Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters

Jayashree Mohan, Amar Phanishayee, and Janardhan Kulkarni, *Microsoft Research;*
Vijay Chidambaram, *The University of Texas at Austin and VMware Research*

This paper is included in the Proceedings of the
16th USENIX Symposium on Operating Systems
Design and Implementation.

July 11–13, 2022 • Carlsbad, CA, USA

Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters

Jayashree Mohan[*][*], Amar Phanishayee[*], Janardhan Kulkarni[*], Vijay Chidambaram[†][‡]
[*]*Microsoft Research* [†]*University of Texas at Austin* [‡]*VMware Research*

Abstract

Training Deep Neural Networks (DNNs) is a popular workload in both enterprises and cloud data centers. Existing schedulers for DNN training consider GPU as the dominant resource and allocate other resources such as CPU and memory proportional to the number of GPUs requested by the job. Unfortunately, these schedulers do not consider the impact of a job's sensitivity to allocation of CPU and memory resources. In this work, we propose Synergy, a resource-sensitive scheduler for shared GPU clusters. Synergy infers the sensitivity of DNNs to different resources using optimistic profiling; some jobs might benefit from more than the GPU-proportional allocation and some jobs might not be affected by less than GPU-proportional allocation. Synergy performs such multi-resource workload-aware assignments across a set of jobs scheduled on shared multi-tenant clusters using a new near-optimal online algorithm. Our experiments show that workload-aware CPU and memory allocations can improve average job completion time by upto $3.4\times$, by better utilizing existing cluster resources, compared to traditional GPU-proportional scheduling.

1 Introduction

The widespread popularity of Deep Neural Networks (DNNs) makes training such models an important workload in both enterprises and cloud data centers. Training a DNN is resource-intensive and time-consuming. Enterprises typically setup large multi-tenant clusters, with expensive hardware accelerators like GPUs, to be shared by several users and production groups [31, 56]. In addition to the model-specific parameters and scripts, jobs specify their GPU demand before being scheduled to run on available servers. Jobs are scheduled and managed either using traditional big-data schedulers, such as Kubernetes [10] or YARN [51], or using modern schedulers that exploit DNN job characteristics for better performance and utilization [11, 26, 33, 35, 42, 46, 55]. These DNN schedulers decide how to allocate GPU resources to many jobs while implementing complex cluster-wide scheduling policies to optimize for objectives such as average job completion times (JCT), makespan, or user-level fairness.
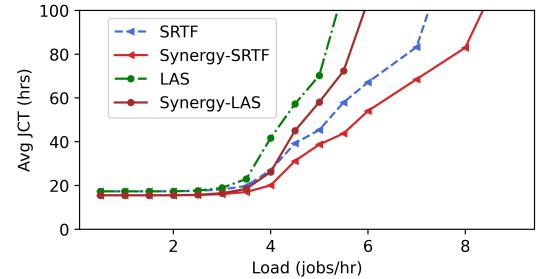

Figure 1: **Average JCT with Synergy**. Synergy is able to significantly reduce average JCT and support higher load for different scheduling policies (shown here on a cluster of 128 GPUs for a Philly-derived trace as we vary load [5]).

Current DNN cluster schedulers assume GPUs to be the dominant resource in the scheduling task [11, 26, 31, 33, 35, 42, 46, 55]; i.e., a user requests a fixed number of GPUs for her DNN job, and when the requested number of GPUs are all available, the job is scheduled to run. Other resources such as CPU and memory are allocated proportional to the number of GPUs assigned to the job (*GPU-proportional* allocation).

However, we identify an important property of DNN training jobs that GPU-proportional allocation is unable to exploit: DNNs exhibit varied sensitivity to the amount of auxiliary resources like CPU and memory allocated to the job. Prior work has shown that ingesting data for ML training jobs, i.e., reading data from storage to memory, and pre-processing them at the CPU is computationally expensive, thereby resulting in *data stalls* in both research [39] and industry scale training at large enterprises such as Google [40] and Facebook [59]. For instance, some image and video recognition models achieve up to $3\times$ speedup by overcoming *data stalls* (§2) when the CPUs allocated exceed their GPU-proportional share, while other models like GNMT are unaffected when the CPUs assigned are less than GPU-proportional share.

Our main insight here is that allocating these auxiliary resources in a workload-aware fashion, rather than the traditional GPU-proportional allocation can significantly improve performance by effectively utilizing *cluster-wide* resources. Based on this insight, we propose *Synergy*, a resource-sensitive scheduler for homogeneous, multi-tenant GPU clusters. Figure 1 shows the average job completion time (JCT)

---

[*]Work done as a MSR intern in Project Fiddle.

in the cluster as we vary load, for two scheduling policies; Synergy's resource-sensitive allocation is able to significantly improve average JCT in the cluster and sustain a higher load compared to GPU-proportional allocation.

Synergy profiles the sensitivity of DNNs to auxiliary resources and allocates them disproportionately among jobs rather than using traditional GPU-proportional allocation. While doing so, Synergy ensures that a job gets less than GPU-proportional auxiliary resources *only* if such an allocation does not degrade the job throughput compared to a GPU-proportional allocation. Such allocation enables Synergy to mitigate data stalls in several models, thereby significantly increasing the overall cluster throughput.

Efficiently exploiting the heterogeneity in resource sensitivity among DNN jobs raises two important problems which have not been tackled by prior work:

- What is the ideal resource requirement for each job (with fixed GPU demand) and how can this be determined with low overhead?
- How should we pack these jobs onto servers along multiple resource dimensions efficiently, especially when we can tune the job's demand for these resources?

Optimistic profiling. Synergy exploits the predictability of DNN computation to measure the job throughput as we vary the amount of CPU and memory allocated to the job. This is performed offline by the Synergy scheduler, prior to job execution on the cluster. However, profiling all possible combinations of CPU, and memory values is computationally expensive. Therefore, Synergy introduces optimistic profiling; it empirically profiles the job throughput for varying CPU allocations, assuming maximum memory allocation. It then *analytically* estimates the job throughput for all combinations of CPU and memory.A key insight that makes such analytical modelling feasible is the predictable nature of job performance to memory allocation when using DNN-aware caching like MinIO [39] that guarantees a certain cache hit rate. We show in §3.1 that our optimistically profiled model performance closely resembles the true empirical values, while significantly reducing profiling time (by up to 30×). Using these profiles, Synergy identifies the best resource allocation beyond which the job throughput has diminishing returns.

Scheduling mechanism. Synergy makes a round-based scheduling decision similar to prior DNN schedulers [42]. In each round (say 5 minutes), we identify the set of jobs that are runnable in the cluster using a scheduling policy such as FIFO [51,57], SRTF [12], LAS [26,43], FTF [35], etc. Synergy's scheduling mechanism then packs these jobs among available servers in the cluster along all resource dimensions identified in the profiling phase. This is analogous to multi-dimensional bin-packing problem, which is NP-Hard [53], and hence requires approximate solutions. But unlike prior work in big-data scheduling which tackles the problem of multi-dimensional bin-packing with fixed resource demands
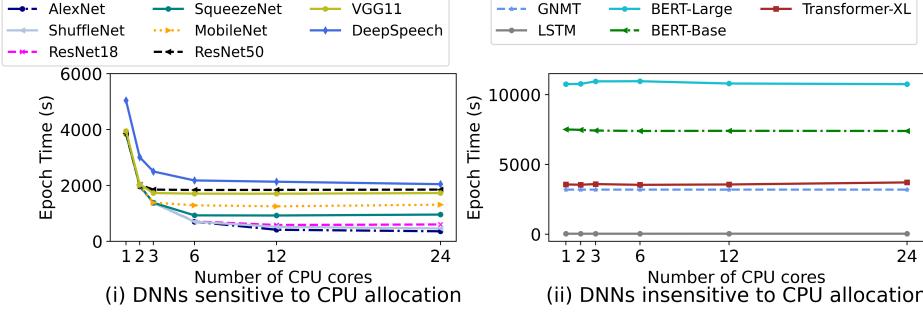
(for *e.g.,* Tetris [23], DRF [21]), Synergy has to contend with fungible resource demands. This introduces two challenges that need to be solved in tandem: First to find an optimal partition of CPU and memory among jobs to maximize throughput while ensuring fair allocations (every job's throughput is at least that of GPU-proportional allocation), and second, a feasible packing of these resources among jobs.

In this paper, we propose two effective algorithms to enable such fungible multi-dimensional bin-packing. Our first algorithm, Synergy-OPT, is formulated as a linear program and enables determining an upper-bound on achievable throughput by an optimal solution for a given workload trace. However, we find that Synergy-OPT is impractical for two reasons: (1) it is computationally expensive as we scale cluster size, and (2) it produces fractional GPU allocations that cannot be achieved in real deployments. Nevertheless, its solution provides an aspirational optimal goal that we can use to measure the efficacy of any practical solution. The second algorithm, Synergy-TUNE, is fast and near-optimal (within 10% of Synergy-OPT in evaluation). If a job to be scheduled does not fit in the cluster along all the resource dimensions, we revert the job demands to GPU-proportional if its current demands are above it. If the job's demands are already GPU-proportional or below, then we find a suitable job in the cluster with higher than GPU-proportional allocation, which is then reverted to GPU-proportional. Synergy-TUNE also outperforms simpler greedy approaches (Synergy-GREEDY) that recursively pack jobs along multiple resource dimensions using a first-fit allocation strategy [20].

