Conspirator: SmartNIC-Aided Control Plane for Distributed ML Workloads

Yunming Xiao, *Northwestern University;* Diman Zad Tootaghaj, Aditya Dhakal, Lianjie Cao, and Puneet Sharma, *Hewlett Packard Labs;* Aleksandar Kuzmanovic, *Northwestern University*

https://www.usenix.org/conference/atc24/presentation/xiao

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Conspirator: SmartNIC-Aided Control Plane for Distributed ML Workloads

Yunming Xiao[1], Diman Zad Tootaghaj[2], Aditya Dhakal[2], Lianjie Cao[2],
Puneet Sharma[2], Aleksandar Kuzmanovic[1]
[1] *Northwestern University,* [2] *Hewlett Packard Labs*

Abstract

Modern machine learning (ML) workloads heavily depend on distributing tasks across clusters of server CPUs and specialized accelerators, such as GPUs and TPUs, to achieve optimal performance. Nonetheless, prior research has highlighted the inefficient utilization of computing resources in distributed ML, leading to suboptimal performance. This inefficiency primarily stems from CPU bottlenecks and suboptimal accelerator scheduling. Although numerous proposals have been put forward to address these issues individually, none have effectively tackled both inefficiencies simultaneously. In this paper, we introduce Conspirator, an innovative control plane design aimed at alleviating both bottlenecks by harnessing the enhanced computing capabilities of SmartNICs. Following the evolving role of SmartNICs, which have transitioned from their initial function of standard networking task offloading to serving as programmable connectors between disaggregated computing resources, Conspirator facilitates efficient data transfer without the involvement of host CPUs and hence circumvents the potential bottlenecks there. Conspirator further integrates a novel scheduling algorithm that takes into consideration of the heterogeneity of accelerators and adapts to changing workload dynamics, enabling the flexibility to mitigate the second bottleneck. Our evaluation demonstrates that Conspirator may provide a 15% end-to-end completion time reduction compared to RDMA-based alternatives while being 17% more cost-effective and 44% more power-efficient. Our proposed scheduler also helps to save 33% GPU hours compared to naive GPU-sharing schedulers by making close-to-optimal decisions while taking much less time than the optimal NP-Hard scheduler.

1 Introduction

Machine learning (ML) has revolutionized a multitude of applications across various facets of our daily lives [33, 44, 45, 61, 69, 83]. This pervasive adoption of ML technologies has in turn catalyzed rapid advancements in the field. A notable trend in ML development is the increasing reliance on larger and more intricate datasets, coupled with the utilization of ever-more complex models. This shift necessitates substantial computational power and extends the time required for both training and inference processes. As a result, the need for efficient ML training and inference has become paramount to ensure the swift deployment of cutting-edge ML applications, accommodating the growing demands of diverse industries.

Numerous solutions have been put forth to address these challenges. One straightforward concept involves distributing ML workloads across a cluster of machines, enabling the completion of computations in a distributed manner. Indeed, distributed ML has evolved into a fundamental approach in industrial ML practices [51, 53, 72]. However, it is crucial to ensure efficient communication in distributed ML setups. It has become evident that relying on the traditional TCP/IP stack for communication occupies too many CPU resources and is prohibitively slow in the context of distributed ML workloads [59, 67, 77]. As a result, alternative approaches that bypass the kernel, such as the Data Plane Development Kit (DPDK) [6], have been extensively explored. Taking this a step further, Remote Direct Memory Access (RDMA) [43] technology offers better performance by facilitating data transmission without the direct involvement of the host CPU.

Besides addressing the communication bottleneck, the host CPU has been proven to be ill-suited for the majority of computations required in ML training and inference [77]. Instead, accelerators like GPUs or TPUs have demonstrated significantly superior performance compared to CPUs. But these accelerators vary in terms of technologies and specifications, making them suitable for different ML workloads [29, 45, 57, 66]. Consequently, optimizing accelerator scheduling plays a pivotal role in enhancing (distributed) ML workloads. In particular, techniques such as NVIDIA Multi-Instance GPU (MIG) [21] that enable accelerator-sharing provide the means for fine-grained resource allocation, ultimately leading to improved performance and efficiency.

Nevertheless, existing approaches tend to concentrate solely on addressing one of these bottlenecks, often overlooking the intricate interplay between them. Notably, optimizations proposed for one bottleneck may inadvertently conflict with those intended for the other. For instance, RDMA-based optimizations typically prioritize the elimination of any data communication barriers, e.g., GPUDirect [19] allows direct read or write access to dedicated GPU memories, which can run counter to the needs of accelerator schedulers. The latter often requires a decision-making process to efficiently route data to its final and optimal destination. One potential solution to reconcile these conflicting requirements is the implementation of an omniscient and omnipotent central scheduler, which would have real-time visibility into the status of every accelerator and could manage each data transfer request accordingly. However, this approach is impractical in reality because of two reasons: (i) network latency is inevitable, and (ii) most shared cluster environments have diverse users with distinct requirements for data security and privacy and consequently demands for data isolation measures.

In this paper, we emphasize the equal significance of both bottlenecks and undertake a comprehensive investigation to address them effectively. We introduce Conspirator, a system designed to tackle both bottlenecks by harnessing the capabilities of SmartNICs. Specifically, we leverage the SmartNIC to mitigate the data communication and accelerator scheduling bottlenecks.

On the one hand, Conspirator leverages RDMA and DMA to facilitate efficient data transfer between remote hosts and local accelerators, minimizing barriers to the communication process. On the other hand, Conspirator obtains real-time visibility into the status of local accelerators and thus enables optimal accelerator scheduling. This holistic approach seeks to achieve efficient communication and optimal resource allocation simultaneously, addressing the dual challenges of distributed ML workloads.

We have implemented a prototype of Conspirator based on NVIDIA's BlueField-3 SmartNICs and A100 GPUs. Our evaluation reveals Conspirator's superior performance across various metrics, including latency, cost-effectiveness, power efficiency, and GPU resource utilization.

Specifically, Conspirator significantly surpasses TCP-based data transmission methods, achieving an improvement in the range of 2x to 4x. When compared to RDMA-based alternatives, Conspirator's strength is evident in local data transmission, leveraging SmartNIC to bypass host CPUs and enhance performance. This results in a local data transmission improvement between 8% and 50%, which varies based on CPU usage. Although the end-to-end latency reduction is capped at 15% due to GPU ML inference latencies where Conspirator aligns with other options, it still presents meaningful enhancements. Furthermore, Conspirator excels in terms of cost-effectiveness, offering a 17% improvement, and power efficiency, with an increase of 44%.

Last, when comparing against GPUDirect-based solutions that directly transfer data to accelerator memories without any barriers, Conspirator only introduces minimal overhead because it shortly buffers data at the SmartNIC, where such delay allows Conspirator to make close-to-optimal scheduling decisions, resulting in substantial savings of over 30% in total consumed GPU hours.

2 Background

2.1 Distributed Machine Learning Workloads

ML has emerged as a prominent trend, with its applications quickly prevailing in various aspects of everyday life and research fields [33,44,45,61,69,83]. Due to its reliance on large datasets and complex models, it demands significant computational power and extended processing time. Consequently, reducing the duration of both ML training and inference is crucial to adapt it for diverse routine applications. To address this, ML experts have effectively harnessed accelerators like GPUs and TPUs [49], which excel in parallel processing capabilities. In addition, distributed computing for ML has become a common practice, providing a solid foundation for optimizing its performance [51,53,72].

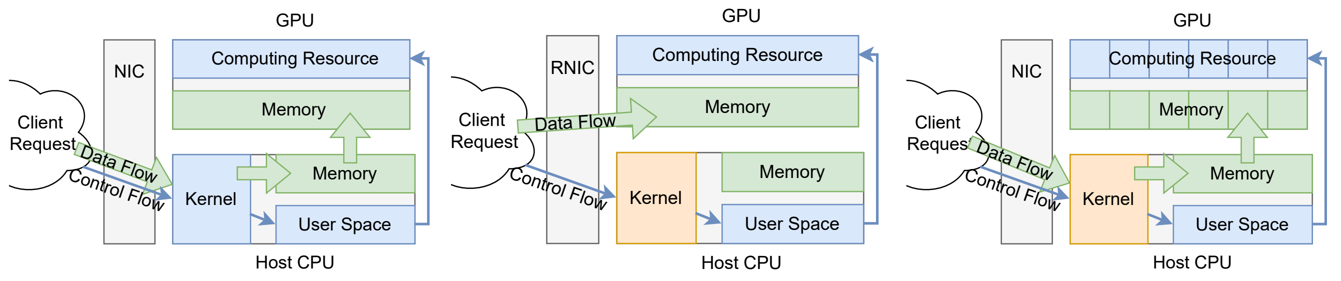
Nevertheless, previous studies have identified inefficient uses of computing resources for distributed ML, resulting in suboptimal performance. These inefficiencies primarily fall into two categories: (i) bottlenecks on CPUs [59,67,77], and (ii) sub-optimal scheduling of accelerators [30,46,77]. Note that since TPUs are not as widely accessible to the average user, this paper focuses on GPUs as the primary ML accelerator.

2.2 Bottleneck on CPUs

The bottleneck on the CPUs is one major reason for the inefficient resource utilization. This arises from various tasks, including data pre-processing, network stack overhead, aided computation for ML, background CPU activities, and more. Among these, data pre-processing often exerts the heaviest pressure on the CPU [59,82]. This has led to proposals advocating for the delegation of data pre-processing to dedicated servers [82].

