MLTCP: A Distributed Technique to Approximate Centralized Flow Scheduling For Machine Learning

Sudarsanan Rajasekaran, Sanjoli Narang*, Anton A. Zabreyko*, Manya Ghobadi
Massachusetts Institute of Technology

ABSTRACT

This paper argues that congestion control protocols in machine learning datacenters sit at a sweet spot between centralized and distributed flow scheduling solutions. We present MLTCP, a technique to augment today's congestion control algorithms to approximate an interleaved centralized flow schedule. At the heart of MLTCP lies a straight-forward principle based on a key conceptual insight: by scaling the congestion window size (or sending rate) based on the number of bytes sent at each iteration, MLTCP flows eventually converge into a schedule that reduces network contention. We demonstrate that MLTCP uses a gradient descent trend with a step taken at every training (or fine-tuning) iteration towards reducing network congestion among competing jobs.

CCS CONCEPTS

• **Networks → Data center networks**; **Network resources allocation**; *Transport protocols*; *Network management*; • **Computing methodologies → Neural networks**.

KEYWORDS

Congestion control, Networks for ML, Resource allocation, Datacenters for ML, Transport layer, DNN training

ACM Reference Format:
Sudarsanan Rajasekaran, Sanjoli Narang*, Anton A. Zabreyko*, Manya Ghobadi, Massachusetts Institute of Technology . 2024. MLTCP: A Distributed Technique to Approximate Centralized Flow Scheduling For Machine Learning. In *The 23rd ACM Workshop on Hot Topics in Networks (HOTNETS '24), November 18–19, 2024, Irvine, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3696348.3696878

1 INTRODUCTION

Efficient flow scheduling is an important and well-studied problem in the networking community [3, 5, 7, 12, 13, 23, 24, 27]. There is a vast body of work on scheduling flows using heuristics, load-balancing mechanisms, and deadlines for network flows.

Traditionally, there have been two broad approaches to implementing flow scheduling. First is the centralized approach, where a central controller collects the network demands from all the flows and computes the desired flow schedule [3, 12, 13, 27, 49]. The second is to approximate heuristics, such as Shortest Remaining Processing Time first (SRPT), in a distributed manner with the help of packet priorities or switch support [5, 7, 23, 44].

Most flow scheduling approaches focus on conventional datacenter traffic, which is bursty and short [9]. Moreover, legacy datacenter flows' arrivals are often independent and unpredictable. Today, with increasing demand for AI-based services, Deep Neural Network (DNN) training and fine-tuning traffic in datacenters has exponentially increased. Unlike traditional datacenter workloads, DNN training and fine-tuning jobs have a periodic traffic pattern where the start time of each training iteration depends on the completion of the preceding iteration, creating a dependency on the flow arrival times [53, 59, 64].

We demonstrate that scheduling techniques that favor jobs based on the shortest remaining processing times (i.e., pFabric [5], PDQ [23], and PIAS [7]) are not always optimal for scheduling DNN jobs. Intuitively, this is because such techniques make local scheduling decisions based on the status of current flows in the network without considering the flow arrival patterns of periodic jobs. This effect becomes adverse in DNN workloads where finishing the flows in one iteration impacts the completion time of subsequent iterations.

Recent studies, such as Muri [64] and Cassini [52, 53], have demonstrated that for DNN workloads, schedules that promote interleaving of communication demands achieve optimal network schedule. They define the idea of interleaving as overlapping the communication phase (high network demand) of one DNN job with the compute phase (low network

