$;
Sort workloads according to r^i_{lower} in descending order;
for all workload w in $\mathcal{I}$ to be placed on GPUs **do**
  Initialize: the allocated GPU resources $r^{ij}_a \leftarrow r^{ij}$, $\forall i \in \mathcal{I}, \forall j \in \mathcal{J}$, after placing an inference workload w, and the minimum increased GPU resources caused by the *performance interference* $r^{min}_{inter} \leftarrow r_{max}$, for placing the workload w on the GPU $q \leftarrow -1$;
  for all GPU device j in $[1, g]$ **do**
    $r^{ij}_a \leftarrow$ `alloc_gpus`$(T^i_{slo}, r^{ij}_a, r^w_{lower})$;
    Calculate the increased GPU resources caused by the *performance interference* $r^{ij}_{inter} \leftarrow r^{ij}_a - r^{ij}, \forall i \in \mathcal{I}$ on the GPU j;
    if $\left(\sum_{i \in \mathcal{I}} r^{ij}_a \leq r_{max}\right)$ && $\left(\sum_{i \in \mathcal{I}} r^{ij}_{inter} < r^{min}_{inter}\right)$ **then**
      Set $q \leftarrow j$, and $r^{min}_{inter} \leftarrow \sum_{i \in \mathcal{I}} r^{ij}_{inter}$;
    end if
  end for; // find an appropriate GPU for a workload w
  if $q == -1$ **then**
    Update $g \leftarrow g + 1$, and $r^{wg} \leftarrow r^w_{lower}$; // add one GPU
  else
    Update $r^{iq} \leftarrow r^{iq}_a, \forall i \in \mathcal{I}$; // enough GPU resources
  end if
end for

4.1 Algorithm Design

To particularly answer "*how to provision GPU resources for a set of DNN inference workloads,*" our *iGniter* strategy in Alg. 1 is quite *intuitive*: We first decide *where to place* inference workloads and then identify *how to allocate* GPU resources to the workloads. To particularly reduce the unallocated GPU resource fragments, *iGniter* sorts the inference workloads according to r^i_{lower} in *descending* order. It puts these workloads onto a new GPU device *only when* there are not enough GPU resources, accordingly to the `ANYFIT` constraint [33].

Inference Workload Placement Strategy. Given a set of DNN inference workloads with their latency SLOs T^i_{slo} and request arrival rates R^i, *iGniter* first obtains the *hardware-specific* coefficients (*i.e.*, P, F, p_{idle}, B_{pcie}, α_f, α_{sch}, β_{sch}) and the *workload-specific* coefficients (*i.e.*, d^i_{load}, $d^i_{feedback}$, n^i_k, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache}) for each inference workload using a *lightweight* coefficient acquisition method elaborated in

Algorithm 2: `alloc_gpus`: GPU resource allocation algorithm for placing an inference workload on a GPU device.

Input: The latency SLO T^i_{slo} and the allocated GPU resources r^{ij}_a of each inference workload $i \in \mathcal{I}$, before placing the inference workload w on the GPU j, as well as the resource lower bound r^w_{lower} of the inference workload w.
Output: Allocated GPU resources r^{ij}_a, after placing the inference workload w on the GPU j.

1: **Initialize:** the allocated GPU resources $r^{wj}_a \leftarrow r^w_{lower}$ of the workload w on the GPU j, and whether the GPU resources require reallocation $flag \leftarrow 1$;
2: **while** $\left(\sum_{i\in\mathcal{I}} r^{ij}_a \le r_{max}\right)$ && $(flag == 1)$ **do**
3: **Initialize:** $flag \leftarrow 0$;
4: **for all** inference workload i on the GPU j **do**
5: Calculate the inference latency $t^{ij}_{inf} \leftarrow$ Eq. (1);
6: **if** $t^{ij}_{inf} > \frac{T^i_{slo}}{2}$ **then**
7: Increase the allocated GPU resources $r^{ij}_a \leftarrow r^{ij}_a + r_{unit}$ for a workload i;
8: Set $flag \leftarrow 1$;
9: **end if**; // `SLO violation occurs`
10: **end for**; // `Reallocate GPU resources`
11: **end while**

Sec. 3.1 (line 1). With such obtained coefficients, *iGniter* calculates the appropriate batch size b^i_{appr} by Eq. (17) and the lower bound of allocated GPU resources r^i_{lower} by Eq. (18) (line 2). By iterating over the *sorted* inference workloads set $\mathcal{I}$, *iGniter* greedily finds an *appropriate* GPU device to host each workload (lines 3-12). In more detail, *iGniter* initializes the allocated GPU resources r^{ij}_a after placing the inference workload on the GPU (lines 5). For each candidate GPU, *iGniter* first calculates the allocated GPU resources r^{ij}_a and the increased resources r^{ij}_{inter} by Alg. 2 (lines 6-8). It then greedily identifies the *appropriate* GPU q which can host the inference workload and cause the *least* performance interference r^{min}_{inter} (lines 9-12). Finally, *iGniter* provisions a new GPU device if there are not enough resources for the inference workload w (*i.e.*, $q == -1$). Otherwise, it directly places such a workload w onto the GPU device q with the minimum increased GPU resources (lines 13-18).

GPU Resource Allocation Strategy. `alloc_gpus` first initializes the allocated GPU resources r^{wj}_a of the workload w as r^w_{lower} on the GPU j (line 1). `alloc_gpus` then iteratively *reallocates* the GPU resources for each workload i on the GPU j, as long as SLO violations still occur for an inference workload i and the GPU j has enough unallocated GPU resources (lines 2-11). Specifically, `alloc_gpus` calculates the inference latency t^{ij}_{inf} by Eq. (1) and judges whether the SLO violation occurs for each workload i (lines 4-6). For these SLO-violated workloads, `alloc_gpus` increases the allocated GPU resources by a unit of GPU resources (*i.e.*, r_{unit}) to guarantee the inference SLOs (lines 7-11).

Remark. As Alg. 1 (line 7) invokes Alg. 2, the time and space complexities of Alg. 1 are in the order of $\mathcal{O}(m \cdot g \cdot n \cdot \frac{m}{g})$ and $\mathcal{O}(m)$, respectively, where m denotes the number of inference workloads and g denotes the number of allocated GPUs. Also, $n = \frac{r_{max} - \sum_{i\in\mathcal{I}} r^{ij}_a}{r_{unit}} + 1$ denotes the cardinality of searching space of the allocated GPU resources for an inference workload. $\frac{m}{g}$ denotes the *expected* number of inference workloads co-located on a GPU. As n is practically limited (*i.e.*, at most 40 values in the real-world scenario), the time complexity of Alg. 1 can be reduced to $\mathcal{O}(m^2)$. To reduce the memory consumption of *iGniter*, we store the sparse matrix r^{ij} in Alg. 1 and Alg. 2 using *adjacency lists*, and accordingly the space complexities of Alg. 1 can be in the order of $\mathcal{O}(m)$. As a result, the runtime and memory overhead of our *iGniter* strategy is well contained and will be validated in Sec. 5.4.