Apart from the data pre-processing, the high communication costs of distributed ML tasks also contribute heavily to CPU contention. For training tasks, distributed ML necessitates transferring significant amounts of data to feed the models during the shuffling phase; for inference tasks, the server may need to handle hundreds of queries per second, with each query involving high-precision pictures that can range up to megabytes in size.

The traditional server architecture with an accelerator is depicted in Figure 1(a). In this architecture, incoming requests traverse the TCP/IP network stack at kernel and the data is stored in the host memory. When dealing with distributed ML workloads that offload computation to the accelerator



(a) Traditional. (b) CPU-bypass Design. (c) SMIF.

Figure 1: Illustrations of alternative designs.

instead of the host CPU, the data must then be copied to the accelerator memory before being utilized by the accelerator's computing units.

Regrettably, previous studies have revealed that the standard server architecture falls short when handling distributed ML workloads with large volumes of requests. For instance, it has been demonstrated that more than 20% of distributed ML tasks spend over 33% of their total time on data ingestion rather than executing meaningful computations [59, 67, 77].

Even worse, CPU contention exacerbates the aforementioned scenarios, as research has shown that high CPU utilization can lead to slowdowns in ML workloads [77]. Such high CPU utilization can be attributed to two factors: (*i*) data processing or simulation tasks involved in the distributed ML pipeline and (*ii*) other CPU-intensive tasks coexisting in a multi-tenant cloud environment.

To address the bottleneck on CPUs and minimize their involvement, leveraging alternative network stacks and technologies becomes imperative. Rather than relying on the TCP/IP stack, which may prove inadequate, adopting lightweight network stacks can offer better performance. A good example of this is Data Plane Development Kit (DPDK) [6], which enables packet processing to bypass the complex Linux kernel, resulting in improved performance [37]. Furthermore, when coupled with in-network aggregation functions provided by programmable switches [67], performance gains can be further amplified.

RDMA takes a leap forward by enabling direct memory access from a remote server, completely bypassing the host CPU [43]. This capability extends to accelerators as well, with initiatives such as NVIDIA's GPUDirect [19] allowing direct read or write access to GPU memory. Figure 1(b) illustrates a typical architecture of distributed ML supported by GPUDirect, where the data is directly forwarded to the GPU memory while the host CPU is only responsible for program initialization, e.g., reserving GPU memory and starting GPU kernel programs when needed. By eliminating the need for host CPU involvement, these techniques effectively alleviate the bottleneck on CPUs [79]. Given the giant benefits from RDMA, it has been prevalently used in data centers [23] despite it requiring extra hardware – the RDMA-enabled NICs (RNICs).

However, the host-CPU bypassing architectures such as those enabled by RDMA and GPUDirect miss the aspect of the accelerator schedule, leading to sub-optimal scheduling as explained below.

2.3 Sub-optimal Accelerator Scheduling

Another key aspect of inefficient resource utilization is improper accelerator scheduling, which encompasses three specific sub-problems. First, the heterogeneity of accelerators is not adequately considered in distributed ML, leading to sub-optimal performance [25, 30, 77]. For instance, some ML tasks may need to run on specific GPUs, while the rest do not. Further, while the computational capacity of the GPUs are different, current ML platforms distribute the load without considering the heterogeneity in compute and memory capacity of the nodes and GPUs [30]. But even when such restrictions are not applicable and tasks can run on any available GPUs, their performance is heavily influenced by the varying hardware specifications and architectures of the GPUs, where the significant heterogeneity in GPUs can result in notable performance disparities.

Second, the presence of ML model heterogeneity also contributes to performance degradation. The growth of the ML community over the past decades has given rise to numerous prominent ML models, each characterized by distinct model architectures, e.g., various formations of deep neural networks [45], and countless model variants generated by various methods such as graph optimizers [29]. These diverse ML models yield different outcomes when applied to the same inputs, catering to different Service Level Objectives (SLOs). For instance, a compressed model may generate slightly less accurate results while consuming significantly fewer computing resources [57]. Consequently, a trade-off between performance, cost, and accuracy emerges, prompting numerous studies exploring this space [66].

Last, GPU sharing plays a crucial role in achieving desirable performance. In cloud environments, by default, each ML task is assigned one physical GPU instance [47, 58, 81], yet the task usually requires only a fraction of the GPU's capabilities. Figure 2 shows an experiment when we run tensorflow training on ResNet152 model on an A100, V100, and T4 GPUs

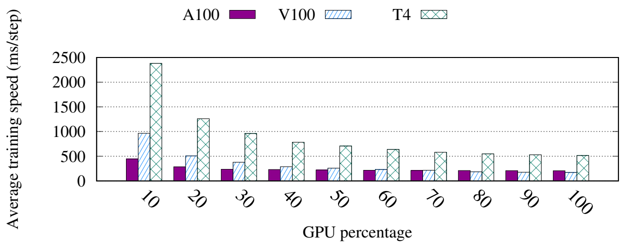


Figure 2: Average latency of ML training workload by using different shares of GPU on A100, V100, and T4 GPUs.

with different GPU percentages. The results show that the application does not utilize the whole GPU – we did not observe much performance improvement when assigning more than 50% of GPU resources. To address this inefficiency, several techniques have been proposed, including NVIDIA Multi-Process Service (MPS) [11], Multi-Instance GPU (MIG) [21], and many more [34, 77]. These techniques enable both time-multiplexing and space-multiplexing of multiple ML tasks on a single GPU device.

These GPU-sharing techniques operate at different levels to accommodate varying isolation requirements. For instance, MIG provides the strongest isolation between different shares of the physical GPU, ensuring data security at the cost of having only limited fixed sharing configurations. On the other hand, MPS allows flexible sharing without implementing GPU memory isolation. It is noteworthy, however, that interference can arise among different running applications or ML models when using GPU-sharing techniques without complete separation of GPU resources, such as in the case of NVIDIA MPS. This adds further complexity to the GPU-sharing decisions when optimal performance is desired. Overall, efforts have been made to incorporate *some yet not all* above-mentioned heterogeneity into the distributed ML workload scheduling [25, 30].

2.4 SmartNIC Comes Into Help

In this paper, we aim to resolve both inefficiencies – the bottlenecks on CPUs and the sub-optimal accelerator scheduling – in the distributed ML workloads. To this end, we find SmartNIC – the RNICs with programmable capabilities – as a solution to aid in overcoming these challenges.

Specifically, there are two types of SmartNICs: on-path and off-path. On-path SmartNICs incorporate programmable units, e.g., FPGAs, directly in the critical path of incoming packets to the NIC. Some example products of on-path SmartNIC include Innova [20] and Marvell [10]. This design enables high-speed processing capabilities. However, programming on-path SmartNICs can be challenging due to the reliance on low-level languages and complex hardware constraints. Previous studies have explored leveraging on-path SmartNICs to distribute the incoming traffic with some naive rules [54]. Yet, this is far from being able to solve the sub-optimal accelerator scheduling mentioned in Section 2.3.

Table 1: Design alternatives and whether they support the desired properties.

Design Option	Efficient CPU Cycle Usage	Flexible Scheduling
Client $\xleftrightarrow{\text{GPUDirect}}$ GPU	✓	×
Client $\xleftrightarrow{\text{RDMA}}$ CPU $\xleftrightarrow{\text{PCIe}}$ GPU	×	✓
Client $\xleftrightarrow{\text{RDMA}}$ SNIC $\xleftrightarrow{\text{DMA}}$ GPU	✓	✓

We thus opt for the off-path SmartNIC, which comes with a general-purpose SoC off the critical path of the incoming packets. Leading examples of off-path SmartNIC products encompass the BlueField family [12, 13]. It is worth highlighting that off-path SmartNICs have predominantly been harnessed for task offloading and in bare-metal cloud environments [31, 40, 50, 56]. However, using off-path SmartNICs as the critical path for incoming packets has been less commonly explored. This is partly due to concerns that their relatively weaker SoCs may introduce bottlenecks, limiting latency and throughput [71, 76].

Nonetheless, we argue that the off-path SmartNIC remains the optimal choice for our purposes for two key reasons, as listed in Table 1. First, it effectively bypasses the host CPU, thus mitigating the bottleneck on the CPU caused by heavy communication costs. Second, the benefits derived from its general programmability outweigh the minor latency advantage that on-path SmartNICs offer. Consequently, the off-path SmartNIC strikes the right balance between performance and flexibility.

3 Conspirator Design

3.1 Design Overview

We propose Conspirator, a SmartNIC-aided control plane for distributed ML workloads. Figure 3 illustrates the architecture of Conspirator where the control plane is located at the off-path SmartNIC SoC.

As illustrated in Figure 4, the initiation process entails the host CPU reserving GPU memory chunks across different (virtual) GPU instances. Concurrently, the host CPU establishes a secure communication pathway with the SmartNIC SoC for (*i*) sending the memory pointers to the control plane residing within the SmartNIC SoC, and (*ii*) acquiring data availability signals.

The SmartNIC SoC provisions a local buffer to process incoming requests, presuming they are transmitted using the RDMA protocol for superior efficiency compared to the traditional TCP/IP stack. Upon receipt of a request, the control plane identifies the most suitable GPU instance for task assignment following the heuristic introduced in Section 3.3 and initiates a DMA transfer of the data.