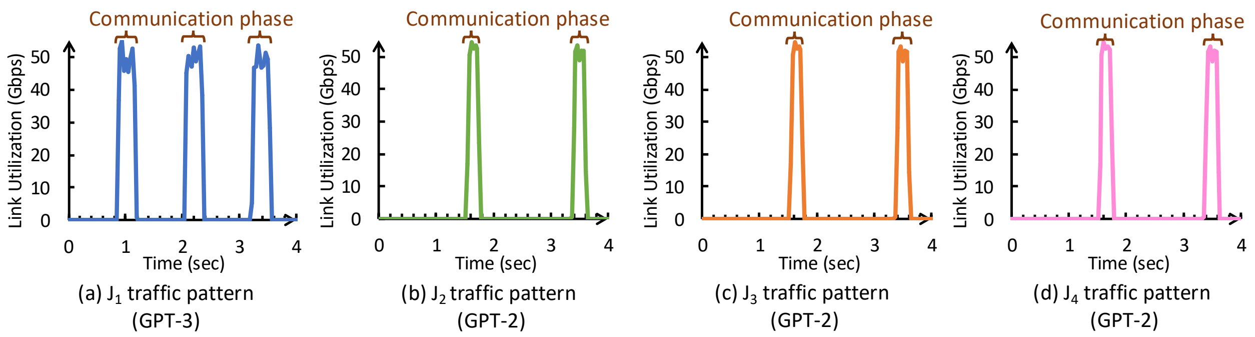

Figure 1: The traffic pattern of jobs J_1, J_2, J_3, and J_4.

demand) of other jobs sharing the link. However, both approaches require a centralized controller to find the optimal schedule.

Although the periodic nature of DNN jobs poses a challenge to traditional schedulers to achieve the optimal, we argue that the same traffic pattern also creates an unprecedented opportunity to approximate a centralized optimal schedule using a distributed approach. In this paper, we introduce MLTCP, a novel approach to leverage congestion control algorithms to approximate interleaved flow schedules for DNN flows in a distributed manner. Importantly, MLTCP does not need any hardware changes or priority queues. Hence, unlike centralized scheduling algorithms, MLTCP is easily deployable and scalable.

MLTCP adjusts the congestion window size based on a linearly increasing function $\mathcal{F}(bytes\_ratio)$, where $bytes\_ratio$ is the ratio of bytes successfully sent during the current training (or fine-tuning) iteration normalized by the total number of bytes sent every iteration (§3). Consequently, $\mathcal{F}(bytes\_ratio)$ enables MLTCP to create unequal bandwidth sharing between competing jobs, which forces the jobs to iteratively converge towards an approximately interleaved state. Our theoretical analysis of MLTCP shows that this iteration-by-iteration convergence is equivalent to performing gradient descent on a loss function that promotes interleaving (§4).

2 MOTIVATION

Distributed DNN Training and Fine-tuning Distributed DNN training and fine-tuning jobs are flooding today's datacenters. These jobs have highly regular, periodic communication demands [46, 52, 53, 59, 64]. Unlike classical work on periodic traffic [36, 54, 55], the arrival of the next flow from a DNN job depends on the completion of the previous. Recent work has shown that interleaving, or overlapping the communication phase of one job with the computation of the others, is optimal for this type of traffic [52, 64].

Example. We illustrate the impact of different scheduling techniques on DNN jobs with a testbed experiment. Consider a cluster with eight A100 GPU servers connected in a dumbbell topology with a single bottleneck link of capacity 50 Gbps. We train four DNN jobs, J_1, J_2, J_3, and J_4 on this cluster. Each job uses 2 GPUs installed on the opposite sides of the bottleneck link. J_1 trains a GPT-3 [11] model using two GPU servers and has a communication traffic pattern represented by Figure 1(a). Jobs J_2, J_3, and J_4 are identical GPT-2 [51] training instances, each using two GPU servers and having a traffic pattern shown in Figures 1(b), 1(c), and 1(d). For simplicity, consider the scenario when all four jobs start the communication phase of their first iteration at the same time.

Centralized approaches. In these approaches, a central entity that is aware of the communication demands of all the jobs, computes the optimal schedule using an Integer Linear Problem (ILP) solver. For instance, Cassini [52] uses a network-aware centralized scheduler to achieve interleaving. Figure 2(a) represents Cassini's optimal interleaved schedule for the four jobs. The average iteration time of job J_1 is 1.2 seconds, and that of jobs J_2, J_3, and J_4 are 1.8 seconds. Centralized approaches achieve optimal scheduling at the cost of being computationally expensive, making it challenging to scale to a large cluster. They also rely on accurate profiling of the network demands to compute the optimal schedule.