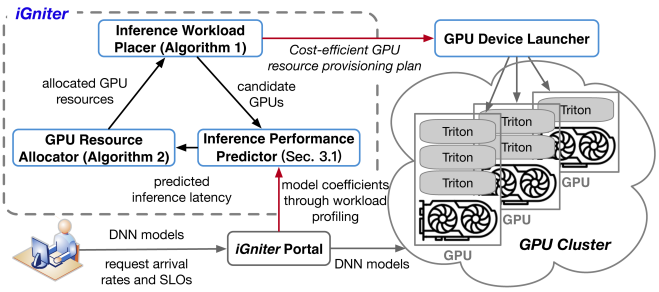

Fig. 10: Overview of our *iGniter* prototype in a GPU cluster.

In particular, *iGniter* can be generalized to the heterogeneous types of cloud instances (with different types of GPU hardware). Given multiple types of GPU instances, we only need to obtain the *hardware-specific* coefficients and a part of *workload-specific* coefficients (*i.e.*, k^i_{sch}, k^i_{act}, p^i, c^i, α^i_{cache} in line 1 of Alg. 1) for each type of GPU device. The rest of Alg. 1 can directly be executed without any modifications. Accordingly, *iGniter* can be easily extended to the heterogeneous cluster, by judiciously selecting the *most cost-efficient type of GPU instances* for DNN inference workloads, which will be validated in Sec. 5.3.

4.2 Implementation of *iGniter*

We implement a prototype of the *iGniter* framework running on Amazon EC2 GPU instances [22] based on NVIDIA Triton [20], which is a representative cloud inference server. More specifically, our *iGniter* prototype is built upon the Triton server v2.12.0 supported by the TensorRT backend framework v8.0.1.6, with over $1,000$ lines of Python, C++, and Linux Shell codes. The source codes of our *iGniter* prototype are publicly available on GitHub (*i.e.*, `https://github.com/icloud-ecnu/igniter`).

iGniter is *periodically executed* to provision GPU resources for newly-arrived inference workloads. As illustrated in Fig. 10, *iGniter* comprises three pieces of modules: an *inference workload placer* and a *GPU resource allocator* as well as an *inference performance predictor*. Specifically, users submit DNN models with their request arrival rates and SLOs to the *iGniter portal*, which can be deployed on a low-end EC2 instance. It initiates a *lightweight* workload profiling on *different types* of GPU devices to acquire the workload-specific and hardware-specific coefficients as elaborated in Sec. 3.1. With such coefficients, the *inference performance predictor* first estimates the inference latency using our performance model designed in Sec. 3.1. It then guides our *GPU resource allocator* and *inference workload placer* to identify an *appropriate* GPU device with the *least* performance interference and guaranteed SLOs from candidate GPUs. To particularly offset the interference impact, Alg. 2 can judiciously adjust allocated GPU resources for both the newly-arrived and originally-placed inference workloads

on a GPU device. According to our cost-efficient GPU resource provisioning plan generated by Alg. 1, the *GPU device launcher* finally builds a GPU cluster and launches the Triton inference serving process for each DNN inference workload on the provisioned GPU devices. In particular, the inference batch size is configured in Triton, and the GPU resources are allocated to each Triton process using the `set_active_thread_percentage` command in MPS.

Dealing with Performance Prediction Errors. The performance prediction errors can cause GPU resource *under-provisioning* to DNN inference workloads, thereby resulting in SLO violations. *iGniter* deals with such violations simply by pre-launching a *shadow* Triton inference serving process *standby* for each workload on a GPU device. Compared with the *original* inference process, such a *shadow* process is allocated an *extra* amount of GPU resources when *active*, which is set as the smaller value of the 10.0% of GPU resources (*i.e.,* the maximum prediction error measured in Sec. 5.2) and the remaining resources on a GPU device. Specifically, the DNN inference requests are first sent to the original Triton inference serving process. User clients then continuously monitor the accumulated P99 latency of each inference workload every second. Once the P99 latency of inference requests violates the latency SLO, *iGniter activates* the *shadow* inference process and kills the original process. It then *redirects* the upcoming inference requests to the *activated shadow* process. We will validate the robustness of *iGniter* in handling the performance prediction errors of DNN inference workloads in Sec. 5.3.

5 Performance Evaluation

In this section, we evaluate *iGniter* by carrying out a set of prototype experiments with four representative DNN models (as listed in Table 3) on Amazon EC2 [22]. Our prototype experiments seek to answer the following questions:

- **Accuracy:** Can our inference performance model in *iGniter* accurately predict the performance of DNN inference workloads? (Sec. 5.2)
- **Effectiveness:** Can our GPU resource provisioning strategy in *iGniter* provide predictable DNN inference while saving the monetary cost in the cloud? (Sec. 5.3)
- **Overhead:** How much runtime overhead of workload profiling and algorithm computation does *iGniter* practically bring? (Sec. 5.4)

5.1 Experimental Setup

GPU Cluster Configurations. We set up a GPU cluster of 10 p3.2xlarge EC2 instances, each equipped with 1 NVIDIA V100 GPU card, 8 vCPUs, and 61 GB memory. On each instance, we launch a Triton inference serving process and its corresponding client with a constant request arrival rate for each DNN inference workload. We measure the seven *hardware-specific* coefficients using the `Nsight Systems` and `nvidia − smi` according to Sec. 3.1. The maximum power P, maximum frequency F, idle power p_{idle}, and available PCIe bandwidth B_{pcie} of NVIDIA V100 are 300 W, 1530 MHz, 53.5 W, and 10 GBps, respectively. The power coefficient α_f, scheduling coefficients α_{sch} and β_{sch} are profiled as -1.025, 0.00475 and -0.00902, respectively.

TABLE 3: Configurations of three Apps with four performance SLOs, *i.e.,* latency (ms) and throughput (req/s) for four representative DNN inference models with *heterogeneous* workload characteristics.

Workload features		AlexNet	ResNet-50	VGG-19	SSD
GFLOPs		0.77	4.14	19.77	62.82
Params (MB)		61.10	25.56	143.67	26.29
App1	Latency	10	20	20	25
	Throughput	1200	400	300	150
App2	Latency	15	30	30	40
	Throughput	400	600	400	50
App3	Latency	20	40	40	55
	Throughput	800	200	200	300

Configurations of DNN Inference Workloads. We select four representative DNN models as listed in Table 3. The AlexNet [23], ResNet-50 [24], and VGG-19 [25] models are used for image classification running on the ImageNet dataset [34], while the SSD [35] model is used for object detection running on the VOC2012 dataset [36]. The four models (AlexNet, ResNet-50, VGG-19, and SSD) have *heterogeneous* workload characteristics, *i.e.,* computation complexity (GFLOPs) and model size (parameters), as elaborated in Table 3. In particular, we use $\{W1, \cdots, W12\}$ to denote the 12 DNN inference workloads with various performance SLOs in terms of latency SLOs and request arrival rates (*i.e.,* expected throughputs) for App1, App2, and App3.

Baselines and Metrics. We compare *iGniter* with the following three strategies: (1) **FFD$^+$:** the First-Fit Decreasing (FFD) algorithm which always allocates the lower bound of GPU resources r^i_{lower} and places inference workloads using FFD; (2) **GSLICE$^+$:** GSLICE [13] patched with our inference workload placement strategy, which tunes the allocated GPU resources and batch sizes according to the average latency and throughput of workloads; (3) **gpu-lets$^+$:** the modified gpu-lets [18], which allocates the GPU resources by maximizing the request throughput and places inference workloads on the best-fit GPUs. We also change the batch size configuration strategy of gpu-lets$^+$ by increasing the batch size to *just* meet the request arrival rate (the same as *iGniter*), as large batch sizes cannot adapt to a low request arrival rate as evidenced in Sec. 2.3. In addition, we focus on two key metrics including the *monetary cost* and *SLO violations*, as elaborated in Sec. 2.3. We particularly calculate the *hourly* monetary cost ($\$/h$) by multiplying the number of provisioned GPU instances and the hourly price of each instance. We do not multiply it by the inference execution time, simply because the model inference requests arrive constantly from users in our scenario.