We implement a prototype of Synergy and an accompanying event-driven simulator in Python. Synergy transparently communicates with the DNN job using a thin iterator API, that is a wrapper around the existing data iterator, thereby requiring minimal code changes to the DNN job script. Across various scheduling policies, and workload traces, we show that Synergy improves cluster objectives such as average JCT by up to 1.5× on a physical cluster of 32 GPUs. On a large simulated cluster of up to 512 GPUs, Synergy improves average JCT by up to 3.4×. Synergy is open sourced at https://github.com/msr-fiddle/synergy.

In summary, our paper makes the following contributions.

- We identify the importance and need for resource-sensitive scheduling of DNN jobs in multi-tenant GPU clusters (§2).
- We present Synergy, a resource-sensitivity aware scheduler that optimistically profiles the job's resource demands and performs disproportionate allocations such that no job achieves lower than GPU-proportional throughput (§3).
- We present a heuristic scheduling mechanism Synergy-TUNE, that maps the allocations calculated by the profiler onto the cluster, while better utilizing the resources compared to a GPU-proportional allocation (§4).
- In extensive experimentation on physical and simulated clusters, Synergy's techniques improve average JCT by up to 3.4×, thus supporting a higher input load (§5).

Figure 2: **CPU sensitivity**. This graph plots the epoch time for DNNs as we vary the CPU:GPU ratio for single-GPU training. Some jobs such as Transformers need as few as 1 CPU core per GPU to achieve maximum training speed; others like ShuffleNet need more than 12 CPU cores per GPU to eliminate data stalls. State-of-the-art GPU VMs have a CPU:GPU ratio as few as 3.

2 Background and Motivation

In this section, we briefly describe DNN scheduling, introduce the terminology used in the rest of the paper, and motivate resource-sensitive DNN cluster scheduling.

Scheduling ML training jobs in a cluster. Training a ML model is a resource intensive and long-running task (order of hours to days). Collocating ML training workloads in a shared, multi-tenant cluster is a very common setup in several large organizations, for both research and production [26, 35, 42, 46, 55]. Our work targets state-of-the-art multi-tenant clusters similar to the ones published by prior large-scale studies by organizations like Microsoft [31] and Alibaba [56]. These clusters use on-premise servers or cloud VMs with pre-defined GPU, CPU, and memory resources. The cluster itself is shared by multiple users and jobs, and each server can host more than one job each with varying resource usage (some heavy on CPU side pre-processing, while others heavy on GPU computation). For example, a server with 8 GPUs can host 8 single-GPU jobs from different users.

Scheduling policy and mechanism. When jobs are submitted to a scheduler, a scheduling policy such as First In, First Out (FIFO) [51, 57], Shortest Remaining Time First (SRTF) [12], Least Attained Service (LAS) [26, 43], or Finish Time Fairness (FTF) [35] decides the set of jobs (J) to be run on the cluster. A scheduling mechanism then identifies where job J should be run, and how much resources to allocate to the job. The GPU demand for a job is fixed (requested by the user), while the CPU and memory allocation is fungible.

GPU-proportional allocation. During DNN training, a mini-batch of data is first fetched from storage to memory, where it is cached for subsequent accesses. It is then pre-processed at the CPU, and then copied over to the GPU for processing. Existing DNN schedulers [26, 35, 42, 55], and those used in real-world GPU clusters [5, 31], including recent schedulers that offer GPU elasticity [30, 48], all allocate CPU and mem-

ory resources to a job using a GPU-proportional allocation. For instance, consider a server with 4 GPUs, 16 CPUs and 200 GB memory. If a job requests 1 GPU, then it is allocated 4 CPUs and 50GB memory.

2.1 Motivation : Resource sensitivity

Insight. The main insight that motivates our work is that DNNs co-scheduled on a cluster exhibit different levels of sensitivity to CPU and memory allocations during training. Therefore, it is possible to improve the overall cluster utilization and efficiency by performing resource-sensitive allocations instead of the ubiquitously used GPU-proportional allocation. Prior work on characterization study of jobs in Microsoft's Philly cluster [31] shows that CPU cycles are under-utilized in multi-tenant clusters; we use this as motivation to show that we can *exploit the disparity in resource requirements **across jobs** to improve overall cluster utilization without any hardware upgrades (storage, CPU, or memory)*.

Figure 2a plots the per-epoch time for various DNNs when trained on a single GPU by varying the number of CPUs allocated to the job (ensuring that the dataset is fully cached for each job). Figure 2a(i) shows that most image and speech models are sensitive to CPU allocations; smaller models like ShuffleNet and ResNet18 require 9–24 CPU cores per GPU to pre-process data items. However, state-of-the-art ML optimized servers and cloud GPU VMs have a CPU:GPU ratio as few as 3 as shown in Table 2b [1–3, 6, 18, 34]. Increasing the CPU:GPU ratio from 3 to 12 results in $3.1\times$ faster training for AlexNet, and increasing it to 9 results in $2.3\times$ faster training for ResNet18. On the other hand, most language models are insensitive to CPU allocations as shown in Figure 2a(ii). This is because they have modest input data pre-processing requirements. Transformer models for example, unlike image classification models, do not perform several unique data augmentation operations for each data item in every epoch [39].

Next, to understand the importance of memory alloca-

Job	Model
J_1	ResNet18
J_2	Audio-M5
J_3	Transformer
J_4	GNMT

Table 1: Example jobs

Server	Job	GPU	CPU	Mem
S_1	$J1$	4	12	250
	$J2$	4	12	250
S_2	$J3$	4	12	250
	$J4$	4	12	250

Table 2: GPU-proportional allocation

| Server	Job	GPU	CPU	Mem
S_1	$J1$	4	23	400
	$J3$	4	1	100
S_2	$J2$	4	12	450
	$J4$	4	12	50

Table 3: Resource-sensitive allocation
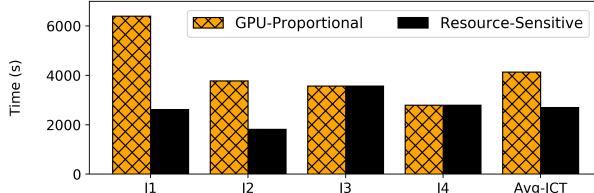
Figure 3: **Resource sensitive scheduling**. We compare the runtime of the jobs with two different schedules; GPU-proportional and resource-sensitive. By allocating resources disproportionately, CPU and memory sensitive jobs see increased throughputs which reduces the average JCT by $1.5\times$.

tions, we train two models; an image classification model - ResNet18 on OpenImages [22] and a language model GNMT on WMT, with varying memory allocations on a server whose GPU-proportional share of memory per GPU is 62GB. We observe that GNMT is insensitive to memory allocation; even if only 20GB memory is allocated (which is the required process memory for training), the training throughput is unaffected. However, increasing the memory from 62GB (GPU-proportional allocation) to 500GB (max) for ResNet18 speeds up training by almost $2\times$. This is because, language models like GNMT, and transformers are GPU compute bound. Therefore, fetching data items from storage if they are not available in memory does not affect training throughput. On the other hand, image and speech models benefit from larger DRAM caches. If a data item is not cached, the cost of fetching it from the storage device can introduce fetch stalls in training [39, 40, 59].

Takeaway. *When two jobs have to be scheduled on the same server, it is possible to co-locate a CPU-sensitive job with a CPU-insensitive one. This allows CPU allocation to be performed in a resource-sensitive manner rather than GPU-proportional allocation. Similarly, it is always beneficial to pack a memory-sensitive job with an insensitive one, allowing disproportionate resource-sensitive sharing of memory to improve the aggregate cluster throughput.*

Example. We now show how resource-sensitivity-aware scheduling can improve cluster efficiency using a simple example. We run the experiment on two physical servers each with 8 GPUs, 24 CPUs and 500GB DRAM (internal servers at a large cloud provider X). Let's say we have 4 jobs in the scheduling queue, each requesting 4 GPUs as shown in Table 1. We consider two different schedules; (1) GPU-proportional allocation and (2) resource-sensitive allocation. The results of these schedules are shown in Table 2 and Ta-

ble 3. Figure 3 compares the epoch time of each of these jobs in the two scenarios. The increased resource allocation to CPU and memory sensitive jobs in Schedule 2 speeds up $J1$ and $J2$ significantly, while leaving the runtime of $J3$ and $J4$ unaffected. The average JCT in the cluster thus drops by $1.5\times$ due to resource-sensitive allocations.

2.2 Synergy Scheduling Policies

Synergy is not constrained to one particular scheduling policy, but is instead general enough to improve a wide range of scheduling policies (*e.g.,* LAS, FIFO, SRTF, FTF, etc), creating Synergy-augmented variants for all of them. The main challenge that Synergy addresses is, finding an efficient partition of available cluster CPU and memory among jobs to maximize throughput while ensuring that every job's throughput is at least that of GPU-proportional allocation. Synergy 's innovation thus lies in exploiting the differences in resource sensitivity across jobs to improve overall cluster metrics.