Distributed approaches. To the best of our knowledge, there is no prior work that achieves distributed flow scheduling for DNN jobs. Hence, we analyze conventional distributed flow schedulers like pFabric [5], PDQ [23], and PIAS [7] that employ heuristics to approximate SRPT schedule [54, 55] using switch hardware to minimize average flow completion times. However, these approaches are not always optimal for periodic traffic, even for a single link.[1] Figure 2(b) represents flow scheduling according to pFabric. pFabric prioritizes jobs J_2, J_3, J_4 which have a smaller communication

[1]The problem of scheduling periodic flows with variable bandwidth demands and having dependencies between flow arrivals and flow completion is NP-hard. The proof involves reduction from 1-D bin packing problem.

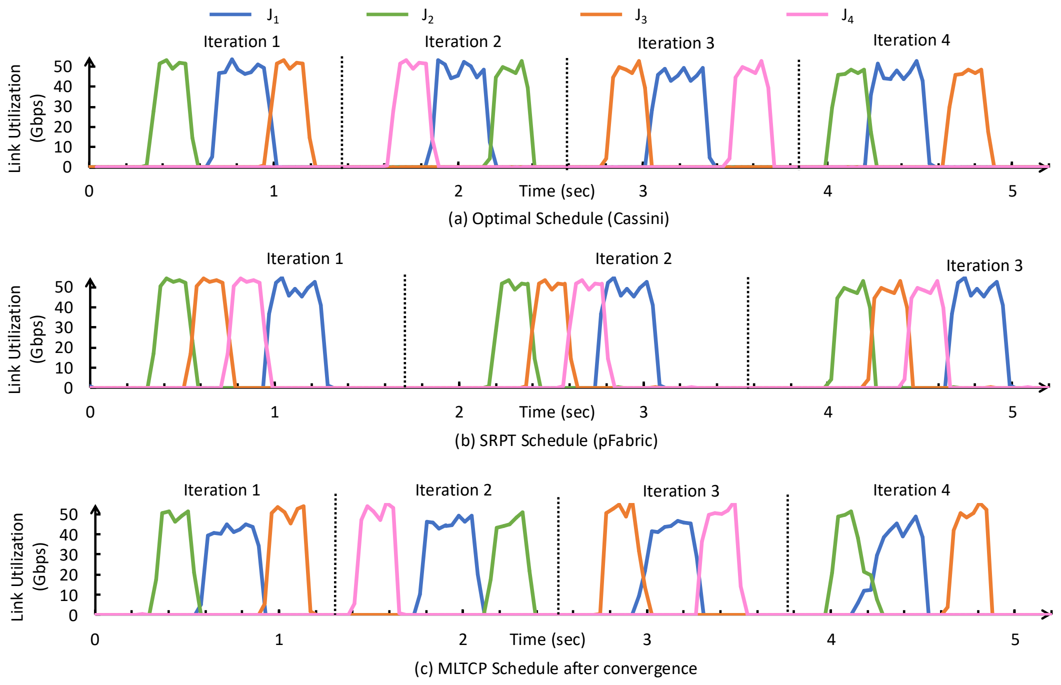

Figure 2: Comparison of different scheduling approaches on the iteration times of four DNN training jobs. MLTCP **achieves the same schedule as the optimal (Cassini), while SRPT (pFabric) does not.**

demand and delay the communication of J_1 every iteration. With pFabric, the average iteration times of all four jobs is 1.8 seconds. The iteration times of jobs J_2, J_3 and J_4 remain close to ideal, but job J_1 incurs a slowdown of 1.5× in its iteration time. The core reason for this is that SRPT blindly prioritizes the shortest flows without considering the inter-dependence between flow arrivals in case of DNN jobs. In this case, SRPT causes head-of-line blocking for J_1 by prioritizing the jobs with smaller flow sizes.

Surprising impact of congestion control on resource interleaving. In this paper, we demonstrate a surprising feature of congestion control protocols that enables service providers to have the best of both centralized and distributed worlds. To do so, we propose MLTCP, a straightforward technique to augment a family of congestion control algorithms. MLTCP-enabled congestion control protocols automatically achieve near-optimal interleaving in a distributed manner without the need for a centralized entity or priority queues or switch hardware support. For our scenario of four jobs, Figure 2(c) shows MLTCP's final interleaved state. Every iteration, MLTCP gradually shifts the communication patterns, akin to a gradient descent approach (§4), to ultimately converge to the optimal interleaved schedule when it stops shifting them further.