5.2 Validating Inference Performance Model in *iGniter*

We evaluate the inference latency of AlexNet, ResNet-50, VGG-19, and SSD by varying the amount of GPU resources, batch size, and the number of co-located inference workloads. We compare our *iGniter* performance model with the state-of-the-art gpu-lets$^+$ model [18]. We illustrate the observed inference latency with error bars of standard deviation by repeating experiments three times.

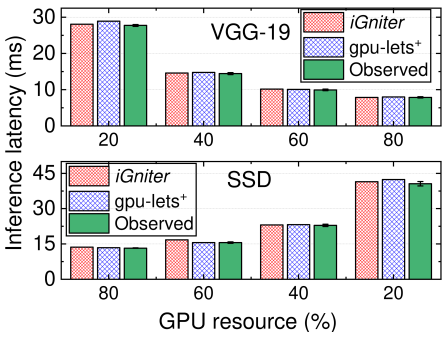

Fig. 11: Comparison of the observed and predicted inference latency of co-located VGG-19 and SSD with different allocated GPU resources and batch size set as 3 under the gpu-lets$^+$ and *iGniter* performance models.

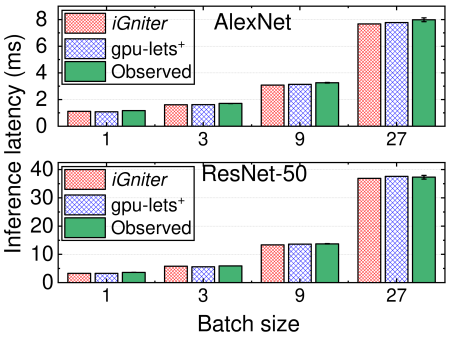

Fig. 12: Comparison of the observed and predicted inference latency of co-located AlexNet and ResNet-50 with 50% of allocated GPU resources and different batch sizes under the gpu-lets$^+$ and *iGniter* performance models.

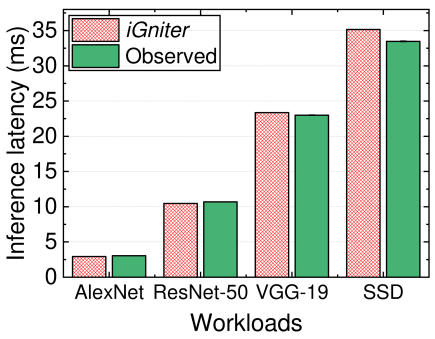

Fig. 13: Comparison of the observed and *iGniter* predicted inference latency of co-located AlexNet, ResNet-50, VGG-19 and SSD with 25% of allocated GPU resources and batch size set as 3.

Can *iGniter* accurately predict the inference latency with different amounts of GPU resources? As shown in Fig. 11, *iGniter* can well predict the inference latency with a prediction error of 0.04% – 2.32% for VGG-19 and 0.89% – 7.61% for SSD, compared with 1.30% – 4.19% and 0.02% – 4.43% under gpu-lets$^+$. Specifically, our predicted inference latency of SSD is basically higher than gpu-lets$^+$ and the observed latency. This is because the active time of SSD predicted by our model is longer than the actual active time, and the contention of GPU power consumption and L2 cache utilization further makes it worse. However, gpu-lets$^+$ offline profiles the actual inference latency for all possible configurations when SSD is running alone. In addition, the predicted inference latency of VGG-19 under *iGniter* is more accurate than that under gpu-lets$^+$. This is because gpu-lets$^+$ does not consider the contention of the GPU scheduler and power consumption. The GPU frequency for running VGG-19 drops from 1,530 MHz to 1,440 MHz due to GPU power contention, which makes the prediction error of gpu-lets$^+$ larger than *iGniter* for VGG-19.

Can *iGniter* accurately predict the inference latency with different batch sizes? As depicted in Fig. 12, *iGniter* can basically predict the DNN inference latency with a prediction error of 3.91% – 5.90% for AlexNet and 1.10% – 9.29% for ResNet-50, compared with 2.67% – 6.23% and 0.78% – 9.76% of gpu-lets$^+$. Specifically, the predicted inference latency of AlexNet under *iGniter* is smaller than the observed latency. This is because the data loading and result feedback phases occupy a larger part (*i.e.*, 7% – 20%) of the inference latency for AlexNet than that for other models (*i.e.*, 1% – 7%). It makes AlexNet share the PCIe bandwidth for a long period of time with other workloads. However, we simply assume that the contention of the PCIe bandwidth can be negligible. Also, *iGniter* underestimates the inference latency of ResNet-50 with a prediction error of 9.29% when the batch size is set as 1. This is because the average GPU active time of ResNet-50 is relatively small (*i.e.*, 0.04 ms), which makes it more sensitive to the GPU scheduler contention than other workloads. As *iGniter* explicitly considers such contention of GPU scheduler, the average prediction error of *iGniter* (*i.e.*, 3.82%) is smaller than that of gpu-lets$^+$ (*i.e.*, 4.15%) for ResNet-50.

Can *iGniter* adapt to the co-location of multiple (4+) inference workloads? As shown in Fig. 13, we observe that *iGniter* can accurately predict the inference latency of the four co-located workloads with a prediction error of 1.53% – 5.02%, while gpu-lets$^+$ fails to predict the inference latency of more than two co-located inference workloads. Specifically, our *iGniter* model captures the interference on the GPU scheduler (Eq. (6)), L2 cache space (Eq. (8)), and power consumption (Eq. (9)) for multiple co-located inference workloads. Taking VGG-19 as an example, *iGniter* can well predict the inference latency with a prediction error of 4.19% when co-located only with SSD (in Fig. 11) and 1.53% when co-located with three inference workloads (*i.e.*, AlexNet, ResNet-50, and SSD in Fig. 13), respectively. The rationale is that: when VGG-19 is co-located with two more workloads (*i.e.*, AlexNet, ResNet-50), *iGniter* can still predict the increase of GPU scheduling delay from 0.19 ms to 0.36 ms and the decrease of GPU active time from 27.54 ms to 22.31 ms (as allocated 5% more GPU resources), as well as the drop of GPU frequency from 1,530 MHz to 1,515 MHz.

5.3 Effectiveness of GPU Resource Provisioning Strategy in *iGniter*

To illustrate the effectiveness of our *iGniter* resource provisioning strategy, we conduct extensive experiments with the 12 inference workloads in Table 3. Specifically, we measure the P99 latency of inference workloads within a period of time (*e.g.*, 30 seconds). During the online resource adjustment, we adopt the resource provisioning plan after *five* adjustments of GPU resources for GSLICE$^+$. Similarly, we select the resource provisioning plan *after* dealing with prediction errors for *iGniter*. As illustrated in Fig. 14, *iGniter* guarantees the P99 inference latency of all 12 inference workloads within their latency SLOs, while saving up to 25% of *hourly* monetary cost compared with gpu-lets$^+$.