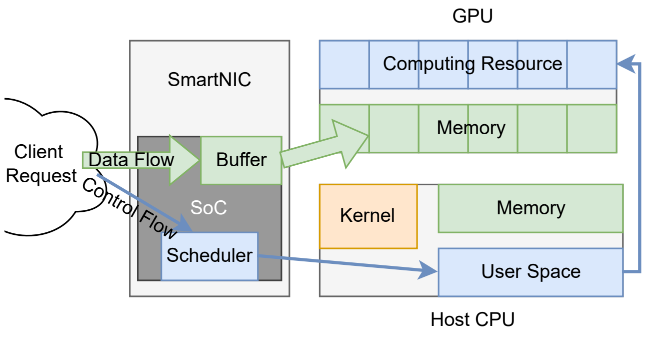


Figure 3: System overview.

Once the DMA transfer is completed, the control plane alerts the host CPU, which subsequently triggers the designated GPU kernel to process the data. Once the processing task concludes, the GPU kernel informs the host CPU, which then forwards the results to the SmartNIC SoC using the secure communication channel. The SmartNIC SoC completes the cycle by returning the results to the original requester.

The above process circumvents the host CPU to the greatest extent, only requiring it to wait on data availability signals and start the proper GPU kernel. All the data transfer is kernel-free and thus occupies few resources.

3.2 SmartNIC-Aided Control Plane

Below, we explain our design and implementation of the SmartNIC-aided control plane.

SmartNIC selection. As explained earlier in Section 2.4, we opt for the off-path SmartNIC, which is equipped with a general-purpose SoC off the critical path of the incoming packets. The most notable product of the off-path SmartNIC is the NVIDIA BlueField family. We implement our control plane with the latest product BlueField-3, which provides 4X more compute power, 2X faster storage processing, and 4X more memory bandwidth compared to Bluefield-2 [14].

Helpers at host CPU. In order to mitigate the performance degradation of ML workloads due to CPU contention, Conspirator adopts an approach that emphasizes simplicity for functions executed on the host CPU. This involves three primary tasks.

The first task is memory reservation at GPU instances. The host CPU is responsible for reserving memory within the (virtual) GPU instances. Subsequently, it maintains the availability of memory pointers and dispatches them to the SmartNIC's controller. The second task is the GPU kernel management. Upon receiving instructions from the SmartNIC's controller, the host CPU undertakes the role of initiating and invoking GPU kernels for processing incoming jobs. Moreover, the host CPU also takes charge of any auxiliary computations that are unsuitable for execution on the GPUs.

All three tasks can only be done by the host CPU because the SmartNIC lacks the capability to directly invoke a GPU

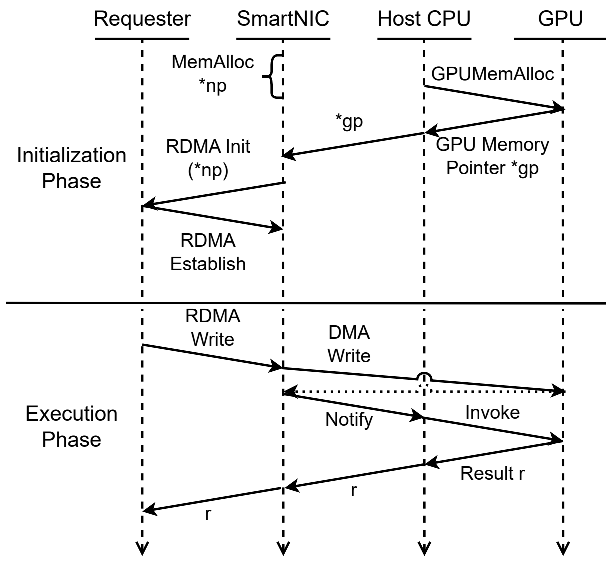


Figure 4: Conspirator procedure flowchart per requester.

kernel, as it connects to the GPU indirectly through a PCIe switch, unlike the direct PCIe connection between the GPU and the host CPU.[1]

As shown in Figure 4, we also assume the results are obtained by the host CPU and then transmitted to the SmartNIC, rather than using the SmartNIC DMA to read the results in the GPU. This is because we want to minimize changes to the existing ML code, where the final step of obtaining the result is usually executed on the CPU. While it is possible to leverage DMA, this would require modifying the ML code to store the results in GPU memory. Given the small size of the results, using DMA offers imperceptible performance benefits and hence does not justify the cost of ML code reconstruction.

For our implementation, we build and install a CUDA extension that creates a GPU tensor object from an address pointer. Then, at the initialization of Conspirator, we reserve memories on each GPU instance using the `doca_gpu_mem_alloc` function of the DOCA library. The reservation function will return the memory pointers, with which we can create Tensor objects later by calling our CUDA extension. This enables us to reuse any existing ML training or inference code in Python, by modifying just one line of code pertaining to the creation of the input tensor.

Control plane at SmartNIC. The controller of Conspirator operates at the subsystem of SmartNIC's ARM SoC. The core function of the controller is proper resource allocation. To achieve that, the controller should first obtain the pointers to all the (virtual) GPU instances from the helper functions at the host CPU. Then, the controller should establish RDMA connections in advance with potential requesters. The controller should assign the requesters with a dedicated local

[1]An exception is the NVIDIA Converged Accelerators [15] which physically integrate the GPU and the SmartNIC. More related discussion please refer to Section 5.

buffer intended for temporary data storage before it is relayed to the GPUs.

After completing the initialization, the controller assumes the task of evaluating the most suitable GPU instance for incoming jobs. After the allocation decision is made, it orchestrates the data transfer to the corresponding GPU and simultaneously issues a signal to the host CPU, thus triggering the commencement of GPU processing.

It is noteworthy that the use of local buffers introduces a minor latency in data transfer. However, the impact of this latency remains marginal due to the nature of zero-copy data handling: the data is written via RDMA and read via DMA, effectively mitigating potential performance drawbacks.

The above functions facilitate four distinct communication channels associated with the SmartNIC SoC: (*i*) a secure communication channel linked to the host CPU, (*ii*) a DMA channel connected to the GPUs, (*iii*) an RDMA channel established with the job requesters, and (*iv*) a supplementary channel connecting to job requesters for negotiating RDMA setup specifics. For channels (*i*) and (*iv*), implementation flexibility allows the use of any reliable transfer protocols. Channel (*ii*) mandates that the GPU shares the same PCIe bus as the SmartNIC, while channel (*iii*) requires the availability of RDMA capabilities, such as the InfiniBand Fabric.

Our implementation relies on the DOCA library given that it provides uniform APIs across RDMA, DMA, and beyond. Specifically, we implement channel (*i*) using the DOCA Comm Channel [16][2], channel (*ii*) using the DOCA DMA library [17], and channel (*iii*) using the DOCA RDMA library [18]. For channel (*iv*), we use the TCP socket. As channel (*iv*) only functions as an RDMA protocol negotiator before the actual data transfer, it does not impact runtime performance.

Concurrency Design. It is noteworthy that, in Figure 4, both the initialization and execution phases between the requester and SmartNIC are handled per job requester. The controller assigns each requester a distinct local buffer to prevent contention. Additionally, the controller maintains a job queue. Whenever it receives a signal indicating that a buffer is filled (i.e., a requester has sent a task and completed transferring the data), it creates a job and adds it to the queue. The controller then periodically executes the resource scheduling algorithm (see Section 3.3) to determine the next actions, such as which jobs are DMA'ed to which GPU instance.

[2]We use DOCA Comm Channel primarily for coding integral considerations: it is the only SmartNIC-CPU communication channel provided by the DOCA library. This channel is supposed to be faster than the TCP connection between SmartNIC and the host CPU. But given the small size of the exchanged data (e.g., signals and results), this benefit will likely be imperceptible.

3.3 ML workload Scheduling on Heterogeneous nodes

Conspirator also envisions the resource scheduling to be pivotal in optimizing the ML workloads. The GPU resource scheduling relies on resource isolation by GPU virtualization techniques, such as MPS [11] and MIG [21], and considerations on heterogeneity for both GPU resources and ML models.

Specifically, when executing multiple jobs on the same GPU, the conventional (default) approach involves CUDA kernels processing each job in a time-multiplexed manner. However, this method becomes inefficient due to the considerable overhead imposed by context switching between different jobs. MPS, on the other hand, introduces the concept of multiple virtual instances, harnessing NVIDIA GPUs' Hyper-Q capability. This empowers multiple processes to leverage concurrent CUDA kernel processing on a single GPU. The outcome is significantly improved performance, attributed to MPS's reduction of context switch overheads and more efficient GPU sharing compared to the default time-multiplexing approach.

MIG enhances the resource isolation paradigm by segregating GPU memory resources within the same physical GPU. Consequently, a MIG instance can be conceptualized as a physical GPU instance. Notably, the key distinction between a MIG instance and a physical GPU lies in the dynamic adjustability of resources for the former. This dynamic resource adaptability enhances resource utilization and further contributes to optimized ML workload execution.

However, existing container platforms (e.g., Kubernetes) typically adhere to a paradigm where GPUs are exclusively assigned to individual containers [3] or employ a time-multiplexing strategy for GPU sharing [1, 2]. They do not consider efficiently sharing GPUs while scheduling applications requiring GPU resources, and hence often cause resource inefficiency and performance degradation. Also, in a multi-tenant environment, software architects need to ensure that the shared GPU memory resource is isolated between tenants. Such isolation can be realized by MIG but not MPS.