2.3 Assumptions & Limitations

In the context of this work, we explicitly highlight certain practical assumptions, many of which are derived directly from large multi-tenant clusters we analyze - homogeneous clusters, fixed GPU allocation for the lifetime of a job, and the use of MinIO cache. Synergy 's design is not tied to these assumptions, but it aids in focused profiling (reducing the dimensionality of the search space). In a large scale, multi-tenant, production cluster, it is practical to assume that there are tens of thousands of accelerators per homogeneous cluster, and the GPU allocation for a job remains constant. While recent works explore scheduling DNN jobs in heterogeneous clusters [11, 33, 42], and GPU elasticity [48], there are several practical challenges in seamlessly supporting these features. For instance, with elastic training, the impact of changing batch sizes and hyperparameters on training accuracy is unclear for a wide variety of tasks. We provide a detailed discussion on the practicality of each of these assumptions made by Synergy, and what it means to relax these assumptions for Synergy in Section 6.

3 Synergy: Design

Overview. Synergy is a round-based scheduler that arbitrates multi-dimensional resources (GPU, CPU, and memory) in a
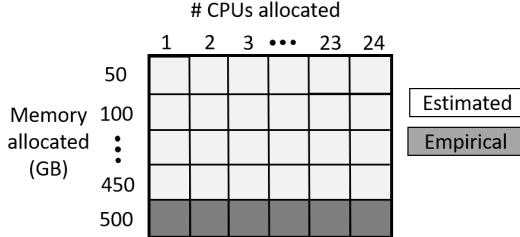
Figure 4: **Optimistic profiling** empirically evaluates the sensitivity of a model to varying # CPUs assuming a fully cached dataset; the rest of the matrix is completed using estimation

homogeneous cluster. Synergy augments existing scheduling policies with *resource sensitivity* in two steps. First, it identifies the job's best-case CPU and memory requirements using *optimistic profiling* (§3.1). Synergy then identifies a set of runnable jobs for the given round using a scheduling policy (*e.g.,* SRTF, FTF, LAS, etc) such that their collective GPU demand is less than or equal to the GPUs available in the cluster. Then, using the profiled resource demands, Synergy packs these jobs on to the available servers along multiple resource dimensions using a near-optimal heuristic algorithm (§4). At the end of a round, the set of runnable jobs are updated using the scheduling policy, and their placement decisions are recomputed. We now discuss both the components of Synergy in detail. Note that Synergy only alters the auxiliary resource allocations; GPU demands are left unaltered for the lifetime of a job and are provided as inputs by the user.

3.1 Optimistic Profiling

A DNN job is profiled for its resource sensitivity once per lifetime of the job, i.e. on job arrival. Each incoming job is profiled by varying the CPU and memory allocated to the job. A *resource sensitivity matrix* is then constructed for discrete combinations of CPU and memory allocations as shown in Figure 4. Since DNN training has a highly predictable structure, empirically evaluating training throughput for a few iterations gives a fair estimate of the actual job throughput [39, 55].

It is easy to see that naively profiling different combinations of CPU and memory can be very expensive. For instance, if the cost of profiling one combination of CPU, and memory for a job is 1 minute, then to profile all discrete combinations of CPU and memory (assuming allocation in units of 50GB) on a server with 24 CPUs and 500GB DRAM takes about 24*10 = 240 minutes (4 hours)!
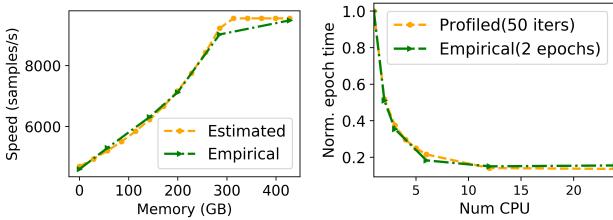
To tackle this problem, Synergy introduces an optimistic profiling technique that exploits the predictability in the relationship between job throughput and memory allocation. We observe that, with DNN-specific, application-level caches like MinIO [39], it is easy to model the job throughput behaviour as we vary the amount of memory allocated to a job at fixed CPU allocation. This is because, MinIO ensures that

a job gets a fixed number of cache hits per epoch. Synergy makes a conscious decision to use application-level MinIO cache instead of Page Cache because MinIO provides memory isolation across independent jobs sharing the machine. If we do not use MinIO, we will have to profile the model at discrete memory allocations which could result in increased profiling costs, and also potentially change the trends in profiling matrix. However, the use of MinIO in Synergy makes cache performance predictable and hence reduces Synergy 's profiling costs – allowing optimistic profiling.

For a given CPU allocation that determines the pre-processing speed, and a known storage bandwidth, it is easy to analytically model the job throughput for varying memory allocation. Therefore, we only need to empirically profile the job for varying CPU values at full memory allocation as shown in Figure 4. All the other entries can be estimated using the above technique. This leads to a $10\times$ reduction in profiling time, bringing it down to 24 minutes! We experimentally validate this in Figure 5a. For a 8-GPU ResNet18 job, we compare the modeled job throughput using Synergy to the empirical results obtained by training the job for 2 epochs with varying memory allocations. As we see in Figure 5a, Synergy's estimations are within 3% of the empirical results, without having to actually run the model.

To further optimize profiling time, we observe that we do not require exact throughput values for a job with varying CPU allocations. We instead need a curve depicting the empirical job throughput. Therefore, instead of profiling the job for all possible CPU values, we pick discrete points for CPU profiling using the following algorithm. We start with the maximum CPU allocation and do a binary search on the CPU values to estimate job throughput. If the profiled point resulted in a throughput improvement that is less than a fixed threshold (say 10%), then we continue binary search on the lower half of CPU values, else we profile more points on the upper half. The idea here is to empirically profile CPU regions that show significant difference in job throughput, while skip those regions with little to no improvement in throughput. We experimentally show the efficacy of our CPU profiling technique in Fig 5b for a 1-GPU ResNet18 job. We compare the normalized job runtime (wrt 1 CPU) using empirical results averaged over 2 epochs of the job and Synergy's optimistic profiling averaged over 50 iterations (approximately, a minute per profile). Synergy is able to mimic the empirical job performance very closely, in under 8 minutes (using just 8 CPU profile points instead of 24). We believe that this is a reasonable overhead as it is incurred only once per lifetime of the job, which typically runs for hours.

After profiling a job on arrival, the job along with its resource sensitivity matrix is enqueued into the main scheduling queue, from which the scheduling policy picks a set of runnable jobs every round.

(a) Memory Validation (b) CPU validation

Figure 5: **Optimistic profiling**. The graphs compare the profiling results to empirical runs for ResNet18

3.2 Scheduling mechanism

Synergy performs round-based scheduling. At the beginning of each scheduling round, Synergy identifies a set of runnable jobs from the scheduling queue that can be packed on the cluster in the current round duration using a scheduling policy such as FIFO, SRTF, LAS, or FTF. Using the resource sensitivity matrix, Synergy packs these jobs onto the available servers in the cluster while satisfying the multi-dimensional resource constraints as opposed to simply performing a GPU-proportional allocation .

Job demand vector. To pack the jobs onto servers, we first construct a job demand vector that indicates the GPU demand, and best-case CPU and memory requirements for the job. We identify the best-case values using the resource sensitivity matrix. We pick the minimum value of CPU and memory that saturates the job throughput.

Packing a job with multi-dimensional resource demands is analogous to multi-dimensional bin packing problem which is NP hard [53]. Therefore, we first evaluate the efficacy of a naive greedy scheduling mechanism as an approximation to tackle the multi-dimensional resource allocation problem.

3.3 Synergy-GREEDY: Greedy Scheduling

A naive greedy multi-resource packing algorithm translates to a first-fit approximation of the multi-dimensional bin packing problem [20]. Given a job demand vector, the greedy algorithm picks the next runnable job decided by the scheduling policy, and places it on the server that can satisfy the job's demands in all dimensions. If no such server exists, the job is *skipped* over for this round and the next runnable job is checked for schedulability. Synergy-GREEDY thus introduces two major problems in the cluster -

- It can result in auxiliary resources being exhausted by jobs, while leaving GPUs underutilized, and fragmented. We show that GPU fragmentation in Synergy-GREEDY severely degrades cluster objectives (5.4).
- It also hurts the fairness of the scheduling policy as some jobs can be skipped over for a long time if their resource demands cannot be satisfied in the cluster.

The challenge ahead of us is to design a scheduling mechanism that eliminates GPU under-utilization due to fragmentation, and upholds the fairness properties of the given scheduling policy, while performing multi-dimensional resource allocation. Before we come up with a heuristic scheduling approach to tackle the above problems, one pertinent question is to understand how good is the allocation produced by our heuristic when compared to an optimal solution.

To this end, we first formulate a theoretical upper bound on the optimal throughput achieved by the cluster given a set of jobs and their resource sensitivity profiles. We then discuss the challenges associated with materializing the optimal allocation on a physical cluster and introduce Synergy-TUNE, an empirically close-to-optimal heuristic solution.

4 Scheduling Algorithms

We first present our formulation of an optimal allocation that provides an upper bound on the achievable cluster throughput.