How can *iGniter* guarantee performance SLOs? As shown in Fig. 14, FFD$^+$ *first* makes 10 out of 12 workloads violate performance SLOs because it does not consider the interference of co-located workloads. In contrast, *iGniter* provisions an additional 25% of GPU resources (*i.e.*, GPU6) and adequately places workloads on GPUs to *proactively* eliminate SLO violations caused by the interference. *Second*, though gpu-lets$^+$ provisions the largest amount of GPU resources, there still exist 3 workloads (*i.e.*, $W7$, $W8$, $W12$) violating performance SLOs. This is because gpu-lets$^+$ does not model the interference on request throughputs and it

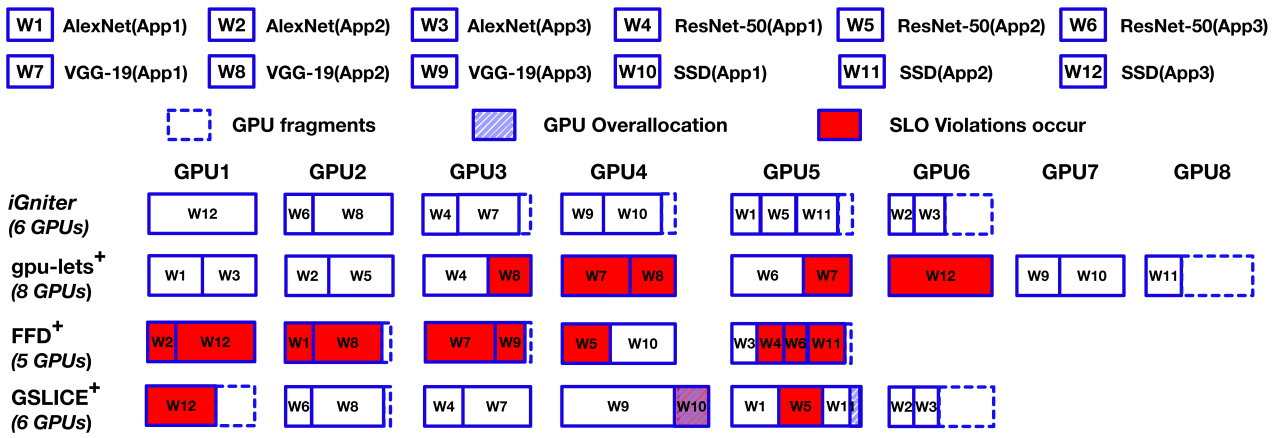

Fig. 14: Comparison of GPU resource provisioning plans for the 12 workloads (*i.e.*, $W1, \cdots, W12$). *iGniter*, gpu-lets$^+$, FFD$^+$, and GSLICE$^+$ provision 6, 8, 5, and 6 GPU devices (p3.2xlarge instances), which achieve \$18.36, \$24.48, \$15.3, and \$18.36 monetary cost per hour, respectively. In addition, the four GPU resource provisioning strategies bring 0, 3, 10, and 3 SLO violations, respectively.

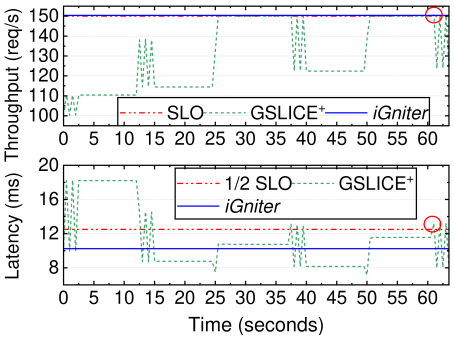

Fig. 15: Comparison of the inference latency and request throughput of $W10$ over time under the GSLICE$^+$ and *iGniter* strategies.

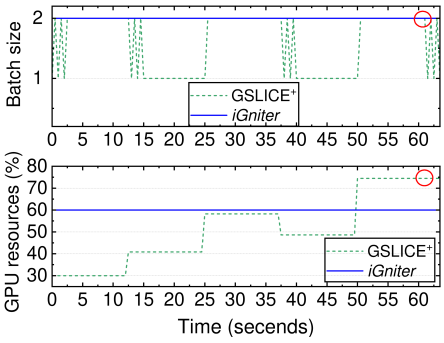

Fig. 16: Comparison of the allocated GPU resources and batch sizes for $W10$ over time under the GSLICE$^+$ and *iGniter* strategies.

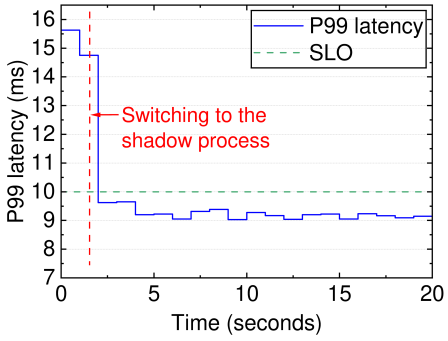

Fig. 17: P99 inference latency of $W1$ (*i.e.*, App1 of AlexNet) over time when *iGniter* handles SLO violations.

simply uses the profiled throughput when the workload is running alone. It inevitably makes workloads easily violate the expected throughput. *Third*, GSLICE$^+$ can cause 3 violations even using our workload placement plan. This is because the *interference-unaware* strategy (*i.e.*, GSLICE$^+$) *separately* adjusts allocated GPU resources and batch size according to a fixed tuning threshold (*e.g.*, 10%), which can make the inference performance *oscillate frequently* around SLOs. We take $W10$ (co-located with $W9$ on GPU4) as an example. As shown in Fig. 15, the average inference latency (*i.e.*, 10.7 ms) is lower than the $\frac{1}{2}$SLO (*i.e.*, 12.5 ms) exceeding the tuning threshold during 25.5 – 37.5 seconds. It then triggers GSLICE$^+$ to reduce the allocated GPU resources, which makes SSD violate the expected throughput (150 req/s). Moreover, GSLICE$^+$ adjusts the GPU resources of $W9$ to 100% at the 51-th second without considering $W10$, and the resources are successfully allocated to $W9$ at the 61-th second (*i.e.*, the red circle in Fig. 15 and Fig. 16). In such a case, the overallocation of GPU resources occurs, which brings SLO violations to both $W9$ and $W10$. In contrast, *iGniter* leverages our analytical inference performance model to *proactively* provision an adequate amount of GPU resources and to configure an appropriate batch size when launching inference workloads on GPUs.

Can *iGniter* deal with the performance prediction errors? The prediction error handling mechanism in *iGniter* further guarantees performance SLOs. In our experiments, such a mechanism only triggers two times (*i.e.*, two prediction errors occur). To illustrate how it works, we take $W1$ co-located with $W5$ and $W11$ on GPU5 as an example. As depicted in Fig. 17, the P99 latency of $W1$ at the first second is 15.6 ms which is higher than the latency SLO (*i.e.*, 10 ms) due to the prediction error. In the next 0.5 seconds, *iGniter* collects the request latency data and judges whether it violates the SLO. If an SLO violation still occurs, *iGniter* switches such an SLO-violated inference workload to the *activated* shadow Triton process at the 1.5-th second. After that, the P99 latency of $W1$ can be guaranteed within the SLO. As we have pre-launched the *shadow* Triton process as elaborated in Sec. 4.2, *iGniter* does not require spending 10 seconds in launching a new Triton process as in GSLICE$^+$.