Approximation error. In the above experiment, MLTCP converges to an interleaved state within 20 iterations. In particular, the average iteration times of the four jobs converge to within 5% of the optimal centralized schedule, and the interleaving remains stable in subsequent iterations. We provide an upper bound on the approximation error of MLTCP in Section 4.

3 FUNDAMENTALS OF MLTCP

This section describes how MLTCP modifies distributed congestion control protocol TCP-Reno to approximate interleaving of communication demands of DNN jobs.

3.1 Augmenting TCP Reno with MLTCP

MLTCP's goal is to stabilize flows belonging to different DNN training and fine-tuning jobs into an interleaved state without using a centralized controller, regardless of job start times, DNN model size, parallelization strategy, or number of flows competing for bandwidth. To serve this purpose, MLTCP exploits a key conceptual insight: DNN flows should *dynamically adjust* their aggressiveness for link bandwidth based on the number of bytes sent in that iteration. This allows the flow closest to completing its iteration to receive a larger share of available bandwidth than the others, thereby finishing faster.

MLTCP's sliding effect. MLTCP's unequal bandwidth allocation creates a "shift" in the start times of the communication phases of different jobs. Such a shift in time causes a sliding effect, where the communication phase of one job slides

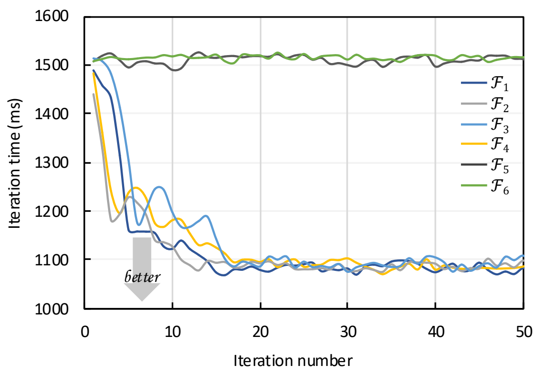

Figure 3: Performance of different $\mathcal{F}(bytes\_ratio)$ functions.

into the compute phase of other jobs (details in section §4) after a few iterations. Following this insight, MLTCP adjusts the congestion window (cwnd) or sending rate of flows based on a bandwidth aggressiveness function, $\mathcal{F}(bytes\_ratio)$, of the number of bytes sent normalized by the total bytes to be sent in one iteration of each flow ($bytes\_ratio = \frac{bytes\_sent}{total\_bytes}$). Following Linux's implementation, throughout this paper, we assume that the congestion window (cwnd) is expressed in packets (not bytes).

MLTCP-Reno. The TCP Reno algorithm uses a cumulative ack mechanism to acknowledge multiple in-flight packets with a single ack. In the additive increase step, the Reno algorithm increases the cwnd by $\frac{\#num\_acks}{cwnd}$ upon receiving a packet having $\#num\_acks$ acks. MLTCP scales this increment by the bandwidth aggressive function, $\mathcal{F}(bytes\_ratio)$, as follows:

$$cwnd \leftarrow cwnd + \mathcal{F}(bytes\_ratio) \times \frac{\#num\_acks}{cwnd} \quad (1)$$

Bandwidth aggressiveness function. MLTCP uses the aggressiveness function $\mathcal{F}$ as a scaling factor for the cwnd increment made during the window (or rate) increase step. Many aggressiveness functions achieve MLTCP's interleaving goals as long as they satisfy the following three requirements: (*i*) the range is large enough to absorb the noise (e.g., slight variations in round-trip time (RTT) or iteration times) in the network; (*ii*) the derivative of the function is non-negative; (*iii*) all flows employ the same bandwidth aggressiveness function.