4.1 Synergy-OPT

Our goal is to allocate CPU and memory to each job so as to maximize overall throughput, while guaranteeing that each job makes at least as much progress as it would do if we allocate its *GPU-proportional share*. It is not hard to show that our problem is NP-hard. So, we resort to finding approximate solutions using LP formulation. To find an *upperbound* on achievable throughput, we solve two LPs. In the interest of space, we describe the first LP formulation here, and summarize the challenges in operationalizing Synergy-OPT. A complete description of Synergy-OPT formulation and proof can be found in the extended version [38]. While the focus of this work is on homogeneous cluster, we show how our formulation can be extended to a heterogeneous GPU cluster in the extended version of the paper [38].

4.1.1 Finding ideal allocation

First, we assume an *idealized setting*: all the CPU and memory available across all the machines is present in one (super) machine. Say there are a total of s homogeneous machines in the cluster. We assume that, there is only one machine with G units of GPU, C units of CPU, and M units of memory. Note that, in reality G_i, C_i, and M_i denote the total GPU, CPU, and memory in each machine i, which is G/s, C/s, and M/s respectively in a homogeneous cluster. Based on this assumption, we find the ideal CPU (c_j^*) and memory (m_j^*) allocation for every job j (whose GPU demand is denoted by g_j) in the set of runnable jobs (J_t) for a round.

The variables of our LP are denoted by $y_{\{c,m,j\}}$, which should be interpreted as follows. If for a job $j \in J_t$, $y_{\{c,m,j\}} = 1$, then it means that in the LP solution c units of CPU and m units of memory are allocated. We further note that for

every job j, there is a variable $y_{\{c,m,j\}}$ for *for every possible* allocation of CPU and memory. We consider these variables in the discrete space as identified by our resource sensitivity matrix (W_j). $W_j[c,m]$ denotes the amount of progress made by job j per round if c units of CPU and m units of (RAM) memory are allocated to job j. For each machine $i \in [s]$, we denote C_g, M_g as the GPU-proportional allocation of CPU and memory. That is, $C_g = C_i/G_i * g_j$ and $M_g = M_i/G_i * g_j$. With a baseline GPU-proportional allocation strategy the progress a job makes in each round is equal to $W[Cg, Mg]$.

Our objective function is to maximize the throughput. We formulate it as follows using our LP variables.

$$\text{Maximize} \quad \sum_{j \in J_t} \sum_{[c,m]} W_j[c,m] \cdot y_{\{c,m,j\}} \quad (1)$$

Now, we enforce constraints such that LP solution is feasible in the idealized setting we talked about.

- Total CPU and memory allocated to jobs is no more than the total capacity available:

$$\sum_{j \in J_t} \sum_{[c,m]} c \cdot y_{\{c,m,j\}} \leq C \quad (2)$$

$$\sum_{j \in J_t} \sum_{[c,m]} m \cdot y_{\{c,m,j\}} \leq M \quad (3)$$

- We want the LP to allocate only one configuration of CPU and memory to each job.

$$\forall j \in J_t: \quad \sum_{[c,m]} y_{\{c,m,j\}} = 1 \quad (4)$$

- LP solution is atleast as good as the fair allocation.

$$\forall j \in J_t: \quad \sum_{[c,m]} W_j[c,m] \cdot y_{\{c,m,j\}} \geq W_j[C_g, M_g] \quad (5)$$

Theorem 4.1. Throughput achieved by LP(1-5) is at least the throughput achieved by an optimal solution to our problem.

Proof. Consider an optimal solution O to our problem. Suppose job j receives c^* units of CPU and m^* units of memory in O. Then we define the following feasible solution to our LP (1-5): Set $y_{c^*,m^*,j} = 1$. Clearly, this is a valid solution and satisfies constraints (1-4). □

In our experiments, we solve this as a Integer Linear Program (ILP) where $y_{\{c,m,j\}}$ takes boolean values. For every job, we define the total CPU (c_j^*) and memory (m_j^*) allocated by the optimal ILP solution as follows.

$$\text{For each job } j, \text{ define} \quad c_j^* := c \quad \text{if} \quad y_{\{c,m,j\}==1}. \quad (6)$$

$$\text{and} \quad m_j^* := m \quad \text{if} \quad y_{\{c,m,j\}==1}. \quad (7)$$

4.1.2 Feasible Allocation on Multiple Machines

Recall that in the LP(1-5), we assumed that all the resources are present on a single machine. In reality, since these resources are spread across machines, we find a feasible allocation on multiple machines by solving a second LP. The objective here is to minimize the number of jobs that get fragmented to account for the communication overhead when jobs are split across machines. The variables of the second LP are denoted by $x_{i,j}$. Here index i denotes the machine and j denotes the job. If $x_{i,j} = 1$, it means that resources of job j (that g_j units of GPU, c_j^* units of CPU, and m_j^* units of memory) are allocated on machine i. Note that $x_{i,j}$ can be fractional; if so, then job j is split across multiple machines. We can prove that the solution to the second LP ensures that the total number of jobs that get fragmented is at most $3s$. Detailed formulation is in the extended version of the paper [38].

4.1.3 Challenges with operationalizing Synergy-OPT

While the allocations identified by Synergy-OPT provides an upper bound on the optimal cluster throughput, it is challenging to materialize these allocations in the real world due to two main reasons;

- Solving two LPs per scheduling round is computationally expensive. As cluster size and the number of jobs per round increases, the time to find an optimal allocation increases exponentially (§5.6)
- The allocation matrix obtained with the second LP can result in fractional GPU allocations when jobs are split across servers; for instance, a valid allocation might assign 3.3 GPUs on server 1 and 2.7 GPUs on server 2 for a 6 GPU job. Realizing such an allocation requires a heuristic rounding off strategy to ensure non-fractional GPU allocations, as GPU time or space sharing, and its impact on job performance is considered beyond the scope of this work.

4.2 Synergy-TUNE

We now describe Synergy-TUNE, our heuristic scheduling mechanism. Our goal is to design a scheduling mechanism that performs multi-dimensional resource allocation for DNN jobs, where the GPU demand is fixed, but the auxiliary resource allocations are fungible. In doing so, we want to ensure that (1) we do not affect the fairness properties of the scheduling policy used, (2) the expensive GPU resources are not underutilized.

Allocation Requirements. Synergy-TUNE's allocation must satisfy the following requirements.

- The GPU, CPU, and memory resources requested by a single-GPU job must all be allocated on the same server.
- A multi-GPU distributed-training job can either be consolidated on one machine, or split across multiple machines.

In the latter case, *the CPU and memory allocations must be proportional to GPU allocations across servers*. For instance, if a job requires (2GPU, 12 CPU, 300GB DRAM), then while splitting it across two servers, we need to ensure that each server gets (1GPU, 6CPU, 150GB DRAM). This is because, multi-GPU jobs train on a separate process on each GPU, and synchronize at regular intervals, i.e., after one or many iterations. The job performance will vary across processes if each GPU does not get the same ratio of resources, and will eventually proceed at the speed of the process with the lowest allocation of CPU and memory.

In a multi-tenant cluster, while carving out resources such as CPUs and memory for jobs, it is import to enforce fairness in terms of throughput achieved by individual jobs. We need to ensure that no job runs at a throughput lower than what it would have achieved if we allocated a GPU-proportional share of CPU and memory resources. Additionally, we need to respect the priority order of jobs identified by the scheduling policy. For instance, a FIFO scheduling policy can be implemented using a priority queue sorted by job arrival times. Synergy-TUNE identifies a set of runnable jobs for a round as the top n jobs from the scheduling queue, whose GPU demands can be exactly satisfied by the available servers in the cluster. Synergy-TUNE picks this runnable job set irrespective of the job's other resource demands - which are fungible. Note that, unlike Synergy-GREEDY, we do not skip over any jobs unless it cannot be scheduled (GPU demand cannot be met). Therefore, we never underutilize the GPUs when the cluster is at full load.

Next, Synergy-TUNE greedily packs each of these runnable jobs along multiple resource dimensions on one of the available servers, with the objective of minimizing fragmentation. To achieve this, Synergy-TUNE sorts the runnable jobs by their GPU demands, followed by CPU, and memory demand. For each job j in order, Synergy-TUNE then picks the server with the least amount of free resources just enough to fit the demand vector of j. If it is a multi-GPU job, then we find a minimum set of servers with sufficient GPU availability that can fit the job's demands in entirety. However, it is possible that the job cannot fit in the cluster along all dimensions. In such a case,

1. We check if the job's demand vector is greater than proportional share of resources, In this case, we switch the job's demand to GPU-proportional share and retry.

2. If the job still does not fit the cluster, or if the job's demand vector was less than or equal to GPU proportional allocation in step (1), then, we do the following.

  (a) We repeat step (1) ignoring the job's CPU and memory requirements. We find a server that can just satisfy the job's GPU requirements. We know by construction that there is atleast one job on this server, which is allocated more than GPU-proportional share of resources. We identify the job or a set of jobs (J_s) on this server by

switching whom to GPU-proportional share, we can release just as much resources required by the current job j. We switch the jobs in J_s to fair-share and by design, job j will fit this server.

  (b) We continue this recursively for all runnable jobs.

In the worst case, all the running jobs in a round could be allocated GPU-proportional share of resources. Therefore, Synergy ensures that its allocations results in job throughputs that are never worse than GPU-proportional allocation. In §5.6, we empirically compare Synergy-TUNE to Synergy-OPT showing that it is practical and near-optimal.