How can *iGniter* save the monetary cost? As the *hourly* monetary cost is proportional to the number of provisioned GPU instances, we simply compare the *allocated GPU resources* of *iGniter* with that of GSLICE$^+$, FFD$^+$, and gpu-lets$^+$. As shown in Fig. 18, we observe that the GPU resources allocated by gpu-lets$^+$ for each workload are larger or equal to *iGniter*. This is mainly due to the following facts: *First*, taking $W4$ (*i.e.*, App1 of ResNet-50) as an example, gpu-lets$^+$ provisions 60% of GPU resources (*i.e.*, the most-efficient amount of GPU resources) and then sets the batch size as 2 to maximize its throughput. In contrast, *iGniter* sets an *appropriate* batch size as 4 and then provisions 32.5% of GPU resources to just meet its performance SLOs. *Second*, gpu-lets$^+$ only allows two co-located inference workloads on a GPU device, while *iGniter* allows multiple (more than

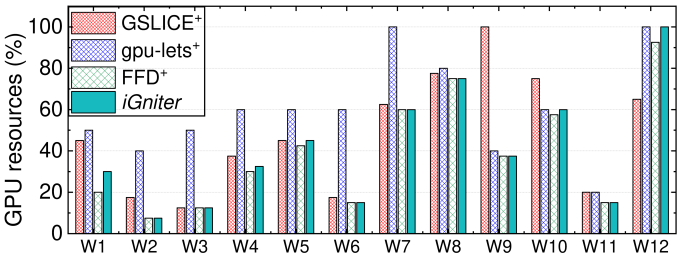

Fig. 18: Comparison of allocated GPU resources for the 12 workloads (*i.e.,* $W1, \cdots, W12$) achieved by the gpu-lets$^+$, FFD$^+$, GSLICE$^+$, and *iGniter* strategies.

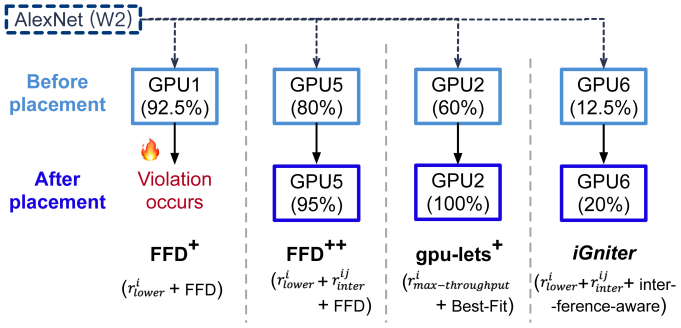

Fig. 19: Comparison of the inference workload (*i.e.,* App2 of AlexNet) placement decisions achieved by the FFD$^+$, gpu-lets$^+$, FFD^{++} (*i.e.,* FFD$^+$ using `alloc_gpus`, Alg. 2), and *iGniter* resource provisioning strategies.

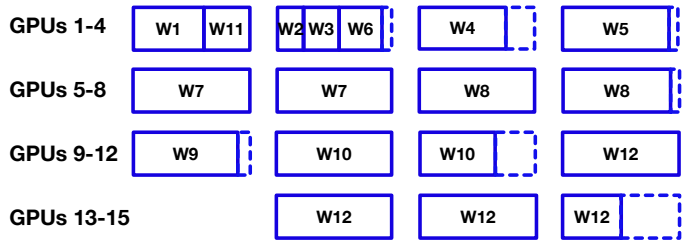

Fig. 20: GPU resource provisioning plans achieved by *iGniter* for the 12 workloads in a cluster of 15 g4dn.xlarge instances without any SLO violations, resulting in $7.89 monetary cost per hour.

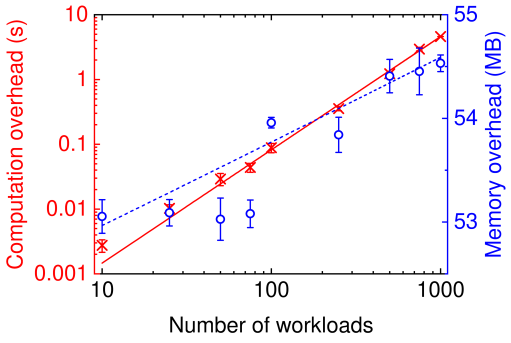

Fig. 21: Computation and memory overhead of *iGniter* by varying the number of DNN inference workloads from 10 to 1,000.

2) workloads concurrently executed. *Third,* gpu-lets$^+$ allows only *five* choices (*i.e.,* 20%, 40%, 50%, 60%, 80%) of GPU resources allocated to inference workloads, while *iGniter* can allocate workloads with an amount of GPU resources with a *fine-grained* GPU allocation unit (*i.e.,* 2.5%). For example, gpu-lets$^+$ and *iGniter* provision $W9$ with 40% and 37.5% of GPU resources, respectively. In addition, though GSLICE$^+$ uses our workload placement plan, it provisions more or equal amounts of GPU resources than *iGniter* for all workloads except $W12$ which violates its latency SLO. This is because GSLICE$^+$ does not reduce its allocated GPU resources, as long as an inference workload meets its performance SLOs and the tuning threshold. FFD$^+$ provisions less or equal amounts of GPU resources than *iGniter* as it always allocates the lower bound (r^i_{lower}) of GPU resources to inference workloads.

How can *iGniter* place inference workloads on GPUs? The *inference workload placer* elaborated in Sec. 4.2 in *iGniter* further reduces the amount of allocated GPU resources. As shown in Fig. 19, FFD$^+$ places $W2$ (*i.e.,* App2 of AlexNet) onto GPU1 according to the lower bound of GPU resources (*i.e.,* r^i_{lower}) which inevitably causes SLO violations due to the overlooked performance interference. FFD^{++} places such a workload onto GPU5 with 15% of GPU resources according to the first-fit GPU that still has an amount (*i.e.,* $r^i_{lower} + r^{ij}_{inter}$ which is calculated by Alg. 2) of GPU resources. As the most-efficient amount of GPU resources (*i.e.,* $r^i_{max_throughput}$) for App2 of AlexNet is 40%, gpu-lets$^+$ places $W2$ onto GPU2 which is selected as the best-fit GPU device. In general, gpu-lets$^+$ allocates more GPU resources than the other strategies as it mainly focuses on improving the inference throughput. In contrast, *iGniter* places $W2$ onto GPU6 with the least amount of GPU resources (7.5%) while guaranteeing the latency SLOs of all workloads. This is because *iGniter* greedily places the inference workload onto the GPU with the least performance interference and allocates GPU resources that *just meet* performance SLOs.

Can *iGniter* adapt to the heterogeneous cluster? To obtain complementary insights, we extend our GPU cluster by adding 20 g4dn.xlarge instances, each equipped with 1 NVIDIA T4 GPU card, 4 vCPUs, and 16 GB memory. After obtaining the *hardware-specific* coefficients and a part of *workload-specific* coefficients on the g4dn.xlarge instance, Alg. 1 can identify the appropriate GPU resource provisioning plan as illustrated in Fig. 20. As the NVIDIA V100 GPU device is equipped with 2× GPU computing resources and 3× memory bandwidth resources compared with the NVIDIA T4 GPU device, *iGniter* provisions 15 g4dn.xlarge instances (T4) while 6 p3.2xlarge instances (V100) for the 12 workloads, respectively. In particular, *iGniter* provisions 2+ g4dn.xlarge instances for $W7$, $W8$, $W10$, and $W12$ to meet their performance SLOs. Finally, as the hourly monetary cost (*i.e.,* $7.89) on g4dn.xlarge instances is much less than that (*i.e.,* $18.36) on p3.2xlarge instances, *iGniter* considers g4dn.xlarge as the most cost-efficient type of instances and it adopts the resource provisioning plan in Fig. 14 for serving the 12 inference workloads.