Figure 3 compares the performance of six different bandwidth aggressiveness functions $\mathcal{F}$. In this experiment, three GPT-2 [51] training jobs compete for bandwidth using MLTCP-Reno. We run six experiments each with a different function, as follows:

- $\mathcal{F}_1 = 1.75(bytes\_ratio) + 0.25$
- $\mathcal{F}_2 = 1.75(bytes\_ratio)^2 + 0.25$
- $\mathcal{F}_3 = 1/(-3.5(bytes\_ratio) + 4)$
- $\mathcal{F}_4 = -1.75(bytes\_ratio)^2 + 3.5(bytes\_ratio) + 0.25$
- $\mathcal{F}_5 = -1.75(bytes\_ratio) + 2$
- $\mathcal{F}_6 = -1.75(bytes\_ratio)^2 + 2$

Algorithm 1 MLTCP-Reno Algorithm

```
1:  procedure INITIALIZE:($TOTAL\_BYTES$, $COMP\_TIME$)
      ▷ Input Parameter $TOTAL\_BYTES$: Total bytes per iteration
      ▷ Input Parameter $COMP\_TIME$: Gap in communication for detecting iteration boundary
2:      $bytes\_ratio = 0$ ▷ Current fraction of bytes sent in this iteration
3:      $bytes\_sent = 0$ ▷ Number of successfully sent bytes
4:      $prev\_ack\_tstamp = 0$ ▷ Timestamp of the previous ack
5:      $MTU = 1500$ ▷ Maximum packet size used by the system
6:  procedure CONGESTION_AVOIDANCE($num\_acks$)
      ▷ Input Parameter $num\_acks$
7:      $bytes\_sent$ += $num\_acks \times MTU$
8:      $curr\_ack\_tstamp$ = GET_REAL_TIME()
9:      $curr\_gap = curr\_ack\_tstamp - prev\_ack\_tstamp$
10:     if $curr\_gap > COMP\_TIME$ then
11:         ▷ Start of new training iteration
12:         ▷ State reset
13:         $bytes\_ratio = 0$; $bytes\_sent = 0$;
14:     else
15:         ▷ Middle of training iteration
16:         $bytes\_ratio = min\left(1, \frac{bytes\_sent}{TOTAL\_BYTES}\right)$
17:     $prev\_ack\_tstamp = curr\_ack\_tstamp$
18:     $cwnd = cwnd + F(bytes\_ratio) * (\frac{num\_acks}{cwnd})$
19:     return
```

All these functions have the same range (0.25 − 2) but $\mathcal{F}_1, ..., \mathcal{F}_4$ are increasing and $\mathcal{F}_5$ and $\mathcal{F}_6$ are decreasing. Figure 3 shows the average training iteration time of different iterations as the jobs start their training process. As shown, the iteration time of MLTCP-Reno with $\mathcal{F}_1, ..., \mathcal{F}_4$ starts to decrease, as the communication demands interleave after ≈20 iterations. On the other hand, the iteration times of MLTCP-Reno with $\mathcal{F}_5$ and $\mathcal{F}_6$ do not improve. Even though different increasing functions take slightly different numbers of iterations to interleave the jobs, they eventually achieve the interleaved state.

We select the bandwidth aggressiveness function used in MLTCP to be linear in $bytes\_ratio$, to simplify MLTCP's implementation in the linux kernel and to minimize computational overhead. We define $\mathcal{F}(bytes\_ratio)$ as:

$$\mathcal{F}(bytes\_ratio) = Slope \times bytes\_ratio + Intercept \quad (2)$$

where $Slope$ and $Intercept$ represent the linear function's slope and intercept, respectively. These are constant parameters tuned based on the link rate and the noise in the system. In this paper we use $Slope = 1.75$ and $Intercept = 0.25$.

3.2 MLTCP-Reno Congestion Avoidance

MLTCP-Reno algorithm. We implement MLTCP-Reno in the Linux kernel using the pluggable congestion module [2, 22] to insert the MLTCP-Reno procedure, shown in Algorithm 1, as a hook into the TCP stack. This function has two essential goals: the first is to update the number of successfully sent bytes, and the second is to adjust the congestion window. MLTCP-Reno is called by the TCP stack

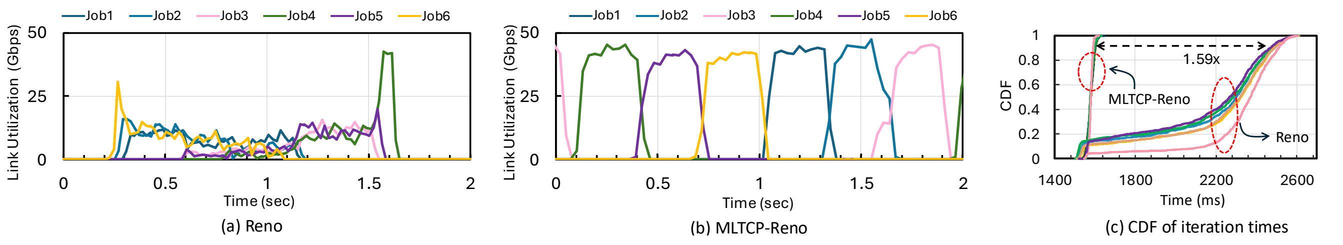

Figure 4: Bandwidth allocation when six jobs share the bottleneck link.