Conspirator desires to realize a fine-grained GPU sharing with heterogeneity considered. To this end, we propose a new bin-packing-based GPU scheduler that works on a container platform (e.g., Kubernetes framework) and provides tenant isolation and security benefits. The tenant isolation means that no two tenants should share the same GPU computing and memory resources. This helps protect against attackers who might illicitly access data from other tenants residing in the same GPU's memory [64, 84]. This requirement arises from real-world demands in our production data centers. Our GPU scheduler realizes tenant isolation by leveraging MIG capabilities while minimizing migration cost and operational costs by leveraging MPS capabilities. Our GPU scheduler also incorporates considerations of heterogeneous GPU types and the diverse demands of ML workloads. This ensures a well-rounded approach that caters to varying computational requirements and optimally allocates resources.

All the above GPU information for making the scheduling decision is stored in the SmartNIC during the initialization phase. Upon receiving a new job request, the controller at the SmartNIC makes the decision locally. Following that decision, the controller directly transfers the data to the target GPU/MIG device and pings the host CPU to start the appropriate GPU kernel.

3.3.1 Problem Formulation

Below, we model the distributed ML workload scheduling[3] as a mixed integer linear programming (MILP) problem.

Inputs. We denote R_{it} as the requested computing resource for job i by tenant t, where $i \in [1,N]$ and $t \in [1,T]$. Additionally, y_{jw} indicates whether the w-th fraction of GPU j has been allocated to at least one job. The total number of GPUs is J, i.e., $j \in [1,J]$, and each GPU j comprises W_j fractions. We assume non-overlapping isolation of computing and memory resources among fractions. For instance, an NVIDIA MIG instance qualifies as a GPU fraction, whereas NVIDIA MPS does not meet this criterion because it does not isolate memory resources.

W_j can vary for different GPU types, based on configurations. For instance, an A30 GPU can accommodate a maximum of 4 MIG instances, while an A100 GPU can host up to 7 instances [21]. Additionally, GPUs might feature diverse fraction shapes. As an illustration, the A100 can offer fractions with 1/7, 2/7, 3/7, 4/7, or 7/7 of total resources, resulting in 14 possible configurations as illustrated in Table 3. Our formulation will determine the effective fractions and impose constraints to ensure that the total resource usage of selected fractions on a GPU does not exceed C_j, the total resources of that GPU.

Outputs. We assume that the distributed ML workload scheduling is dynamic. The binary variable k_{ijtw} represents the prior assignment of jobs in the system, whereas x_{ijtw} represents the latest job assignments. Furthermore, Y_j is a binary variable indicating whether GPU j has any job assignments, and δ_i is a binary variable indicating whether job i requires migration for optimal assignment. The cost of migrating an ongoing job i is denoted by m_i. Table 2 summarizes the notation used in this paper.

Problem formulation. Our primary goal is to minimize both GPU operational and job migration costs, considering the substantial costs associated with migration. We introduce coefficients ε_1 and ε_2 to adjust the objective. The mixed integer problem representing ML workload scheduling (which we refer to as **MinGPUCost** problem) is defined as follows.

[3]While our paper focuses on ML workloads, our scheduler is also applicable to other non-ML jobs, as long as they provide the necessary inputs such as the requested accelerator resource.

Table 2: Main notations employed in this paper

Notation	Explanation
R_{it}	The requested GPU resource for job i by tenant t
y_{jw}	Decision variable to assign GPU j's wth fraction for at least one job (when set to 1) and not otherwise (when set to 0)
W_j	The number of fractions for GPU j
α_{jw}	The computing resources of GPU j's wth fraction
C_j	The total capacity of GPU j
k_{ijwt}	The existing job assignment that shows if job i is scheduled on GPU j for tenant t, if set to 1 and not otherwise.
x_{ijwt}	The latest job assignment
Y_j	Whether GPU j has been assigned for at least one job
δ_i	Decision variable to migrate job i when set to 1 and not otherwise (when set to 0).
m_i	The migration cost for job i

$$\min_{x_{ijwt}} \ \varepsilon_1 \sum_j Y_j + \varepsilon_2 \sum_i m_i \delta_i \tag{1}$$

$$s.t. \ \sum_t \sum_j \sum_w x_{ijwt} = 1, \forall i \in [1,N] \tag{1a}$$

$$\sum_j \sum_w x_{ijwt} \le R_{it}, \forall i \in [1,N], \forall t \in [1,T] \tag{1b}$$

$$\sum_t \sum_i R_{it} \cdot x_{ijwt} \le \alpha_{jw} \cdot y_{jw}, \forall j \in [1,J], \forall w \in [1,W_j] \tag{1c}$$

$$\sum_w \alpha_{jw} \cdot y_{jw} \le C_j, \forall j \in [1,J] \tag{1d}$$

$$Y_j \ge \frac{\sum_w y_{jw}}{N}, \forall j \in [1,J] \tag{1e}$$

$$\delta_i = 1 - \sum_t \sum_j \sum_w x_{ijwt} \cdot k_{ijwt}, \forall i \in [1,N] \tag{1f}$$

$$A_{jt} \ge \frac{\sum_t \sum_w x_{ijwt}}{N}, \forall j \in [1,J], \forall t \in [1,T] \tag{1g}$$

$$\sum_t A_{jt} \le 1, \forall j \in [1,J] \tag{1h}$$

$$\delta_i, x_{ijwt}, k_{ijwt}, A_{jt}, y_{jw}, Y_j \in \{0,1\} \tag{1i}$$

Specifically, Equation 1a ensures the assignment of each job i only once, while Equation 1b enforces assignments solely for valid jobs where $R_{it} > 0$. Following this, Equation 1c safeguards that the cumulative resources allocated to jobs within the w-th fraction of GPU j do not surpass the fraction's total capacity. Further, Equation 1d extends this constraint to encompass the cumulative resources allocated across all GPU fractions, ensuring that the aggregated resources of

Table 3: The GPU fraction resources for A100.

w	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\alpha_{jw}(\%)$	14	14	14	14	14	14	14	28	28	28	42	42	57	100

the selected fractions do not exceed the physical GPU's total capacity.

Next, Equation 1e serves to flag GPUs that have been assigned to at least one job. Subsequently, Equation 1f determines whether the migration of job i is necessary to achieve optimal resource allocation. To enhance data security in GPU sharing within a multi-tenant context, Equations 1g and 1h introduce the intermediary variable A_{jt}, and they together ensure that all jobs assigned to the same GPU fraction originate from a single tenant. Last, Equation 1i comprehensively catalogs all binary variables implicated within the formulation.

Theorem 3.1. ***MinGPUCost** is an NP-Hard problem.*

Proof. Bin packing is a well-known combinatorial optimization problem that is strongly NP-Hard [36, 38, 68]. We prove **MinGPUCost** problem is NP-Hard by showing that bin packing problem is a special case of **MinGPUCost**. To reduce the bin packing problem to an instance of our problem, we create one tenant with a finite set I of job requests (items), a size for each job request $R_i, \forall i \in I$. Further, we assume having J GPUs (bins) of the same type (homogeneous case) and a migration cost of zero (*i.e.,* $\varepsilon_2 = 0$). We also assume each GPU only uses the largest MIG instance (MIG 7g.80gb in our case) with a bin capacity of 100. This creates an instance of bin packing problem. Since the bin packing problem is NP-hard, the **MinGPUCost** problem is also NP-hard. □

3.3.2 Greedy Heuristic

Since MinGPUCost is an NP-Hard problem, finding the optimal resource allocation entails solving Equation 1 through techniques like the Branch and Bound algorithm [32]. Yet, it is crucial to acknowledge that such approaches often entail considerable computational complexities. Unfortunately, these complexities do not align with our objective of minimizing latency and maximizing throughput, which demands a more efficient strategy.

Consequently, we introduce a practical and expedient solution in the form of a greedy heuristic, which ignores the job migration and efficiently computes resource allocations for incoming jobs with a time complexity of $O(NJW_j)$.

Below, we explain our heuristic algorithm presented in Algorithm 1 in detail. First, lines L1-2 synchronize existing allocations and initialize variables T_{ijw} representing GPU fraction occupancy by tenant. L3 then iterates through all jobs. For each job i, L5-10 determines the tenant t_i who requested it.

Algorithm 1 ML Workload Scheduling Heuristic.

```
Require: $R_{it}, y_{jw}, W_j, \alpha_{jw}, C_j, k_{ijwt}$
Ensure: $x_{ijwt}$
 1: $x_{ijwt} \leftarrow k_{ijwt}$ // Synchronize existing allocations
 2: $T_{ijw} \leftarrow 0$ // GPU fractions' tenant occupation
 3: for $i \leftarrow 1$ to $N$ do
 4:     $success \leftarrow False$
 5:     for $t \leftarrow 1$ to $T$ do
 6:         if $R_{it} = 1$ then
 7:             $t_i \leftarrow t$ // Determine the tenant for job $i$
 8:             break
 9:         end if
10:     end for
11:     for $j \leftarrow 1$ to $J$ do
12:         if $success = True$ then
13:             break
14:         end if
15:         for $w \leftarrow 1$ to $W_j$ do
16:             if $success = True$ then
17:                 break
18:             else if $T_{ijw} \neq t_i$ then
19:                 continue
20:             else if $y_{jw} = 1$ then
21:                 $r_{left} \leftarrow \alpha_{jw}$
22:                 for $i \leftarrow 1$ to $N$ do
23:                     $r_{left} \leftarrow r_{left} - x_{ijwt_i} * R_{it_i}$
24:                 end for
25:                 if $r_{left} \geq R_{it}$ then
26:                     $x_{ijwt} \leftarrow 1$
27:                     $success \leftarrow True$
28:                 end if
29:             else
30:                 $r_{left} \leftarrow C_j$
31:                 for $w_2 \leftarrow 1$ to $W_j$ do
32:                     $r_{left} \leftarrow r_{left} - y_{jw} * \alpha_{jw}$
33:                 end for
34:                 if $r_{left} \geq \alpha_{jw}$ and $r_{left} \geq R_{it}$ then
35:                     $x_{ijwt} \leftarrow 1$
36:                     $y_{jw} \leftarrow 1$
37:                     $T_{ijw} \leftarrow t_i$
38:                     $success \leftarrow True$
39:                 end if
40:             end if
41:         end for
42:     end for
43: end for
```

Next, we iterate through all GPUs (L11-40) and their fractions (L15-39) until the current job i is successfully allocated (L12-14 and L16-17).