4.3 Implementation

We implement Synergy and an associated simulator in Python. Our scheduler is event-driven. There is a global event queue where job arrivals, schedule events, and deploy events are queued. These events are handled in the order of their arrival time. There is a priority job queue, where all the jobs arriving into the cluster are added, post profiling. This queue is sorted by the priorty metric decided by the scheduling policy; for instance, SRTF sorts the jobs in the order of job remaining time.

When a schedule event occurs, the scheduler collects a list of runnable jobs from the job queue and identifies the appropriate placement for these jobs for the following round, either using Synergy-GREEDY, Synergy-TUNE or Synergy-OPT. Then when a deploy event is triggered, these allocations are deployed on to the cluster. By default, every job requests for a lease update to continue running on the same server [42]. The scheduler then either grants a lease update or terminates the lease for the job, adding it back to the job queue.

The scheduler and the DNN jobs interact via a thin API provided by the Synergy data iterator. DNN job scripts must be updated to call the Synergy iterator which is a wrapper around the default PyTorch [8] and DALI [7] iterators. The iterator handles registering the job with the scheduler, and appropriately sending lease updates. It also checkpoints the job to a shared storage if its lease is terminated. The iterator also synchronizes across GPU processes for a multi-GPU job to ensure that each process makes identical progress. We use gRPC [4] to communicate between the scheduler and the jobs.

We implement Synergy-OPT in cvxpy [19] for use in our simulator. The optimistic profiling module is also implemented in Python, and it profiles the incoming jobs hooked to the Synergy iterator, prior to the job's initial addition to the scheduling queue (a one time overhead for each job).

5 Evaluation

In this section, we use trace-driven simulations from production cluster traces, and physical cluster deployment to evaluate

Task	Model	Dataset
Image	Shufflenetv2 [58] AlexNet [32] Resnet18 [28] MobileNetv2 [50] ResNet50 [28]	ImageNet [49]
Language	GNMT [54] LSTM [47] Transformer-XL [16]	WMT16 [9] Wikitext-2 [36] Wikitext-103 [36]
Speech	M5 [15] DeepSpeech [27]	Free Music [17] LibriSpeech [45]

Table 4: **Models used in this work**.

Policy (Metric)	Workload Split	Mechanism	Time (hrs)	
			Deploy	Simulate
FIFO (Makespan)	60-30-10	Proportional	16	15.67
		Tune	11.6	11.33
		Opt	-	11.01
SRTF (Avg JCT)	30-60-10	Proportional	4.81	4.52
		Tune	3.21	3.19
		Opt	-	3.06
SRTF (99 Percentile JCT)	30-60-10	Proportional	17.32	16.85
		Tune	8.59	8.54
		Opt	-	8.21

Table 5: **Physical cluster experiments**. This table compares the makespan, average JCT, and 99th percentile JCT for two different traces; (1) a static trace using FIFO (2) a dynamic trace using SRTF. Synergy-TUNE improves makespan by 1.4×, average JCT by 1.5 × and 99th percentile JCT by 2×.

the efficacy of Synergy. Our evaluation seeks to answer the following questions.

- Does Synergy's resource-sensitive scheduling improve cluster objectives such as makespan and average JCT in a physical cluster (§5.2) and in trace-driven simulations of large-scale clusters (§5.3) ?
- How does Synergy-TUNE and Synergy-GREEDY perform with different workload splits and how well do they utilize available resources (§5.4)?
- How does Synergy perform on different CPU:GPU ratios (§5.5)?
- Compare Synergy-TUNE to Synergy-OPT (§5.6)?
- Compare Synergy to big data schedulers (§5.7)?

5.1 Experimental setup

Clusters. Our experiments run on both a physical and a large simulated, homogeneous cluster. Our experiments are performed on state-of-the-art internal servers at Microsoft - these servers are part of a larger multi-tenant cluster. We run physical cluster experiments on a cluster with 32 V100 GPUs across 4 servers. Each server has 500GB DRAM, 24 CPU cores, and 8 GPUs. Unless otherwise specified, our experiments assume a CPU:GPU ratio of 3 and fair-share memory allocation of 62.5GB per GPU, matching the server configurations above. For simulations, we assume two cluster sizes; a 128 GPU cluster across 16 servers and a 512 GPU cluster across 64 machines, where each machine resembles the physical server configuration mentioned above.

Models. Our experiments consider 10 different DNNs (CNNs, RNNs, and LSTMs) as shown in Table 4. We categorize these models by task (image, language, and speech) and assign a certain weight to these tasks in our traces. We call this a workload *split*. For instance, if the split for a given trace is (30,40,30), then the percentage of image, language, and speech models in the job trace is 30%, 40% and 30% respectively. All experiments are performed on PyTorch 1.1.0.

Traces. We run our physical and simulated experiments using publicly available production traces from Microsoft Philly cluster [5].We show evaluation with the actual Philly trace preserving the job GPU demand, arrival time, and duration,

on a cluster of 512 GPUs in §5.3.1. We use a subrange of the trace containing 8000 jobs.

However, to comprehensively evaluate how Synergy reacts to varying cluster load, workload composition, and job duration, for all other experiments, we construct a production-derived trace as follows: we extract job GPU demand from the Philly trace and assign a model based on the chosen *split*. We then appropriately scale the job runtime and arrival time for the chosen cluster size, while keeping the job duration distribution similar to the one in Philly trace as follows:

- **Duration**. The duration of each job for the baseline GPU-proportional allocation is sampled from an exponential distribution: the job duration is set to 10^x minutes, where x is drawn uniformly from [1.5,3] with 80% probability, and from [3,4] with 20% probability similar to the trace duration used in prior work [42].
- **Arrival**. We classify derived traces into two kinds based on the job arrival time : (1) a *static* trace where all the jobs arrive at the start of the workload, and (2) a *dynamic* trace, where the job arrival time is determined by load, a Poisson distribution at a rate λ.

The derived traces with varying job arrival rates uses a 128 GPU cluster. In both cases, we report the average metrics such as JCT across a set of 1000 jobs in steady state.

For the physical cluster experiment, we choose a fixed arrival rate for the derived trace that keeps our cluster at *full load* (GPU demand of all runnable jobs > available GPUs in the cluster). For the simulated experiments, we vary the load λ on the cluster to evaluate its impact on cluster metrics. For the simulated experiments, we show results for two trace categories - (1) all jobs request single-GPU (2) multi-GPU distributed training jobs that request upto 16 GPUs.

Policies and metrics. We evaluate Synergy against GPU-proportional scheduling for 4 different scheduling polices; FIFO, SRTF, LAS, and FTF. For a static trace, we measure makespan (time to complete all jobs submitted at the beginning of the trace) and for the dynamic job traces, we measure

| | **Avg JCT(hrs)** | | |
Policy	**SRTF**	**LAS**	**FIFO**
GPU-prop.	30	32	71
Synergy	26	28	62
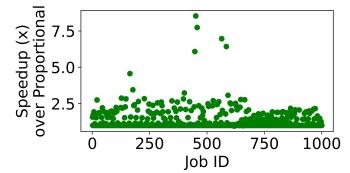
(a) Average JCT with Synergy

JCT (hrs)		**Short**	**Long**
Avg	Prop.	2	80
	Synergy	1.7	68
99p	Prop.	9	660
	Synergy	4	641
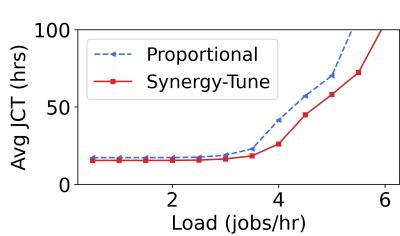
(b) Cluster metrics (SRTF)
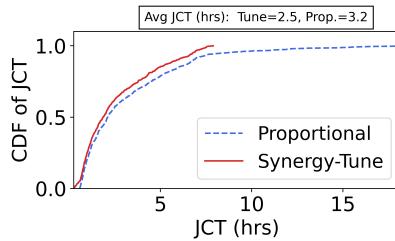
(c) JCT speedup across jobs
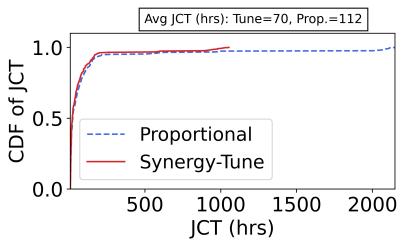
Figure 6: **Evaluation on Philly Trace**. On a real production trace, Synergy improves avg JCT across a range of scheduling policies over GPU-proportional scheduling. The JCT of individual jobs improves by upto 9× with Synergy.

(a) LAS (multi)
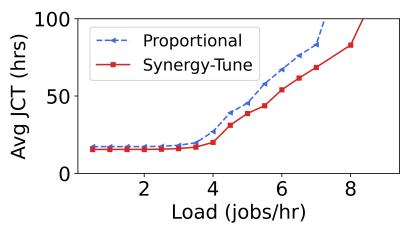
(b) CDF of JCT at load 4 (short)
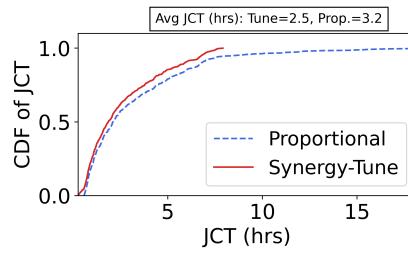
(c) CDF of JCT at load 4 (long)
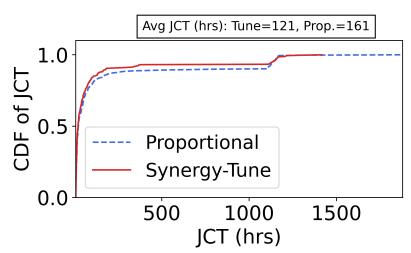
Figure 7: **Average JCT and CDF of long and short jobs for LAS policy**.