5.4 Runtime Overhead of *iGniter*

We evaluate the runtime overhead of *iGniter* in terms of the profiling overhead of DNN inference workloads, and the computation time and memory consumption of *iGniter* resource provisioning strategy (*i.e.,* Alg. 1). Specifically, we launch a p3.2xlarge EC2 instance to profile the *workload-specific* coefficients *only once* for each inference workload. The profiling time of AlexNet [23], ResNet-50 [24], VGG-19 [25], and SSD [35] models are 231, 247, 240, and 237

TABLE 4: Comparison of predictable DNN inference systems on GPUs.

Strategies	Interference awareness	Spatial sharing	Profiling overhead	Workload placement	Batching
Clipper [16]	✗	✗	N/A	✗	✓
BatchDVFS [37]	✗	✗	lightweight	✗	✓
Nexus [9]	✗	✗	lightweight	✓	✓
Clockwork [12]	✗	✗	lightweight	✓	✓
Morphling [38]	✗	✗	lightweight	✗	✓
Cocktail [11]	✗	✗	lightweight	✗	✗
INFaaS [17]	✓	✗	lightweight	✓	✓
Scrooge [10]	✗	multiple	heavy	✓	✓
MIG-serving [39]	✗	multiple	heavy	✓	✓
INFless [40]	✗	multiple	lightweight	✓	✓
GSLICE [13]	✗	multiple	N/A	✗	✓
gpu-lets [18]	✓	2	heavy	✓	✓
iGniter	✓	multiple	lightweight	✓	✓

seconds, respectively. In addition, we profile the *hardware-specific* coefficients with VGG-19 *only once for a given GPU type* and the profiling time is merely 229 seconds. The experiment results above show that the profiling overhead of inference workloads is within several (around 4) minutes, which is far less than the runtime overhead of gpu-lets [18] (*i.e.*, over several hours) in our experiments.

After obtaining the performance model coefficients, we proceed to run our *iGniter* strategy in Alg. 1 on a p3.2xlarge EC2 instance. The computation overhead and memory consumption of *iGniter* are negligible, which are merely 3.64 milliseconds and 53.17 MB, respectively. As the number of workloads is increased to $1,000$ shown in Fig. 21, the computation overhead is still within 4.61 seconds and the memory overhead is less than 55 MB. This is because the computation time and memory consumption of Alg. 1 are quadratic to and linear to the number of DNN inference workloads, respectively, as analyzed in Sec. 4.1. As a result, the runtime overhead of our *iGniter* strategy can be acceptable in practice.

6 Related Work

Achieving Predictable DNN Inference on GPUs. As summarized in Table 4, there have been a number of works on guaranteeing DNN inference performance SLOs on GPUs. In the scenario of ***disabling GPU sharing*** (*i.e.*, a GPU serves one DNN inference at a time), Clipper [16] proposes caching, adaptive batch size, and dynamic model selection techniques to achieve low-latency and high-throughput DNN inference. BatchDVFS [37] combines adaptive batching with the DVFS technique to maximize the inference request throughput while guaranteeing the power caps.

In the scenario of ***temporal sharing*** of GPUs, Nexus [9] proposes batching-aware scheduling based on Clipper [16] to improve the GPU utilization. Clockwork [12] designs fine-grained request-level scheduling to order user requests based on their latency SLOs. Morphling [38] utilizes meta-learning to quickly configure the batch size, CPU cores, GPU memory, GPU timeshare, and GPU type for each inference workload. While sharing the adaptive batching and workload placement techniques with the prior works above, *iGniter* aims to *cost-efficiently* guarantee the performance SLOs based on GPU *spatial sharing*, instead of maximizing the request throughput of inference workloads. To further reduce the monetary cost of DNN inference, two more recent works (*i.e.*, Cocktail [11], INFaaS [17]) design the heterogeneous instance/accelerator selection, resource autoscaling, and dynamic model-variants selection techniques for cost-effective resource provisioning. These techniques above can be incorporated into *iGniter* to further save the inference budget. In addition, our *SM-level* resource scaling in *iGniter* (*i.e.*, r_{unit} in Algorithm 2) is more *fine-grained* than the *device-level* resource scaling in Cocktail and INFaaS.

In the scenario of ***spatial sharing*** of GPUs, Scrooge [10] leverages the CUDA streams and batching techniques to pack DNN inference on VMs to ensure the performance SLOs of media applications. Using the latest multi-instance GPU (MIG) [41] featured A100 GPUs, MIG-serving [39] optimizes a set of GPU partitions and DNN inference deployments to meet performance SLOs. To further maximize the request throughput, INFless [40] adopts batching and heterogeneous CPU-GPU resources for DNN inference in the serverless platform. GSLICE [13] and gpu-lets [18] *separately* adjust the batch size and allocated GPU resources for inference workloads. However, the prior works above are mostly oblivious to performance interference and thus they tend to cause long-tail latency due to the severe GPU resource contention. In contrast. *iGniter* proactively considers (*i.e.*, minimizes) the performance interference among co-located inference workloads and *jointly* optimizes the GPU resource allocation and batch size configuration.

Modeling Performance Interference in Clouds. There have been prior works on modeling the performance interference [42] and hardware heterogeneity [43] of cloud CPU instances. For instance, VELTAIR [44] builds a simple linear interference model using L3 cache miss rate and L3 access statistics. To particularly model the performance interference among co-located VMs based on ***temporal sharing*** of GPUs, Xu *et al.* [45] build a random forest regression model with a set of factors such as GPU/memory utilization and the average kernel length. As DNN training and inference workloads become prevailing in the cloud [46], Horus [47] leverages GPU utilization to estimate the performance interference among co-located *DNN training* jobs through fitting a quadratic function, while *iGniter* focuses on modeling the *DNN inference* performance using a set of easily-accessible GPU system and workload metrics.

Different from the interference above caused by the *context switching of temporal sharing* of GPUs, NVIDIA MPS allows DNN inference to ***spatially share*** GPU resources. To model the interference caused by GPU resource contention, Prophet [48] characterizes the contention of GPU processing elements and DRAM bandwidth [49] as well as PCIe bandwidth in the *default* mode of MPS [50]. Based on the *MPS with limited GPU resources*, gpu-lets [18] builds a linear regression model using the L2 cache and DRAM bandwidth utilization to predict the latency increases for only *two* inference workloads. However, it requires profiling a number (*e.g.*, thousands) of possible workload configurations, which brings heavy runtime overhead. Different from the models above, *iGniter* builds an analytical model to predict the interference among multiple (*i.e.*, more than 2) inference

workloads by a *lightweight* workload profiling with a limited number (*i.e.*, 11) of configurations. Moreover, our *iGniter* model *comprehensively* considers the severe contention of GPU scheduler, L2 GPU cache space, and GPU power consumption among co-located inference workloads.