While the job i is not yet allocated, L18-19 determine whether the current GPU fraction is occupied by another tenant. If so, proceed to the next fraction. If the GPU fraction has been previously allocated to a job belonging to tenant t_i (L20), L21-24 calculate whether there are available resources for job i. If resources are available, L25-28 assign job i to

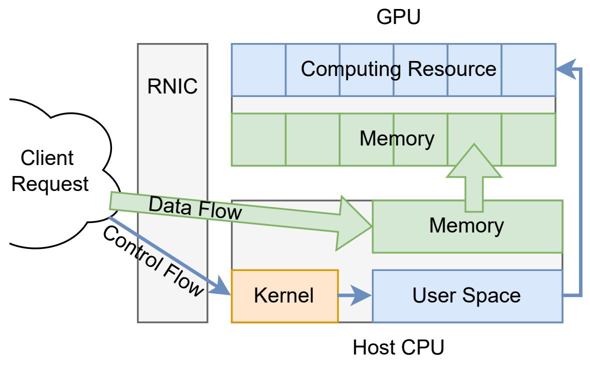


Figure 5: An alternative design which relies on RDMA for remote data transmission and exhibits a local scheduler at host CPU.

this fraction. If the GPU fraction is currently unoccupied (L29), L30-34 determine if it can be initialized and whether it has sufficient resources for job i. If both conditions are met, L35-38 assign job i to this fraction. It is noteworthy that in Algorithm 1, we have presented only the generation of x_{ijwt} as an example to streamline the explanation. Other outputs such as Y_j can also be obtained from the algorithm without introducing any additional complexity. Overall, this heuristic efficiently computes resource allocations for incoming jobs with a time complexity of $O(NJW_j)$, where N is the total number of jobs, J is the total number of GPUs and W_i is the maximum number of fractions among the GPUs.

4 Evaluation

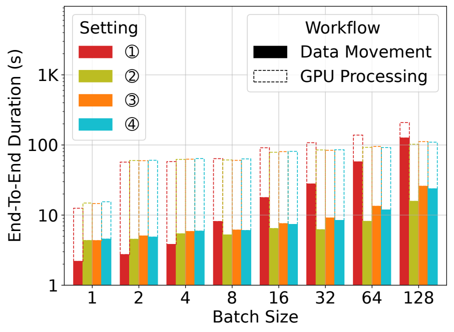
Below, we comprehensively assess Conspirator from two distinct perspectives: end-to-end performance and the advantages derived from workload scheduling. We initiate this evaluation by presenting the results of our end-to-end system analysis, conducted using a testbed comprising servers equipped with BlueField-3 SmartNICs and A100 GPUs. Subsequently, we proceed to the performance evaluation of Conspirator's workload scheduling. This evaluation is underpinned by a real-world dataset portraying GPU usage patterns in contemporary data center environments.

4.1 End-To-End Evaluation

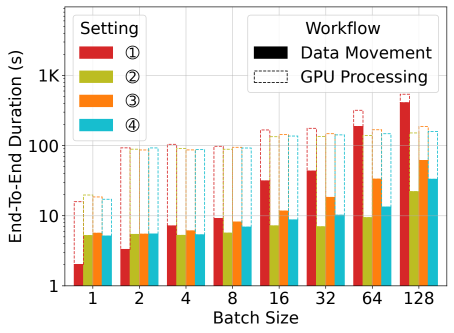
4.1.1 Evaluation Setup

Our testing infrastructure comprises two servers, each equipped with BlueField-3 SmartNICs and A100 GPUs. To facilitate a comprehensive comparison, we consider several distinct configurations:

① TCP Server with Host CPU Handling: In this setup, job requests are managed by a TCP server on the host CPU, which subsequently conveys the job data to the GPU. (Figure 1(a))



(a) 0% background CPU load.



(b) 80% background CPU load.

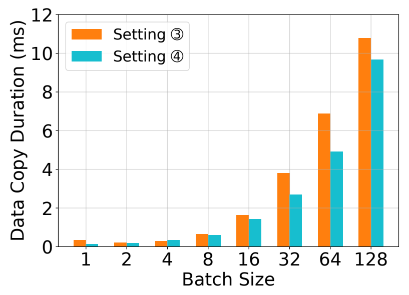
Figure 6: End-to-end duration.

② Direct Data Transfer to GPU Memory: This configuration involves the direct transfer of job data to GPU memory through NVIDIA's GPUDirect RDMA technology. (Figure 1(b))

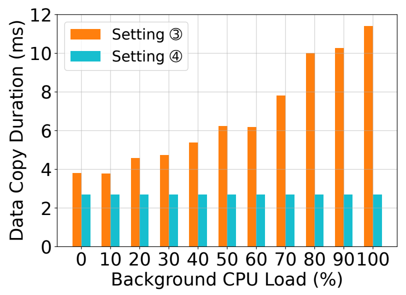
③ Direct Data Transfer to Host Memory: Here, the job data will be transferred to a chunk of memory located at the host. The host CPU will later decide on the job scheduling and transfer the data to the corresponding GPU memory using the CUDA library. This setting closely resembles Conspirator, as it employs RDMA for remote data transmission and incorporates a local scheduler. The primary distinction lies in the absence of SmartNIC utilization; instead, the scheduler resides on the host CPU. Figure 5 provides an architectural representation and information flow for this setting.

④ Conspirator Implementation: In the Conspirator framework, job requests are transmitted via RDMA to the SmartNIC SoC, which then orchestrates data transfer to the appropriate GPU memory using DMA. (Figure 3)

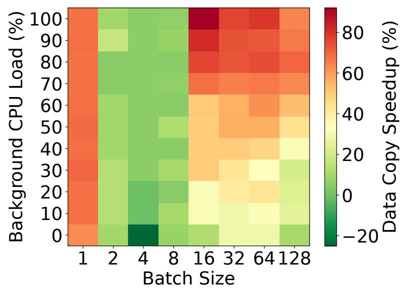
These distinct configurations enable us to conduct a comprehensive performance assessment and comparative analysis of Conspirator against other pertinent alternatives. In our evaluation, we employ an ML inference workload designed to process incoming batches of images from the ImageNet dataset [8]. We further assume that the images have been pre-processed before arriving at Conspirator, e.g., adopting a similar architecture where data pre-processing is delegated



(a) Impact of batch sizes.



(b) Impact of batch sizes background CPU load. Batch size is 16.



(c) Setting ④'s speedup to setting ③.

Figure 7: Local data transmission evaluation.

to dedicated servers [82] which act as the job requesters in our settings. It follows that the transmitted data will be in a transformed format. Specifically, for a batch size of one image, the data will occupy around 600 KB.

4.1.2 Results

End-to-end performance. In Figure 6(a), we illustrate the end-to-end duration across various data path configurations and diverse job batch sizes. The results indicate that configuration ① is generally less performant than other RDMA-based configurations, except for scenarios with small batch sizes (*i.e.,* one or two). The relative efficiency of ① in these cases can be attributed to the current limitations in RDMA's optimization when using Nvidia's DOCA library. However, it is noted that utilizing other RDMA-based libraries, as evidenced in existing research [76], would likely enhance the performance of RDMA-based settings over TCP even for small batch sizes. The rest of the outcome is expected as the TCP/IP stack has been proven to incur a notably high overhead.

Next, setting ② allows the job requester to directly place data at the target GPU memory, without any hindrance in the way. Indeed, such a setup brings the fastest end-to-end duration compared to all other settings, despite that the performance discrepancies with other RDMA-based settings are marginal. For example, Conspirator (setting ④) only adds a minimal delay of up to a few seconds, or up to 8% of the total time, for making scheduling decisions as well as the local data transmission using DMA, compared to setting ②. In contrast, setting ② does not allow any GPU scheduling and may lead to sub-optimal performance (see Section 4.2), unless we unrealistically assume that the job requester has real-time visibility into every accelerator. We thus discard this option.

Setting ③ and ④ both leverages the power of RDMA and GPU scheduling simultaneously. The difference is that setting ③ decides on the GPU scheduling and transfers the job data to the GPU using the host CPU whereas setting ④ places such job at the SmartNIC SoC. Figure 6(a) shows that setting ④ outperforms setting ③ with respect to the end-to-end duration in most scenarios. The performance gap between settings ③ and ④ becomes larger if we only consider the data movement and exclude the ML inference at GPU.