(a) SRTF (multi)

(b) CDF of JCT at load 5.5 (short)

(c) CDF of JCT at load 5.5 (long)

Figure 8: **Average JCT and CDF of long and short jobs for SRTF policy**.

the average job completion time (JCT) of a subset of jobs in steady state (cluster at full load), and their CDF.

5.2 End-to-End Physical Cluster Experiments

For the physical cluster experiments, we run a Synergy-TUNE (*tune*) and GPU-proportional allocation (*proportional*) for two different workload traces. (1) A static production-derived trace of 100 jobs with a *split* (60,30,10), scheduled using FIFO and evaluated for makespan. (2) A dynamic production-derived trace with continuous job arrivals and a split of (30,60,10), scheduled using SRTF and evaluated for average and 99th percentile JCT. Both scenarios use an appropriately sized trace that keeps the cluster fully loaded. We compare the obtained results to that of the simulator by replaying the same trace. Additionally, we compare our metrics to the upper bound generated by the optimal solution, Synergy-OPT (*opt*). The results are shown in Table 5.

Synergy-TUNE reduces the makespan of static trace by 1.4× when compared to GPU-proportional allocation. For the dynamic trace, Synergy-TUNE reduces average JCT of steady-state jobs by 1.5× while reducing the 99th percentile JCT of these jobs by 2× as shown in Table 5.
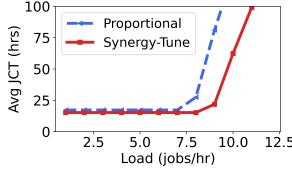
We compare the observed results from physical experiments to the same trace replayed on our simulator. As shown in Table 5, the difference between metrics in real and simulated clusters are less than 5%, demonstrating the fidelity of the simulator. We also see from Table 5 that the cluster objectives achieved by Synergy-TUNE are within 4% of the optimal solution in this case. We do not deploy the optimal allocations due to the challenges enumerated in §4.1.3
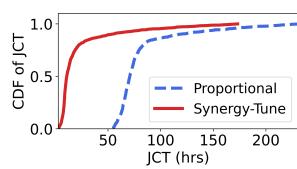
5.3 End-to-end results in simulation

5.3.1 Simulation with production traces

We run simulated experiments on a cluster of 512 GPUs across 64 servers using a subrange of the publicly available Philly trace published by Microsoft [5]. We assume a workload split of (20,70,10) for this trial. Table 6a lists the average JCT with Synergy and GPU-proportional scheduling for three differ-

(a) FIFO (single) (b) CDF (9 jobs/hr)

Figure 9: **Average JCT and CDF for FIFO**. Synergy improves the average JCT significantly compared to alllocation for varying cluster load. At a load of 9 jobs/hr, Synergy reduces average JCT from 81hrs to 22hrs, which is close to the upperbound of 20hrs predicted by Synergy-OPT.

ent scheduling policies. Across all policies, Synergy is able to reduce the average JCT compared to GPU-proportional scheduling due to better split of resources between jobs. The gains in Synergy can be attributed to reallocating the underutilized resources from a job to a different, resource-sensitive job whose throughput can improve with the increased allocation.
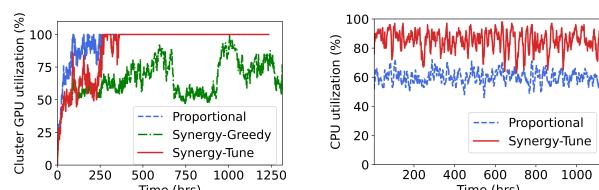
We show a detailed overview of the average and 99th percentile JCT for SRTF policy in Table 6b.We split the set of 1000 monitored jobs into short (JCT < 4 hrs) and long jobs. Synergy reduces the tail of the distribution by 2.2× for short jobs and the average JCT of both long and short jobs by 15%. For each of the 1000 monitored jobs, we plot the individual job speedup with respect to GPU-proportional scheduling in Figure 6c. We see that Synergy speeds up jobs by upto 9× using better resource allocations.

5.3.2 Simulation with varying load
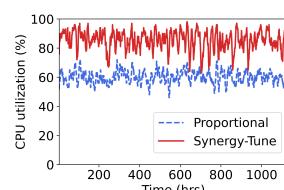
We run simulated experiments on a cluster of 128 GPUs across 16 servers using production-derived traces. We evaluate Synergy against GPU-proportional allocation mechanism for 4 different scheduling policies - FIFO, SRTF, LAS and FTF. We run dynamic workload traces, where jobs arrive continuously at a rate governed by a Poisson distribution. We show results for both single-GPU traces (where all jobs request 1 GPU) and multi-GPU traces (where jobs request upto 16 GPUs). Our metric of evaluation is the average JCT of a set of 1000 jobs in cluster steady state.

We show the results for three scenarios : LAS (multi-GPU trace) in Figure 7, SRTF (multi-GPU trace) in Figure 8, and FIFO (single GPU trace) in Figure 9. In all cases, we assume a workload split of (20,70,10). We plot both average JCT and the CDF of job completion times for a specific cluster load in both scenarios. For the multi-GPU trace, we split the CDF into those for short and long jobs to distinctly differentiate the tail of the distribution. We make three key observations.

First, Synergy-TUNE improves average JCT by up to 3.4× in the single-GPU trace, and up to 1.6× in the multi-GPU trace by speeding up resource sensitive jobs with disproportionate allocation. The improvement in average JCT is higher

(a) GPU utilization (b) CPU utilization

Figure 10: Cluster resource utilization

as the load increases, because at low load the cluster is not at full capacity. As load increases, jobs start to get queued and incur queuing delay before being scheduled on the cluster. Since Synergy significantly speeds up individual jobs using disproportionate resource allocation, pending jobs can get scheduled faster, thereby reducing their queuing delays. Therefore Synergy improves cluster metrics by both reducing qeuing delays and speeding up individual jobs. Note that, in GPU-proportional allocation, at higher loads, all CPUs and memory in the system are allocated to the running jobs but they can still be underutilized by individual jobs. We show later in Figure 10b, how Synergy's resource-sensitivity aware allocation improves CPU utilization in the system compared to GPU-proportional allocation. At low load, jobs are spread across the cluster and the unallocated CPU and memory is assigned to the jobs that benefit from additional auxiliary resources. Second, Synergy-TUNE is able to sustain a larger cluster load than GPU-proportional allocation. For multi-GPU scheduling with LAS, Synergy-TUNE reduced the 95th percentile JCT of long jobs by 2×. Third, the average JCT achieved with Synergy-TUNE is within 10% of the optimal solution in all cases.

Similarly, for FTF scheduling policy, Synergy-TUNE observed 2.3× and 2× improvement in average JCT for a single-GPU and multi-GPU trace respectively.

5.4 Impact of workload split

Workload split decides the percentage of resource sensitive jobs in the workload. As the percentage of speech and image models increase in the trace, there may not be enough spare CPU and memory resources to perform disproportionate allocation, as they are mostly CPU- and memory-hungry. Figure 11 plots the average JCT with varying load for 3 different workload splits with FIFO scheduling for multi-GPU jobs. As the percentage of resource-sensitive jobs increase, we observe that Synergy-GREEDY breaks down, and ends up degrading JCTs significantly compared to a GPU-proportional allocation. This is because, the naive greedy technique results in resource fragmentation when the demand along CPU and memory dimensions are high, leaving several GPUs underutilized. Whereas, by the design of Synergy-TUNE, it allocates at least as many resources required to achieve the throughput of GPU-proportional allocation; therefore, even in the worst case workload split shown in Figure 11c, where all the jobs

(a) Split=(20,70,10)   (b) Split=(33,33,33)   (c) Split=(50,0,50)
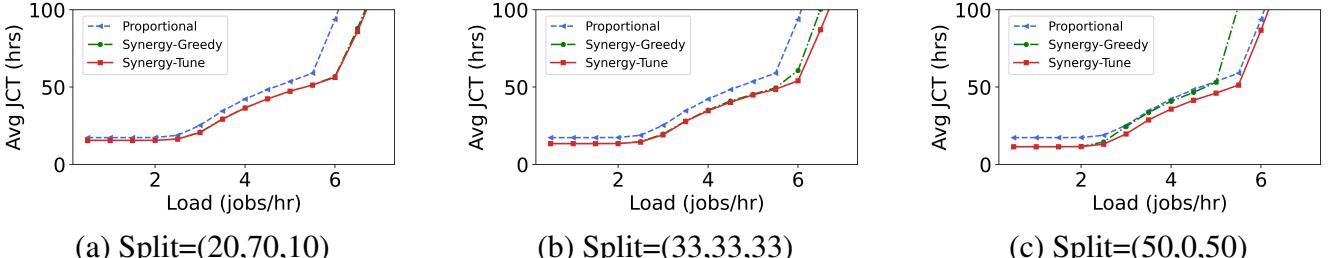
Figure 11: Evaluation of Synergy with varying workload split