7 Conclusion and Future Work

This paper presents the design and implementation of *iGniter*, an interference-aware GPU resource provisioning framework for achieving predictable DNN inference in the cloud. By leveraging the key system and workload metrics, we first devise a lightweight analytical performance model to capture the performance interference of inference workloads co-located on GPUs. Such a performance model further guides the design of a cost-efficient GPU resource provisioning strategy in *iGniter*. It jointly optimizes the GPU resource allocation and batch size configuration to greedily minimize the performance interference of DNN inference workloads. Extensive prototype experiments on Amazon EC2 demonstrate that *iGniter* can guarantee the performance SLOs of cloud-based DNN inference workloads, while saving the monetary cost by up to 25% compared with the state-of-the-art resource provisioning strategies.

We plan to extend *iGniter* in the following directions: (1) provisioning DNN inference workloads with multiple types of GPU hardware or accelerators, (2) allocating multiple GPU instances to a DNN inference workload with an extremely large request arrival rate, (3) negotiating the tradeoff between minimizing the monetary cost and maximizing the performance of DNN inference workloads, (4) deploying a dynamic temporal and spatial GPU sharing strategy for time-varying request arrival rates, and (5) examining the effectiveness of *iGniter* in the mixed deployment scenario of DNN inference and training workloads.

Acknowledgments

The corresponding author is Fei Xu (fxu@cs.ecnu.edu.cn). This work was supported in part by the NSFC under grant No.61972158, in part by the Science and Technology Commission of Shanghai Municipality under grant No.20511102802 and No.18DZ2270800. Li Chen's work was supported by grants from BoRSF under Contract Numbers LEQSF(2019-22)-RD-A-21 and LEQSF(2021-22)-RD-D-07, and NSF under Award Number OIA-2019511. Zhi Zhou's work was supported in part by the National Key Research & Development (R&D) Plan under grant 2022YFB4500704, in part by NSFC under Grant 62172454. Fangming Liu's work was supported in part by The Major Key Project of PCL (PCL2022A05).

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] P. Jain, X. Mo, A. Jain, H. Subbaraj, R. S. Durrani, A. Tumanov, J. Gonzalez, and I. Stoica, "Dynamic Space-Time Scheduling for GPU Inference," in *Proc. of NeurIPS*, Dec. 2018, pp. 1–8.

[3] NVIDIA. (2019, May) Intel Inference NVIDIA GPUs. [Online]. Available: https://blogs.nvidia.com/blog/2019/05/21/intel-inference-nvidia-gpus/

[4] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep Interest Evolution Network for Click-Through Rate Prediction," in *Proc. of AAAI*, vol. 33, no. 01, 2019, pp. 5941–5948.

[5] NVIDIA. (2018, May) JD AI Video Inferencing. [Online]. Available: https://blogs.nvidia.com/blog/2018/02/13/jd-ai-video-inferencing/

[6] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das *et al.*, "Elastic Machine Learning Algorithms in Amazon Sagemaker," in *Proc. of ACM SIGMOD*, 2020, pp. 731–737.

[7] Google Cloud. (2021, Nov.) Vertex AI. [Online]. Available: https://cloud.google.com/vertex-ai

[8] Omdia. (2021, Aug.) NVIDIA Maintains Dominant Position In 2020 Market. [Online]. Available: https://omdia.tech.informa.com/pr/2021-aug/nvidia-maintains-dominant-position-in-2020-market-for-ai-processors-for-cloud-and-data-center

[9] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philiposei, A. Krishnamurthy, and R. Sundaram, "Nexus: a GPU Cluster Engine for Accelerating DNN-based Video Analysis," in *Proc. of ACM SOSP*, Oct. 2019, pp. 322–337.

[10] Y. Hu, R. Ghosh, and R. Govindan, "Scrooge: A Cost-Effective Deep Learning Inference System," in *Proc. of ACM SOCC*, Nov. 2021, pp. 624–638.

[11] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, M. T. Kandemir, and C. R. Das, "Cocktail: A Multidimensional Optimization for Model Serving in Cloud," in *Proc. of USENIX NSDI*, Apr. 2022, pp. 1–17.

[12] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like Clockwork: Performance Predictability from the Bottom Up," in *Proc. of USENIX OSDI*, Nov. 2020, pp. 443–462.

[13] A. Dhakal, S. G. Kulkarni, and K. K. Ramakrishnan, "GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform," in *Proc. of ACM SOCC*, Oct. 2020, pp. 492–506.

[14] NVIDIA. (2021, Jun.) NVIDIA Multi-Process Service. [Online]. Available: https://docs.nvidia.com/deploy/mps

[15] W. Zhang, Q. Chen, N. Zheng, W. Cui, K. Fu, and M. Guo, "Towards QoS-awareness and Improved Utilization of Spatial Multitasking GPUs," *IEEE Transactions on Computers*, pp. 1–14, 2021.

[16] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency Online Prediction Serving System," in *Proc. of USENIX NSDI*, Mar. 2017, pp. 613–627.

[17] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated Model-less Inference Serving," in *Proc. of USENIX ATC*, Jul. 2021, pp. 397–411.

[18] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing," in *Proc. of USENIX ATC*, Jul. 2022, pp. 199–216.

[19] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2014.

[20] NVIDIA. (2021, Nov.) NVIDIA Triton Inference Server. [Online]. Available: https://github.com/triton-inference-server/server

[21] S. Kim, S. Oh, and Y. Yi, "Minimizing GPU Kernel Launch Overhead in Deep Learning Inference on Mobile GPUs," in *Proc. of HotMobile*, Feb. 2021, pp. 57–63.

[22] Amazon. (2021, Nov.) Amazon Elastic Compute Cloud (Amazon EC2). [Online]. Available: https://aws.amazon.com/ec2/

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of IEEE CVPR*, Jun. 2016, pp. 770–778.

[25] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. of ICLR*, May 2015, pp. 1–14.

[26] H. Vanholder, "Efficient Inference with TensorRT," in *Proc. of GPU Technology Conference*, vol. 1, 2016, pp. 2–2.

[27] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs," in *Proc. of IEEE RTAS*, Jul. 2019, pp. 29–41.