To mimic a real-world data center scenario with ongoing background CPU utilization, particularly in shared environments, we added CPU load while testing our four configurations. Figure 6(b) shows that under high background CPU load (80%), the performance differences between the settings become more pronounced. Specifically, the end-to-end performance of configuration ① is up to 5 times slower than the others, or 12 times slower if excluding GPU processing time. Configuration ④ consistently outperforms ③ by up to 30s or 20% of the total end-to-end latency, while only adding a few seconds of delay compared to ②, with the maximum discrepancy of about 8s at a batch size of 128.

Zooming in data movement. We further narrow our focus to settings ③ and ④ with respect to the data movement, which consists of three sub-tasks: (*i*) RDMA transmission, (*ii*) local data transmission, and (*iii*) GPU scheduling. Among them, setting ③ and ④ only differ in the second sub-task of local data transmission, which we take a closer look at below.

Figure 7(a) depicts the durations for local data transmission for settings ③ and ④ with varying batch sizes. Overall, setting ④ consistently outperforms ③ across most scenarios. An exception arises when the batch size is 4, and this anomaly could potentially be attributed to some internal mechanisms within the CUDA library. The most significant performance disparity is observed when the batch size is 1, with setting ④ reducing the duration by an impressive 63%. This phenomenon can be again attributed to internal mechanisms within the CUDA library, as we consistently observe that the host CPU takes less time to transfer data for batch sizes 2 and 4 compared to batch size 1. For the remaining batch sizes, setting ④ reduces the duration of local data transmission by percentages ranging from 3% to 10%.

While the performance improvement may not appear substantial, one of the key advantages of setting ④ lies in its less reliance on the host CPU compared to setting ③. This characteristic becomes particularly valuable in real-world cluster environments where the host CPU may frequently be

Table 4: Cost efficiency of Conspirator.

Hardware	Price (Normalized)	Power Consumption	Throughput (Normalized)	Cost-Effectiveness	Power Efficiency
Host CPU	\$1,000	800W	1,000	1.0	1.25
SmartNIC	\$231	150W	270	**1.17 (+17%)**	**1.8 (+44%)**

occupied by various tasks, whether related to ML or unrelated to it. To emulate such scenarios, we introduced an additional CPU-intensive program, thereby introducing background host CPU utilization. As demonstrated in Figure 7(b), the performance of setting ③ is negatively affected by the presence of background CPU utilization. The local data transmission duration increases as the background CPU utilization rises, escalating from less than 4 ms to a maximum of over 11 ms. Consequently, the speedup achieved by setting ④ in comparison to setting ③ grows from 29% to a substantial 76%.

Figure 7(c) provides a comprehensive overview of the speedup achieved when transitioning from setting ④ to ③, considering various batch sizes and background CPU utilization levels. It is noteworthy that speedup is evident in all configurations except when the batch size is 4 and when CPU utilization is 0%, as previously discussed. The most substantial speedups are observed when the batch size is either 1 or greater than 8. In summary, setting ④ yields an average speedup of 36%. Importantly, the average speedup increases as background CPU utilization rises, ranging from 17% when CPU utilization is at 0%, through a substantial 30% when CPU utilization is at 30%, to an impressive 50% when CPU utilization reaches 100%. It is worth noting that if we account for RDMA data transmission, these numbers would be halved, but they still represent significant improvements.

Cost effectiveness and power efficiency. Beyond performance advantages, Conspirator's most significant benefits lie in its cost effectiveness, defined as throughput relative to the purchase price, and power efficiency, measured as throughput per maximum power consumption. Table 4 reveals that although a host CPU might have higher throughput due to more CPU cores, SmartNIC (setting ④) outperforms a setup relying solely on host CPUs (setting ③) in terms of cost and power, being 17% more cost-effective and 44% more power-efficient. Notably, the pricing for the NVidia BlueField-3 SmartNIC listed in the table represents its total cost after normalization. Considering that when SmartNIC is absent, a standard server would still require a NIC for operation, the actual cost-effectiveness and power efficiency of Conspirator are likely even more advantageous than indicated. This underscores Conspirator's significant value in data center environments.

4.2 ML Workload Scheduling

Evaluation Setup. Next, we evaluate the ML workload scheduling capabilities of Conspirator. To provide a meaningful basis for comparison, we have implemented two additional scheduling approaches. First, we have created an optimal scheduler that solves the mixed ILP problem defined in Equation 1 using OR-Tools [7]. To obtain the optimal GPU assignment at all times, we set ε_2 in Equation 1 to be zero, meaning that there is no cost for job migration. Second, we have designed a heuristic approach similar to our system where it leverages NVIDIA MPS to enable multiple jobs from the same tenant to share the same GPU. However, it does not consider the use of MIG technology.

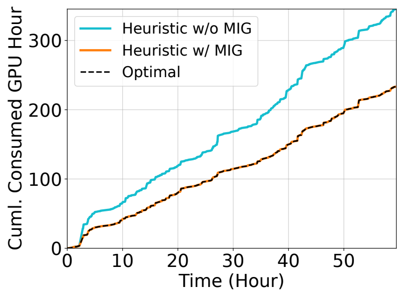
Our evaluation is conducted using the Alibaba Cluster Trace GPU 2020 dataset [4, 77], which captures a wide range of training and inference jobs within a large production cluster featuring diverse GPU types. The original dataset chronicles the activities of more than 1,300 users over a span of two months. To streamline the presentation of our evaluation results, we have restricted the analysis to a subset considering activities over a continuous 60-hour period.

Results. Figure 8(a) shows the cumulative consumed GPU hour by all the jobs over time following the scheduling decisions made by either the heuristics or the optimal scheduler. The jobs, when scheduled optimally, collectively require 233 GPU hours. Remarkably, our heuristic achieves this identical result without any performance degradation, surpassing the total of 345 GPU hours necessitated by simpler heuristics that do not take advantage of the MIG feature. This illustrates a 33% savings on the total consumed GPU hours.

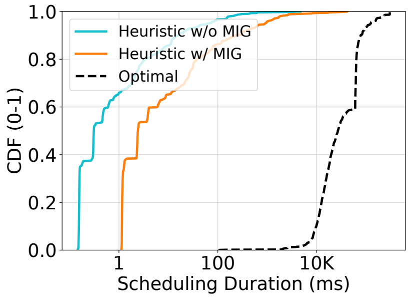
Next, we proceed to examine the scheduling durations. Figure 8(b) shows the CDFs of scheduling durations for both the heuristics and optimal scheduler that solve the MILP problem. Unfortunately, the optimal scheduler, while capable of providing the optimal solution, exhibits significant delays, ranging from at least 100 ms to potentially several minutes.[4] Such protracted scheduling times outweigh the advantages of optimal scheduling and render it impractical for real-world systems. In contrast, our heuristic is swift, necessitating a median time of only 4 ms – over 1,000x faster than the optimal scheduler – to determine the optimal scheduling. This imposes an acceptable level of overhead on Conspirator. Note that the scale of the dataset used for this evaluation necessitates the availability of over 80 GPUs for scheduling. In real-world deployments, single machines typically host only a few GPUs. As a result, the scheduling duration of our proposed heuristic within Conspirator is expected to be significantly shorter.

We further evaluate the applicability of the scheduler. Figure 8(c) illustrates the scheduling durations when executing on the SmartNIC with no background use and on the host CPU

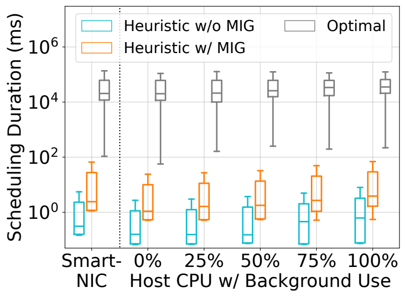
[4]A cut-off timer is set to 10 minutes to prevent optimal scheduling tasks from taking excessively long to complete.



(a) Cumulative consumed GPU hour over time.



(b) CDF of scheduling durations.



(c) Boxplots of scheduling durations under different scenarios.

Figure 8: Scheduler evaluations.

with varying levels of background usage. Overall, the host CPU without high contention outperforms the SmartNIC, an expected result given the latter's weaker SoC. However, when background CPU usage reaches or exceeds 75%, scheduling on the host CPU becomes slower than on the SmartNIC. In either case, the scheduling heuristic completes in a few milliseconds, even when handling a larger-than-usual problem scale, remaining within the acceptable range. This demonstrates the good applicability of our scheduler.

In conclusion, these results underscore the efficacy of our proposed heuristic, which effectively achieves GPU resource scheduling that is close to optimal or even optimal in this particular dataset. Importantly, it accomplishes this with significantly reduced time requirements, rendering it a practical solution and a good fit in Conspirator.

5 Discussion

Distributing client requests to both SmartNIC SoC and CPU. One additional alternative is to leverage both the SmartNIC SoC and CPU for accepting requests at the same time. As Xing et al. [76] suspects, SmartNIC internally reserves some NIC cores for each endpoint, *i.e.,* the SmartNIC SoC and host. Hence, it is possible to gain performance improvement if both the SoC and the host take requests and ship them to the GPU. However, this requires careful tunning on both the ratio of the requests and the scheduling algorithm – as it needs to work distributively in this scenario – to maximize the benefits. We leave this as future work.