(a) Ratio 4   (b) Ratio 5   (c) Ratio 6
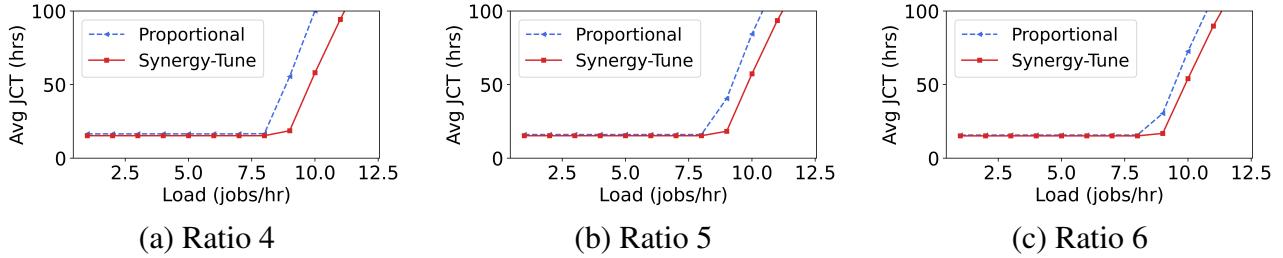
Figure 12: Evaluation of Synergy across different CPU:GPU Ratio

are CPU- and memory-sensitive, Synergy-TUNE performs as good as GPU-proportional allocation.

Resource utilization. Figure 10a plots the GPU allocation over time for the workload in Figure 11c at a load of 5.5 jobs/hr where the cluster GPU demand is higher than 100%. While Synergy-TUNE is able to sustain a higher load by finishing jobs faster, Synergy-GREEDY severely under-utilizes GPU resources throughout the workload, trading it off for higher CPU and memory allocation. At low loads, as shown in Figure 10b, GPU-proportional allocation only utilized 60% of the available CPU resources, while Synergy-TUNE utilized it up to a 90%, resulting in 1.5× lower average JCT.

5.5 Impact of CPU:GPU ratio

While our prior experiments assume a CPU:GPU ratio of 3 (similar to the NVIDIA DGX-2), Figure 12 plots the average JCT for a FIFO scheduler on a single-GPU trace as we increase cluster load and vary the CPU:GPU ratio from 4 to 6 (corresponding to other server SKUs in Table 2b). As the CPU:GPU ratio in a server increases, the baseline GPU-proportional scheduler gets more CPU cores per GPU, thereby reducing data stalls in the baseline. This in turn, reduces the gap between GPU-proportional and Synergy-TUNE. Despite that, at a load of 9 jobs/hr, Synergy-TUNE lowers the avg JCT by 3.4×, 3×, 2.2×, and 1.8× for a CPU:GPU ratio for 3, 4, 5 and 6 respectively.

5.6 Comparison to Synergy-OPT

Calculating optimal allocations for every scheduling round with Synergy-OPT can be quite expensive, especially for large


(a) DRF   (b) TETRIS
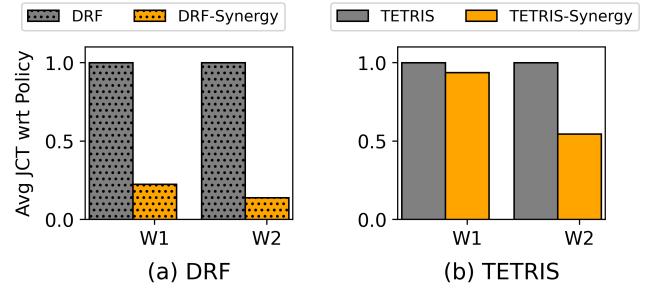
Figure 13: Comparison to big data scheduling policies

cluster sizes. We experimentally validated that the time taken for per-round allocations for Synergy-OPT increases exponentially with increasing cluster sizes, while that for Synergy-TUNE is hardly a second. We also show experimentally that the allocations given by Synergy-TUNE are close to those estimated by Synergy-OPT in §5.2 and §5.3.2. For a cluster size of 128 GPUs used in our experiments, Synergy-TUNE converges at allocations that are within 10% of the optimal value, 200× faster than Synergy-OPT.

5.7 Comparison to DRF and Tetris

Big data schedulers like Dominant Resource Fairness (DRF) [21] and Tetris [23] have explored multi-dimensional resource allocation for map-reduce jobs. DNN jobs have different properties when compared to big-data jobs. DNN jobs are gang-scheduled, meaning they can run only when all the GPUs requested by them are available on the cluster at once. Further, the auxiliary resource requirements like CPU and memory are fungible unlike the GPU demand. DRF and Tetris assume resources to be statically allocated throughout the life-

time of a job, whereas Synergy assumes these resources to be fungible and could result in varied allocations throughout the lifetime of a DNN job. Furthermore, profiling the DNN job's resource demands is unique to Synergy; big data schedulers assume that the job request already encodes resource demands across all dimensions. To evaluate Synergy against these policies, we assume that the best-case resource requirement for CPU and memory is fed as input to the bigdata scheduling policies using Synergy's profiling mechanism.

On a cluster of 128 GPUs, we evaluate these policies on two different workload compositions : W1 (20,70,10), and W2 (50,0,50) and compare the naive policy with its Synergy-variant, which allows resource tuning. W1 represents a workload split with a good mix of resource-sensitive as well as resource-insensitive jobs. W2 is a workload dominated by resource-sensitive jobs, which is one of the worst-case scenarios for multi-dimensional scheduling as it could lead to GPU fragmentation (explained in §5.4)

We plot the results in Figure 13. Tuning resource allocation across jobs using Synergy reduced the average JCT of DRF by 7.2$\times$ and that of Tetris by 1.8$\times$ for the workload split W2. This is because Synergy is able to allocate auxiliary resources in a fungible-manner every round, whereas the big-data scheduler's static allocations performs similar to greedy techniques, resulting in GPU fragmentation, and thereby degrading the overall cluster metrics. Synergy performs the best in each scenario as it uses the best-case resource demands of jobs to perform fungible, disproportionate allocation.

6 Discussion and Future Work

In this section, we elaborate on some of the assumptions made by Synergy, derived from our experiences with large scale deployed cluster schedulers at Microsoft, and discuss what happens if these assumptions are relaxed.

Homogeneous clusters. Scheduling in Synergy assumes that the GPU cluster is homogeneous. This assumption is based on the practical observation that our clusters have thousands of accelerators per homogeneous cluster [5]. While there is heterogeneity in hardware across clusters, it is often the case that users select one homogeneous cluster to run their job in production. For instance, a production cluster could have two homogeneous virtual clusters (VCs), each comprising of a specific generation of GPU. Each VC is managed separately, and assigned to a specific task - training or inference, for predictable performance. While recent works have explored the impact of blurring these boundaries and scheduling across heterogeneous hardware [11, 33, 42], such co-scheduling poses several practical challenges [52]. For example, some tasks such as low-latency inference are business-critical, user-facing applications which need to run on specific hardware, and need data isolation. Others have specific GPU memory requirements, or need advanced hardware features like NVLink.

Hence, users in our production settings specify a specific instance type to run each of their jobs on. Hence it is useful for a scheduler to optimize resource utilization in the context of homogeneous clusters. That said, Synergy 's ideas can also be extended to a heterogeneous cluster by profiling CPU and memory requirements along an additional dimension - GPU type, at an additional profiling cost. The optimal algorithm can then maximize throughput based on a 3-dimensional resource-sensitivity matrix W_j. We present the formulation for this in the extended version of the paper [38].

Use of MinIO. Synergy assumes the use of MinIO [39] because it is a DNN-aware caching mechanism that outperforms traditional OS page caching and allows performance predictability. It provides resource isolation and reduces storage fetch stalls [39]. If we do not use MinIO, we will have to profile the model at discrete memory allocations which will increase the profiling costs, and also potentially change the trends in profiling matrix.

Preprocessing overhead. Preprocessing for vision tasks includes random cropping and transformations of the image in the critical path. Reusing the same transformed images across epochs hurts accuracy [34, 37, 39], whereas it is practically infeasible to pre-process offline due to the prohibitive storage cost (dataset size * epochs). It is possible to alter the extent of CPU intensiveness by varying the number of augmentations performed. In this work, we have assumed that the augmentations required for each model are as specified by the published models themselves and we do not change this so as to not affect accuracy. On the horizon, we do observe recent schemes such as RandAugment [14], AutoAugment [13] which consider more computationally-intensive augmentation schemes (and associated accuracy gains). Such a rising trend in extreme preprocessing, makes a strong case for a system like Synergy.

Sharing storage and network. In our paper, we show how to reallocate CPUs and memory across jobs resident on the same server, for example, by co-locating a CPU-intensive task with a non CPU-intensive task. For our DNN training jobs, we assume that a dataset is downloaded locally and loaded into server memory when the job is started (constrained by the memory allocation limits). Prior work has similarly looked at co-locating network-intensive jobs with non network-intensive jobs [26, 35], but unlike Synergy, reallocation of shared network bandwidth is not explicitly handled by those schedulers. We leave it to future work to explore how ideas in Synergy can also be extended to reason about demands that individual jobs place on storage and network bandwidths.