whenever an ack packet is received. This information is readily available from the socket data structure (line 7). If the time since the last received ack is greater than the computation time (line 10), then it resets MLTCP's parameters (line 13). Otherwise, it computes *bytes_ratio* based on the current *bytes_sent* (line 16). Finally, we record the current time to compare against the next time an ack is received and update the congestion window (line 18).

Obtaining *TOTAL_BYTES* and *COMP_TIME*. The total bytes in each iteration (*TOTAL_BYTES*) and computation time (*COMP_TIME*) are constant for each job and depend on the size of the DNN model, the parallelization strategy, and the communication collective. In our implementation, we automatically learn these values by measuring the total amount of data and computation time during the first few iterations. We measure the computation time by detecting gaps in the ack arrivals that exceed several round-trip times (RTTs).

Figure 4(a) shows the bandwidth allocation of six identical GPT-2 [51] jobs competing on a network link using TCP Reno. The figure shows heavy network congestion, where all jobs take longer to finish. In contrast, Figure 4(b) shows the same setup with MLTCP-Reno, where all jobs achieve a near-optimal interleaved state. Figure 4(c) highlights the tail iteration time speedup of 1.59× achieved using MLTCP compared to standard TCP-Reno over the lifetime of the jobs.

4 ANALYSIS OF MLTCP

In this section, we provide a theoretical analysis of MLTCP and its iteration-by-iteration progress toward an interleaved state for DNN jobs.

Compatibility and network demand assumptions. We limit the scope of our analysis to scenarios in which an interleaved schedule exists [52], and the network demand phase of each job is continuous and constant within an iteration. Under these two assumptions, MLTCP is guaranteed to converge to the optimal resource interleaving with an error linearly bounded by the noise in the system.

We show that the process of convergence is essentially a gradient descent over a loss function that hits minimum when the communication demands of different jobs are interleaved. Unlike centralized approaches that solve for an optimum in one shot, MLTCP explores the solution space to find a minima at a rate governed by the bandwidth aggressiveness function.

The key idea behind MLTCP is to adjust the cwnd based on the bytes sent in an iteration (§3). As a result, MLTCP divides the link capacity between the flows unequally when they compete for the network. This difference in the bandwidth leads to unequal progress of the current iteration of the jobs, causing shifts in the start times of the communication phase of their subsequent iterations. In this section, we formally define the "shift" created by MLTCP's unequal bandwidth sharing and use it to construct the loss function for gradient descent. To understand the principle, let us consider a simple example of two identical DNN training jobs. The same analysis applies to any combination of jobs that satisfy the compatibility and network demand assumptions.

Figure 5 illustrates two identical DNN training jobs sharing a network link with capacity C and *ideal iteration time* T. This ideal iteration time is achieved when each job is executed in isolation, as shown in Figure 5(a). Each iteration of training has a duration of T seconds. The communication phase lasts $a \times T$ seconds, where $a < 1$ is a constant depending upon the DNN job. The difference in the start times of the i^{th} iterations of the two jobs is given by Δ_i. Figure 5(b) shows the i^{th} and $(i+1)^{th}$ iterations of the two jobs using MLTCP. In this scenario, MLTCP allocates more than half of the bandwidth to the first job, allowing it to complete its current iteration early, and delays the second job by assigning it a lower bandwidth. This phenomenon causes the difference in the start times of the next iteration of the two jobs to increase to Δ_{i+1} where $\Delta_{i+1} = \Delta_i + Shift(\Delta_i)$. We refer to the increase in the start times of the next iteration relative to the previous one as the $Shift$ introduced by MLTCP. This $Shift$ caused in the communication pattern induces a sliding effect, which, over multiple iterations, aids in separating the communication demands of the jobs. Figure 6 illustrates the shift and sliding behavior of MLTCP-Reno when two GPT-2 [51] training jobs share a bottleneck link in our testbed.