[28] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU," in *Proc. of ICPP*, Oct. 2013, pp. 826–833.
[29] NVIDIA. (2021, Nov.) NVIDIA Nsight Systems. [Online]. Available: https://developer.nvidia.com/nsight-systems
[30] ——. (2019, May) NVIDIA System Management Interface. [Online]. Available: https://blogs.nvidia.com/blog/2019/05/21/intel-inference-nvidia-gpus/
[31] ——. (2021, Nov.) NVIDIA Nsight Compute. [Online]. Available: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html
[32] H. Abdi *et al.*, "The Method of Least Squares," *Encyclopedia of measurement and statistics*, vol. 1, pp. 530–532, 2007.
[33] D. S. Johnson, "Near-Optimal Bin Packing Algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1973.
[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. of IEEE CVPR*, Jun. 2009, pp. 248–255.
[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *Proc. of ECCV*, Oct. 2016, pp. 21–37.
[36] M. Everingham and J. Winn, "The Pascal Visual Object Classes Challenge 2012 (VOC2012) Development Kit," Tech. Rep., May 2012.
[37] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated Batching and DVFS for DNN Inference on GPU Accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2496–2508, 2022.
[38] L. Wang, L. Yang, Y. Yu, W. Wang, B. Li, X. Sun, J. He, and L. Zhang, "Morphling: Fast, near-optimal auto-configuration for cloud-native model serving," in *Proc. of ACM SOCC*, Nov. 2021, pp. 639–653.
[39] C. Tan, Z. Li, J. Zhang, Y. Cao, S. Qi, Z. Liu, Y. Zhu, and C. Guo, "Serving DNN Models with Multi-Instance GPUs: A Case of the Reconfigurable Machine Scheduling Problem," *arXiv preprint arXiv:2109.11067*, 2021.
[40] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "INFless: A Native Serverless System for Low-Latency, High-Throughput Inference," in *Proc. of ACM ASPLOS*, Feb. 2022, pp. 768–781.
[41] NVIDIA. (2021, Jun.) NVIDIA Multi-Instance GPU User Guide. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/
[42] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 3012–3025, 2014.
[43] F. Xu, F. Liu, and H. Jin, "Heterogeneity and Interference-Aware Virtual Machine Provisioning for Predictable Performance in the Cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2470–2483, 2016.
[44] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "VELTAIR: Towards High-Performance Multi-tenant Deep Learning Services via Adaptive Compilation and Scheduling," in *Proc. of ACM ASPLOS*, Feb. 2022, pp. 388–401.
[45] X. Xu, N. Zhang, M. Cui, M. He, and R. Surana, "Characterization and Prediction of Performance Interference on Mediated Pass through GPUs for Interference-Aware Scheduler," in *Proc. of USENIX HotCloud*, Jul. 2019, pp. 1–8.
[46] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training," in *Proc. of ICPP*, Aug. 2019, pp. 1–11.
[47] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 88–100, 2021.
[48] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers," in *Proc. of ACM ASPLOS*, Apr. 2017, pp. 17–32.
[49] W. Zhang, Q. Chen, K. Fu, N. Zheng, Z. Huang, J. Leng, and M. Guo, "Astraea: Towards QoS-Aware and Resource-Efficient Multi-Stage GPU Services," in *Proc. of ACM ASPLOS*, Feb. 2022, pp. 570–582.
[50] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 681–696, 2016.

Fei Xu received the B.S., M.E., and Ph.D. degrees in 2007, 2009, and 2014, respectively, all from the Huazhong University of Science and Technology (HUST), Wuhan, China. He received Outstanding Doctoral Dissertation Award in Hubei province, China, and ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award in 2015. He is currently an associate professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests include cloud computing and datacenter, virtualization technology, and distributed systems.

Jianian Xu received his B.S. degree in Polymer Materials and Engineering from Qingdao University of Science and Technology in 2019. He is currently working toward the master's degree in the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests focus on cloud computing and distributed machine learning systems.

Jiabin Chen received his B.S. degree in Optoelectronic Information Science and Engineering from Harbin Institute of Technology, Weihai in 2019. He is currently working toward the master's degree in the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests focus on cloud computing and distributed machine learning systems.

Li Chen received the BEngr degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2012 and the MASc degree from the Department of Electrical and Computer Engineering, University of Toronto, in 2014 and the PhD degree in computer science and engineering from the Department of Electrical and Computer Engineering, University of Toronto, in 2018. She is currently an assistant professor with the Department of Computer Science, School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, USA. Her research interests include big data analytics systems, cloud computing, datacenter networking, and resource allocation.

Ruitao Shang received her B.S. degree in Computer Science from East China Normal University (ECNU) in 2020. She is currently pursuing her MS degree in Computer Science in the School of Computer Science and Technology at ECNU. Her current research interests focus on cloud computing and distributed machine learning systems.

Zhi Zhou received the B.S., M.E., and Ph.D. degrees in 2012, 2014, and 2017, respectively, all from the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the School of Computer Science and Engineering at Sun Yat-sen University, Guangzhou, China. In 2016, he was a visiting scholar at University of Göttingen. He was nominated for the 2019 CCF Outstanding Doctoral Dissertation Award, the sole recipient of the 2018 ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award, and a recipient of the Best Paper Award of IEEE UIC 2018. His research interests include edge computing, cloud computing, and distributed systems.

Fangming Liu (S'08, M'11, SM'16) received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, the First Class Prize of Natural Science of Ministry of Education in China, as well as the Second Class Prize of National Natural Science Award in China.

APPENDIX

Proof. We first calculate the appropriate batch size b^i_{appr} that just meets the arrival rate. Specifically, we substitute Eq. (1) into Constraint (14), yielding $\frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback} \geq t^{ij}_{gpu}$ when an inference workload i is running on a GPU j. Given a batch size, the GPU execution latency increases as the amount of allocated GPU resources decreases. Accordingly, in order to minimize the amount of GPU resources, we set the GPU execution latency to the maximum value as

$$t^{ij}_{gpu} = \frac{T^i_{slo}}{2} - t^i_{load} - t^i_{feedback}. \tag{20}$$

By substituting Eq. (20), Eq. (2), and Eq. (3) into Constraint (13), we have $b^i \geq \frac{T^i_{slo} \cdot R^i \cdot B_{pcie}}{2 \cdot (B_{pcie} + \cdot R^i \cdot d^i_{load})}$. In addition, a larger batch size generally indicates a higher GPU execution latency given an amount of allocated GPU resources. Accordingly, we simply choose the *appropriate* batch size b^i_{appr} that *just meets* the arrival rate, which is given by

$$b^i_{appr} = \left\lceil \frac{T^i_{slo} \cdot R^i \cdot B_{pcie}}{2 \cdot (B_{pcie} + R^i \cdot d^i_{load})} \right\rceil.$$

In more detail, if we increase the batch size b^i_{appr}, the GPU resources allocated to the workload i requires increasing. Otherwise, if we reduce the batch size b^i_{appr}, it will violate Constraint (13) (*i.e.,* the request arrival rate cannot be guaranteed). Accordingly, we consider b^i_{appr} as the *appropriate* batch size for our optimization problem.

We next obtain the lower bound r^i_{lower} of GPU execution resources for each workload i as follows. By substituting b^i_{appr}, Eq. (1), Eq. (3), Eq. (4), Eq. (5), Eq. (6), Eq. (8), and Eq. (11) into Constraint (14), we calculate the amount of allocated resources r^{ij} on a GPU device as below,

$$\begin{aligned} r^{ij} &\geq \frac{k^i_1 \cdot (b^i_{appr})^2 + k^i_2 \cdot b^i_{appr} + k^i_3}{\left(\frac{T^i_{slo}}{2} - \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}}\right) \cdot \frac{f^j}{F} - k^i_5 - k^i_{sch} \cdot n^i_k} - k^i_4 \\ &\geq \frac{k^i_1 \cdot (b^i_{appr})^2 + k^i_2 \cdot b^i_{appr} + k^i_3}{\frac{T^i_{slo}}{2} - \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch} \cdot n^i_k} - k^i_4. \end{aligned}$$

As the GPU resources are allocated in units of r_{unit} which is set as 2.5% for NVIDIA V100 GPUs, the lower bound r^i_{lower} of GPU execution resources for each workload i can be calculated by

$$r^i_{lower} = \left\lceil \frac{\gamma^i}{\delta^i \cdot r_{unit}} - \frac{k^i_4}{r_{unit}} \right\rceil \cdot r_{unit},$$

where $\gamma^i = k^i_1 \cdot (b^i_{appr})^2 + k^i_2 \cdot b^i_{appr} + k^i_3$ and $\delta^i = \frac{T^i_{slo}}{2} - \frac{(d^i_{load} + d^i_{feedback}) \cdot b^i_{appr}}{B_{pcie}} - k^i_5 - k^i_{sch} \cdot n^i_k$. □