GPU elasticity and sharing. While some recent works explore transparently changing the GPU allocation during the life of a job [48], the impact of changing batch sizes and hyperparameters on training accuracy is unclear for a wide variety of tasks. It is therefore practical to assume that the

GPU demand of a job is constant throughout its lifetime as is the case for jobs in our production clusters.

Synergy works by improving the throughput of jobs that are bottlenecked on data stalls. For jobs that have data stalls, GPU efficiency cannot be improved by multiplexing (spatial sharing) because they are waiting for input data. However, for a subset of jobs that are insensitive to auxiliary resource allocation, GPUs could be multiplexed between jobs. It would be interesting to explore how to impart resource-sensitivity awareness alongside GPU spatial sharing, which we leave for future work.

Tradeoff between consolidation and allocation. When multi-GPU jobs are split across physical servers, they may incur a penalty due to network communication [41, 55]. DNN jobs therefore prefer consolidation. In this work, we assume that no more than a server's worth of CPU or memory resources can be allocated to a job if its GPU demands can be satisfied by one server. However, we find that some jobs may benefit from giving up consolidation if the throughput gain due to increased CPU or memory allocation is higher than the penalty due to splitting. We leave the exploration of the trade off between consolidation and allocation, while taking into account the network overhead, to future work.

Leveraging model and pipeline parallelism. Our evaluation assumes distributed data-parallel jobs. But model and pipeline parallel execution schemes also have an input stage that ingest and pre-process data. Unlike data-parallel training, each stage in the pipeline might have a different CPU-GPU and memory-GPU requirement. While these jobs would have to be profiled to identify the CPU and memory sensitivity of each stage of the pipeline, Synergy 's contributions directly carry forward to such settings.

7   Related Work

DNN cluster schedulers. A number of recent schedulers for DNN workloads each focus on improving a certain objective; Cluster utilization (Gandiva [55]), JCT (Tiresias [26]), and fairness (Themis [35], Gandiva-Fair [11]). Some have also looked at exploiting performance heterogeneity among accelerators to improve cluster objectives [33, 42]. All these schedulers assume GPU to be the dominant resource in the scheduling task; i.e., a user requests a fixed number of GPUs for her DNN job, and when the requested number of GPUs are all available, the job is scheduled to run. Rather than allocating a fixed number of GPUs, building on GPU-elasticity for a single job [44], some recent schedulers like AFS [30] and Pollux [48] leverage throughput metrics to provide GPU elasticity in multi-tenant clusters (in addition to tuning batch size and learning rate). However, in all these cases, auxiliary resources such as CPU and memory are allocated proportional to the number of GPUs allocated to the job. Existing schedulers thus ignore *resource-sensitivity* of the DNN tasks

to CPU and memory. Synergy shows that, irrespective of the number of GPUs allocated, auxiliary resource-sensitive allocation is crucial to achieve better cluster utilization.

Big data schedulers. Our work builds upon the insights drawn from the rich literature of schedulers for big data jobs [21, 23–25, 29, 51]. Big data schedulers like Tetris [23], and DRF [21] have looked at the problem of multi dimensional resource allocation for big data jobs. They propose new scheduling policies aimed at optimizing a specific cluster objective for jobs whose resource demands are prior known. In contrast, the primary resource in a DNN job is the accelerator (GPU), whose requirement is specified by the job; other resources are fungible. Our work exploits this insight to perform disproportionate allocations by profiling job resource sensitivity, and then appropriately packing them onto servers.

Data stalls. Recent, deep characterization studies explored the impact of CPU and memory on individual DNN jobs [39, 40] Unlike prior work that focuses on individual jobs, the focus of our paper is on the tricks we can play when we schedule multiple jobs together in a cluster.

Disaggregated data prep. There have been recent orthogonal efforts that aim at reducing the cost of data preprocessing, and thereby the load on CPUs using disaggregated data prep [59]. However, one has to pay the network cost of shuffling preprocessed tensors, which could quickly become the bottleneck especially for vision models with rich datasets. Synergy on the other hand, assumes standard pre-processing pipelines at the training servers, and aims to reduce the cost of pre-processing using better resource allocation.

8   Conclusion

This paper introduces Synergy, a resource-sensitive scheduler for DNN training jobs. Synergy is based on the insight that not all jobs exhibit the same level of sensitivity to CPU and memory allocation during DNN training; breaking the shackles of GPU-proportional allocation can result in improved utilization of existing cluster resources and improved job and cluster-wide objectives. Our experiments on physical and large simulated clusters show that Synergy can reduce average JCT by upto 3.4× over GPU-proportional allocation.

Acknowledgements

References

[1] Amazon EC2 P3 - Ideal for Machine Learning and HPC - AWS. https://aws.amazon.com/ec2/instance-types/#p3.

[2] Azure NC_v3 Series. https://docs.microsoft.com/en-us/azure/virtual-machines/ncv3-series.

[3] Azure ND_v2 Series. https://docs.microsoft.com/en-us/azure/virtual-machines/ndv2-series.

[4] gRPC. https://grpc.io/.

[5] Microsoft philly traces. https://github.com/msr-fiddle/philly-traces.

[6] NVIDIA DGX-2: Enterprise AI Research System. https://www.nvidia.com/en-us/data-center/dgx-2/.

[7] NVIDIA DALI. https://github.com/NVIDIA/DALI, 2018.

[8] Pytorch. https://github.com/pytorch/pytorch, 2019.

[9] Wmt16. http://www.statmt.org/wmt16/, 2020.

[10] Brendan Burns, Brian Grant, David Oppenheimer, Eric A. Brewer, and John Wilkes. Borg, omega, and kubernetes. *Commun. ACM*, 59(5):50–57, 2016.

[11] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi 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.

[12] Mark Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *2nd USENIX Symposium on Internet Technologies and Systems, USITS'99, Boulder, Colorado, USA, October 11-14, 1999*. USENIX, 1999.

[13] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 113–123. Computer Vision Foundation / IEEE, 2019.

[14] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[15] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE, 2017.

[16] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2978–2988. Association for Computational Linguistics, 2019.

[17] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis. *arXiv preprint arXiv:1612.01840*, 2016.

[18] NVIDIA DGX-1. https://www.nvidia.com/en-us/data-center/dgx-1/.

[19] Steven Diamond and Stephen P. Boyd. CVXPY: A python-embedded modeling language for convex optimization. *J. Mach. Learn. Res.*, 17:83:1–83:5, 2016.

[20] György Dósa and Jirí Sgall. First fit bin packing: A tight analysis. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 538–549. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[21] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011.

[22] Google. Open images dataset. https://opensource.google/projects/open-images-dataset.

[23] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 455–466. ACM, 2014.

[24] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 65–80. USENIX Association, 2016.

[25] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 81–97, Savannah, GA, November 2016. USENIX Association.

[26] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, pages 485–500. USENIX Association, 2019.

[27] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011.

[30] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 721–739. USENIX Association, 2021.

[31] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[33] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. Allox: compute allocation in hybrid clusters. In

Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 31:1–31:16. ACM, 2020.

[34] Gyewon Lee, Irene Lee, Hyeonmin Ha, Kyung-Geun Lee, Hwarim Hyun, Ahnjae Shin, and Byung-Gon Chun. Refurbish your training data: Reusing partially augmented samples for faster deep neural network training. In Irina Calciu and Geoff Kuenning, editors, *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 537–550. USENIX Association, 2021.

[35] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi 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.

[36] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[37] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. Checkfreq: Frequent, fine-grained DNN checkpointing. In *19th USENIX Conference on File and Storage Technologies, FAST 2021, February 23-25, 2021*, pages 203–216. USENIX Association, 2021.

[38] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Synergy: Resource sensitive DNN scheduling in multi-tenant clusters. *CoRR*, abs/2110.06073, 2021.

[39] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in DNN training. *Proc. VLDB Endow.*, 14(5):771–784, 2021.

[40] Derek Gordon Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk. tf.data: A machine learning data processing framework. *Proc. VLDB Endow.*, 14(12):2945–2958, 2021.

[41] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15. ACM, 2019.

[42] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 481–498. USENIX Association, 2020.

[43] Misja Nuyens and Adam Wierman. The foreground-background queue: A survey. *Perform. Evaluation*, 65(3-4):286–307, 2008.

[44] Andrew Or, Haoyu Zhang, and Michael J. Freedman. Resource elasticity in distributed deep learning. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020.

[45] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 5206–5210. IEEE, 2015.

[46] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 3:1–3:14. ACM, 2018.

[47] PyTorch. Word-level language modeling rnn. https://github.com/pytorch/examples/tree/master/word_language_model.

[48] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 2021.

[49] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[51] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop YARN: yet another resource negotiator. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16. ACM, 2013.

[52] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu 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 22)*, pages 945–960, 2022.

[53] Gerhard J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Inf. Process. Lett.*, 64(6):293–297, 1997.

[54] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[55] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 595–610. USENIX Association, 2018.

[56] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 533–548. USENIX Association, 2020.

[57] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10,*

Boston, MA, USA, June 22, 2010. USENIX Association, 2010.

[58] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[59] Mark Zhao, Niket Agarwal, Aarti Basant, Bugra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. Understanding and co-designing the data ingestion pipeline for industry-scale recsys training. *CoRR*, abs/2108.09373, 2021.