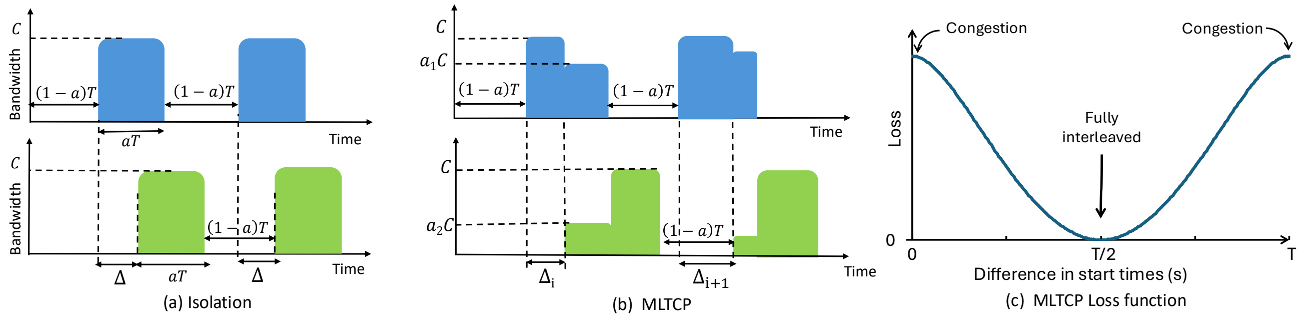

Figure 5: MLTCP shifts the communication pattern

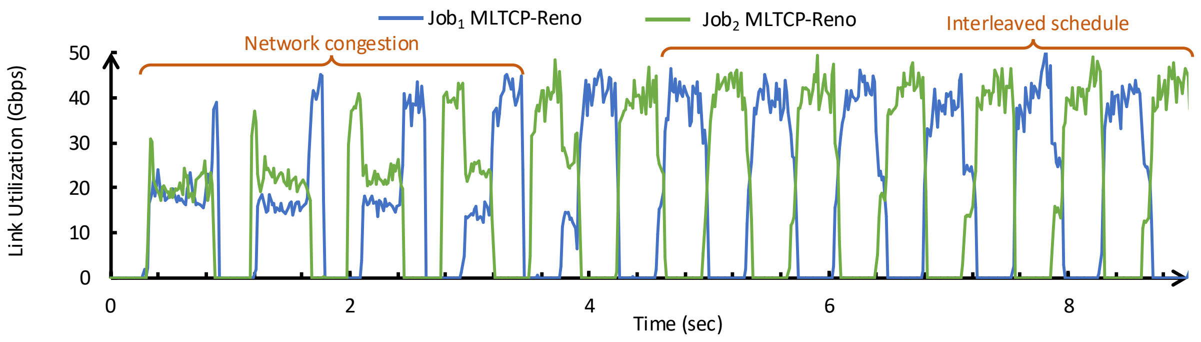

Figure 6: MLTCP interleaves the communication demands of two GPT-2 jobs over few iterations.

Given the bandwidth aggressiveness function in Equation 2, we mathematically compute the shift, $Shift$, as a function of the start time difference Δ_i, between the jobs in the i^{th} iteration as:

$$Shift(\Delta_i) = \frac{Slope \times \Delta_i \times (a \times T - \Delta_i)}{a \times T \times Intercept \ + \Delta_i \times Slope} \quad (3)$$

MLTCP generates a shift in the communication pattern which accumulates over multiple iterations to gradually slide them to a configuration that is close to optimal interleaving. To understand the final state it converges to, we define a loss function as the negative integral of the shift function, as given by:

$$Loss(\Delta_i) = \int_{\Delta=0}^{\Delta_i} -Shift(\Delta)\, d\Delta \quad (4)$$

Given the compatibility and network demand assumptions, the loss function obtained by MLTCP is guaranteed to have only global optima. Hence, our approach converges to a global optimum, i.e., a fully interleaved state. Different aggressiveness functions give rise to different shift functions and, eventually, to different loss functions.