Non-NVIDIA GPU Technologies. Other GPU vendors such as AMD and Intel also offer peer-to-peer data movement technologies. AMD's PeerDirect [22] allows the transfer of data from GPU to other PCIe devices such as NICs. Meanwhile, Intel offers DMABuf [9] with similar features as NVIDIA's GPUDirect. However, these techniques are at different levels of maturity and run only on hardware from specific vendors. We thus opt for NVIDIA solutions in this paper.

Converged Accelerator Cards. In response to the challenges posed by bottlenecks in distributed ML workloads, industries have explored novel hardware architectures to address these limitations. Notably, the introduction of Nvidia's A100X converged accelerator [15] stands out, leveraging the robust performance of the NVIDIA Ampere architecture in conjunction with the enhanced latency reduction capabilities offered by the NVIDIA BlueField-2 data processing unit (DPU). More concretely, the converged accelerator physically integrates the GPU and the SmartNIC with a PCIe bus. This allows the SmartNIC to play the role of the host CPU, *i.e.,* directly managing the GPU kernel.

It is noteworthy, however, that such hardware architectures come with two major limitations. First, the SmartNIC's ARM SoC, acting as the host CPU in this context, demonstrates inadequacy in terms of computational strength. It struggles to efficiently handle auxiliary aspects of ML workloads, consequently leading to an overall performance degradation. Second, the converged accelerators are constrained by a one-to-one correspondence between the DPU and the GPU, meaning that the DPU in embedded mode can communicate solely with one A100 GPU node on the converged card. In contrast, our proposed architecture boasts the advantage of overcoming this limitation, demonstrating compatibility with multiple GPUs on the same node. This becomes particularly pivotal in the context of large machine learning models, such as large language models (LLMs) [61], where the model's scale exceeds the capacity of a single GPU.

Balance of System Configurations. While this paper focuses on reducing the communication costs to relieve the bottleneck on CPUs, we acknowledge that it is not the only source. The CPU contention is resulted by the collective efforts of many tasks, including data pre-processing, network stack overhead, aided computation for ML, background CPU tasks, and more. The specific bottlenecks in ML inherently depend on (*i*) ML workload characteristics, (*ii*) the balance of the system configurations, and (*iii*) other background workloads.

For example, some ML models are more computationally intensive than others [79], and training with parameter servers burdens the CPU more than all-reduce training [48]. Further, in the evaluation of Conspirator, we assume an inference workload, where communication and network stack will have a significant impact on CPU load because of the large amount of data volume transmitted.

The balance of system configurations, i.e., the hardware capabilities of the GPUs, CPUs, and NICs, also plays an important role. For example, a server equipped with a strong GPU and a relatively weak CPU may experience network stack bottlenecks even if data pre-processing is delegated to another server [82]. In addition, servers handling ML tasks in multi-tenant environments face additional CPU loads.

Irrespective of what causes the CPU bottleneck, a high CPU utilization rate degrades ML performance [77]. Conspirator alleviates CPU load by removing it from the data path — regardless of whether the data is pre-processed or not —, leveraging the more cost-effective SmartNICs (see Table 4). From a different perspective, *Conspirator promotes a more cost-effective system configuration balance.*

On-path vs off-path SmartNICs. While this paper focuses on off-path SmartNICs due to their easier programmability, on-path SmartNICs merit further exploration. On-path SmartNICs come in various implementations; the job scheduling algorithm cannot be realized in some, such as the Netronome/Agilio SmartNIC with P4 programmable hardware [5]. Others, based on FPGAs [10, 20], may have the potential to implement the functionalities of Conspirator. We expect some of these to offer better performance than the off-path SmartNICs demonstrated in this paper. However, the reliance on low-level languages and complex hardware constraints poses significant development challenges. We acknowledge that the lack of further exploration and comparison of on-path SmartNICs is a limitation of our paper, and we leave this for future work.

6 Related Work

Heterogeneous accelerator scheduling. Specialized accelerators are increasingly deployed for ML workloads [26, 27, 39, 42, 60, 77, 78]. These accelerators demonstrate diverse performance characteristics across different workloads. While existing cluster schedulers for accelerators are proficient at managing the allocation of these workloads among multiple users, they have typically focused on optimizing one objective, such as fairness, throughput, or latency [42, 60, 78]. For instance, Optimus [62] and Tiresias [42] propose a GPU scheduling approach for distributed training workload that aims at minimizing the average job completion time. MLaaS [77] conducts an in-depth analysis of extensive workload traces within Alibaba, showing the advantages of GPU sharing in operational GPU data centers. The findings also highlight that, in their suggested approach, the CPU can act as a potential bottleneck. Gavel [60] is a scheduling policy designed for deep learning workloads, employing a round-robin-based scheduling mechanism to ensure that jobs receive their optimal allocation in alignment with the designated scheduling policy. Unfortunately, existing works do not consider the heterogeneous performance of workloads running on different GPU architectures.

Communication optimization for distributed ML workloads. Some ML schedulers [63, 67] study the impact of communication bandwidth on ML workload efficiency and propose to mitigate the communication overhead of ML training. Compression is widely studied and employed in the current distributed ML training [24, 28, 35, 73–75, 80]. In parallel, there are ongoing initiatives in the field that concentrate on delegating the responsibility of gradient aggregation to programmable switches [52, 67]. Conspirator does not conflict with any of the above work as it focuses on optimizing the local data transmission workflow.

SmartNIC. Recent works on SmartNICs focus on harnessing the computational capabilities of the SmartNICs to offload various application workloads [31, 55, 56, 65, 70]. Some propose using P4 switches to accelerate parameter server (PS) training applications [67]. While others propose to accelerate all-reduce by using InfiniBand switch [41]. Recent studies have demonstrated substantial advantages in offloading specific functions from host CPUs to more specialized hardware. iPipe [55] introduces the actor programming model as a solution for offloading applications like KV stores, distributed transaction systems, and real-time analytics to SoC-based SmartNICs. E3 [56] and λ-NIC [31] focus on offloading microservices to SoC-based SmartNICs. While these approaches share the common goal with Conspirator of alleviating the host CPU burden through task offloading and utilizing SmartNIC resources to enhance task acceleration, they do not specifically tackle the unique challenges associated with data movement between accelerators and workload placement and scheduling.

7 Conclusion

We have introduced Conspirator, a SmartNIC-assisted control plane designed to optimize distributed ML workloads by concurrently addressing two critical bottlenecks: CPU limitations and suboptimal accelerator scheduling. To attain this objective, Conspirator uses the SmartNIC to overcome these challenges and balance efficient data communication and optimal accelerator scheduling.

We have developed a prototype of Conspirator on the Nvidia BlueField-3 SmartNIC and conducted a comprehensive comparison with state-of-the-art RDMA-based alternatives. Our evaluation reveals that Conspirator yields a 15% reduction in end-to-end completion time compared to RDMA-based alternatives while being 17% more cost-effective and 44% more power efficient. Furthermore, our proposed scheduler not only contributes to a 33% reduction in GPU hours compared to naive GPU-sharing schedulers but also makes close-to-optimal decisions efficiently, requiring significantly less time than an optimal NP-Hard scheduler.

References

[1] Alibaba., GPU Sharing Device Plugin in Kubernetes. https://github.com/AliyunContainerService/gpushare-device-plugin.

[2] Deepomatic., Support for shared GPUs by declaring GPUs multiple times. https://github.com/Deepomatic/shared-gpu-nvidia-k8s-device-plugin.

[3] Kubernetes., Schedule GPUs. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/.

[4] Alibaba Cluster Trace GPU 2020, 2020. https://github.com/alibaba/clusterdata/tree/master/cluster-trace-gpu-v2020.

[5] Agilio CX SmartNICs, 2023. https://netronome.com/agilio-smartnics/.

[6] DPDK: Home, 2023. https://www.dpdk.org/.

[7] Google OR-tool, 2023. https://developers.google.com/optimization.

[8] ImageNet, 2023. https://www.image-net.org/index.php.

[9] Intel DMA Buf, 2023. https://github.com/intel/linux-intel-lts/blob/master/Documentation/driver-api/dma-buf.rst.

[10] Marvell LiquidIO III, 2023. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf.

[11] Multi-Process Service :: GPU Deployment and Documentation, 2023. https://docs.nvidia.com/deploy/mps/index.html.

[12] NVidia BlueField-2 DPU Datasheet, 2023. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet?lx=LbHvpR&topic=networking-cloud.

[13] NVidia BlueField-3 DPU Datasheet, 2023. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf.

[14] NVIDIA Bluefield-3 Networking Platform, 2023. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield.

[15] NVIDIA Converged Accelerators, 2023. https://www.nvidia.com/en-us/data-center/products/converged-accelerator/.

[16] NVIDIA DOCA Comm Channel Programming Guide, 2023. https://docs.nvidia.com/doca/sdk/comm-channel-programming-guide/index.html.

[17] NVIDIA DOCA DMA Programming Guide, 2023. https://docs.nvidia.com/doca/sdk/dma-programming-guide/index.html.

[18] NVIDIA DOCA RDMA Programming Guide, 2023. https://docs.nvidia.com/doca/sdk/rdma-programming-guide/index.html.

[19] NVIDIA GPUDirect, 2023. https://developer.nvidia.com/gpudirect.

[20] NVIDIA Mellanox Innova-2 Flex Open Programmable SmartNIC, 2023. https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/.

[21] NVIDIA Multi-Instance GPU User Guide, 2023. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html.

[22] ROCM Peer Direct, 2023. https://rocm.docs.amd.com/en/latest/how_to/gpu_aware_mpi.html.

[23] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl, A. Bhagat, G. Bhaskara, T. Brokhman, L. Cao, A. Cheema, R. Chow, J. Cohen, M. Elhaddad, V. Ette, I. Figlin, D. Firestone, M. George, I. German, L. Ghai, E. Green, A. G. Greenberg, M. Gupta, R. Haagens, M. Hendel, R. Howlader, N. John, J. Johnstone, T. Jolly, G. Kramer, D. Kruse, A. Kumar, E. Lan, I. Lee, A. Levy, M. Lipshteyn, X. Liu, C. Liu, G. Lu, Y. Lu, X. Lu, V. Makhervaks, U. Malashanka, D. A. Maltz, I. Marinos, R. Mehta, S. Murthi, A. Namdhari, A. Ogus, J. Padhye, M. Pandya, D. Phillips, A. Power, S. Puri, S. Raindel, J. Rhee, A. Russo, M. Sah, A. Sheriff, C. Sparacino, A. Srivastava, W. Sun, N. Swanson, F. Tian, L. Tomczyk, V. Vadlamuri, A. Wolman, Y. Xie, J. Yom, L. Yuan, Y. Zhang, and B. Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, pages 49–67. USENIX Association, 2023.

[24] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu. Gradient compression supercharged high-performance data parallel DNN training. In R. van Renesse and N. Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 359–375. ACM, 2021.

[25] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 1:1–1:16. ACM, 2020.

[26] H. Chen, Y. Ni, A. Zakeri, Z. Zou, S. Yun, F. Wen, B. Khaleghi, N. Srinivasa, H. Latapie, and M. Imani. Hdreason: Algorithm-hardware codesign for hyperdimensional knowledge graph reasoning. *CoRR*, abs/2403.05763, 2024.

[27] H. Chen, A. Zakeri, F. Wen, H. E. Barkam, and M. Imani. Hypergraf: Hyperdimensional graph-based reasoning acceleration on FPGA. In N. Mentens, L. Sousa, P. Trancoso, N. Papadopoulou, and I. Sourdis, editors, *33rd International Conference on Field-Programmable Logic and Applications, FPL 2023, Gothenburg, Sweden, September 4-8, 2023*, pages 34–41. IEEE, 2023.

[28] J. Chen, T. Lan, and C. Joe-Wong. Rgmcomm: Return gap minimization via discrete communications in multi-agent reinforcement learning. In M. J. Wooldridge, J. G. Dy, and S. Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 17327–17336. AAAI Press, 2024.

[29] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Q. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. TVM: an automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 578–594. USENIX Association, 2018.

[30] J. Cho, D. Z. Tootaghaj, L. Cao, and P. Sharma. Sla-driven ML inference framework for clouds with hetergeneous accelerators. In *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

[31] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum. λ-nic: Interactive serverless compute on programmable smartnics. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*, pages 67–77. IEEE, 2020.

[32] J. Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[33] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[34] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 492–506, 2020.

[35] J. Fei, C. Ho, A. N. Sahu, M. Canini, and A. Sapio. Efficient sparse collective communication and its application to accelerate distributed deep learning. In F. A. Kuipers and M. C. Caesar, editors, *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, pages 676–691. ACM, 2021.

[36] S. P. Fekete and J. Schepers. New classes of lower bounds for bin packing problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 257–270. Springer, 1998.

[37] M. Furukawa and H. Matsutani. Accelerating distributed deep reinforcement learning by in-network experience sampling. In *30th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2022, Valladolid, Spain, March 9-11, 2022*, pages 75–82. IEEE, 2022.

[38] M. R. Garey and D. S. Johnson. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms in combinatorial optimization*, pages 147–172. Springer, 1981.

[39] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. K. Reinhardt, and M. C. Herbordt. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*, pages 922–936. IEEE, 2020.

[40] P. Gootzen, J. Pfefferle, R. Stoica, and A. Trivedi. DPFS: dpu-powered file system virtualization. In *Proceedings of the 16th ACM International Conference on Systems and Storage, SYSTOR 2023, Haifa, Israel, June 5-7, 2023*, pages 1–7. ACM, 2023.

[41] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, et al. Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction. In *2016 First*

International Workshop on Communication Optimizations in HPC (COMHPC), pages 1–10. IEEE, 2016.

[42] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, 2019.

[43] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. RDMA over commodity ethernet at scale. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 202–215. ACM, 2016.

[44] T. Hayashi, R. Yamamoto, K. Inoue, T. Yoshimura, S. Watanabe, T. Toda, K. Takeda, Y. Zhang, and X. Tan. Espnet-tts: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 7654–7658. IEEE, 2020.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[46] 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. *CoRR*, abs/1901.00041, 2019.

[47] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 947–960. USENIX Association, 2019.

[48] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 463–479. USENIX Association, 2020.

[49] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 1–12. ACM, 2017.

[50] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostic, Y. Kwon, S. Peter, and E. Witchel. Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 756–771. ACM, 2021.

[51] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013.

[52] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift. ATP: in-network aggregation for multi-tenant learning. In J. Mickens and R. Teixeira, editors, *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 741–761. USENIX Association, 2021.

[53] K. Lee, M. Lam, R. Pedarsani, D. S. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Trans. Inf. Theory*, 64(3):1514–1529, 2018.

[54] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella. PANIC: A high-performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 243–259. USENIX Association, 2020.

[55] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. ipipe: A framework for building distributed applications on multicore soc smartnics. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019.

[56] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana. E3: energy-efficient microservices on smartnic-accelerated servers. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA,*

USA, July 10-12, 2019, pages 363–378. USENIX Association, 2019.

[57] J. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5068–5076. IEEE Computer Society, 2017.

[58] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 289–304. USENIX Association, 2020.

[59] D. G. Murray, J. Simsa, A. Klimovic, and I. Indyk. tf.data: A machine learning data processing framework. *Proc. VLDB Endow.*, 14(12):2945–2958, 2021.

[60] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia. {Heterogeneity-Aware} cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 481–498, 2020.

[61] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[62] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[63] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 16–29, 2019.

[64] R. D. Pietro, F. Lombardi, and A. Villani. CUDA leaks: A detailed hack for CUDA and a (partial) fix. *ACM Trans. Embed. Comput. Syst.*, 15(1):15:1–15:25, 2016.

[65] Y. Qiu, Q. Kang, M. Liu, and A. Chen. Clara: Performance clarity for smartnic offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 16–22, 2020.

[66] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis. Infaas: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 397–411. USENIX Association, 2021.

[67] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik. Scaling distributed machine learning with in-network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 785–808. USENIX Association, 2021.

[68] X. Schepler, A. Rossi, E. Gurevsky, and A. Dolgui. Solving robust bin-packing problems with a branch-and-price approach. *European Journal of Operational Research*, 297(3):831–843, 2022.

[69] J. Skolnick, M. Gao, H. Zhou, and S. Singh. Alphafold 2: Why it works and its implications for understanding the relationships of protein sequence, structure, and function. *J. Chem. Inf. Model.*, 61(10):4827–4831, 2021.

[70] D. Z. Tootaghaj, A. Mercian, V. Adarsh, M. Sharifian, and P. Sharma. Smartnics at edge for transient compute elasticity. In *Proceedings of the 3rd International Workshop on Distributed Machine Learning*, pages 9–15, 2022.

[71] M. Tork, L. Maudlej, and M. Silberstein. Lynx: A smartnic-driven accelerator-centric architecture for network servers. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 117–131. ACM, 2020.

[72] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer. A survey on distributed machine learning. *ACM Comput. Surv.*, 53(2):30:1–30:33, 2021.

[73] Z. Wang, H. Lin, Y. Zhu, and T. S. E. Ng. Hi-speed DNN training with espresso: Unleashing the full potential of gradient compression with near-optimal usage strategies. In G. A. D. Luna, L. Querzoni, A. Fedorova, and D. Narayanan, editors, *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*, pages 867–882. ACM, 2023.

[74] Z. Wang, X. Wu, Z. Xu, and T. Ng. Cupcake: A compression scheduler for scalable communication-efficient distributed training. *Proceedings of Machine Learning and Systems*, 5, 2023.

[75] Z. Wang, Z. Xu, A. Shrivastava, and T. Ng. Zen: Near-optimal sparse tensor synchronization for distributed dnn training. *arXiv preprint arXiv:2309.13254*, 2023.

[76] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen. Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004, Boston, MA, July 2023. USENIX Association.

[77] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 945–960. USENIX Association, 2022.

[78] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[79] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang, and L. Zhou. Fast distributed deep learning over RDMA. In *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, pages 44:1–44:14. ACM, 2019.

[80] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[81] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. M. Lau, Y. Wang, Y. Xiong, and B. Wang. Hived: Sharing a GPU cluster for deep learning with guarantees. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 515–532. USENIX Association, 2020.

[82] M. Zhao, N. Agarwal, A. Basant, B. Gedik, S. Pan, M. Ozdal, R. Komuravelli, J. Pan, T. Bao, H. Lu, S. Narayanan, J. Langman, K. Wilfong, H. Rastogi, C. Wu, C. Kozyrakis, and P. Pol. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, pages 1042–1057. ACM, 2022.

[83] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1059–1068. ACM, 2018.

[84] Z. Zhou, W. Diao, X. Liu, Z. Li, K. Zhang, and R. Liu. Vulnerable GPU memory management: Towards recovering raw data from GPU. *Proc. Priv. Enhancing Technol.*, 2017(2):57–73, 2017.