To visualize the loss function in the example of two jobs, we take $a = 1/2$ (i.e. 50% communication phase) in Figure 5(c) for simplicity. For this choice of a, the loss function is minimum at value $\Delta_i = T/2$, when network contention is minimum, and the communication demands are interleaved. Even for the case of many different jobs, the loss function has the same characteristics as that of this simple example. Note that changing the configuration based on the shift function adds a negative derivative of the loss function to the configuration. In other words, MLTCP performs a gradient descent on the loss function, shifting the communication patterns after every iteration. Given any starting configuration, MLTCP using gradient descent gradually converges to the minimum of the loss function.

As MLTCP is essentially a gradient descent, it converges to a stable optimum. However, small perturbations arising due to slight variations in compute durations, network latencies, and clock synchronization in a practical environment disturb the system state. We model all these perturbations as zero mean Gaussian noise in the iteration time of each job. Under this model, we quantify the approximation error of MLTCP based on how far is the steady state from the optimal interleaved schedule. Assuming that the noise distribution has a standard deviation σ, MLTCP's convergence error also follows a normal distribution with mean zero and standard deviation $2\sigma \times \left(1 + \frac{Intercept}{Slope}\right)$. Hence, the approximation error is linearly bounded by the intensity of noise in the iteration times of the jobs.

5 DISCUSSION

Generalization to multi-resource scheduling. MLTCP's scope is not limited to network scheduling. The aggressiveness function $\mathcal{F}(bytes\_ratio)$ is generalizable to other resource scheduling problems by replacing *bytes_ratio* with the progress of the job. For example, in the case of CPU cores, the operating system's scheduler tracks the progress of each task, and assigns a number of CPU cores based on the desired aggressiveness function. The dimension of gradient descent space increases with the number of jobs. For allocating multiple resources among multiple jobs periodically, the loss becomes a function of the overlap across all resources. The relative shifts for each job, calculated from the gradient of this function, thus takes into account each resource type.

Fairness between MLTCP and TCP flows. TCP's throughput is inversely proportional to the square root of loss of probability [41]. Our analysis shows that the throughput of our MLTCP-Reno flows is inversely proportional to the loss probability. Intuitively, this implies that given the same packet loss probability, an MLTCP-Reno flow claims more bandwidth share than a standard Reno flow. However, MLTCP-Reno flows would not starve the other legacy flows because MLTCP allocates non-zero bandwidth to all the competing flows. To safeguard high-priority legacy TCP traffic, we modify NCCL's FAST socket plugin [1] to support selecting a desired congestion control algorithm. This allows for choosing different aggressiveness functions for different classes of traffic. For latency-sensitive traffic, in order to acquire most of the bandwidth, we recommend using a bandwidth aggressiveness function with larger values.

6 RELATED WORK

Congestion control. There is a vast literature on congestion control. Many rely on feedback signals indicating congestion in the network and reduce their sending rate [4, 8, 10, 16, 19, 26, 32, 34, 37, 42, 62, 66]. Others are deadline-aware [58, 60], router-assisted [5, 15, 31], and receiver-based [20]. We chose to augment TCP-Reno because it is a classic congestion control algorithm. Other congestion control schemes are augmented in a similar way to induce shifts in communication start times.

Flow scheduling. There are two broad directions for implementing flow scheduling: centralized flow scheduling and heuristic-based distributed flow scheduling. In the centralized approaches, the network demands of different flows are sent to a central entity, which computes the optimal schedule for all the flows [3, 12, 13, 27, 49]. The distributed approaches often implement heuristics like SRPT [54, 55], Shortest Remaining Job first (SJF), Least Attained Service first (LAS), Earliest Deadline First (EDF), with the help of the switch and priority queues [5, 7, 23, 44].

Periodic resource scheduling. Beyond networks, there have been efforts to schedule periodic tasks across limited resources, including real-time and embedded systems [17, 33]. However, these have explicit deadlines, and inter-arrival times have no dependency on completion times. Cyclic scheduling is a well-studied problem in mathematics [14, 56]. The most general form of this problem, from Serafini et al. [56], expresses our scenario but is shown to be NP-hard.

Accelerating DNN Training. Prior work demonstrated that generic flow schedulers are not optimal for DNN training jobs [40, 46]. DNN-specific flow schedulers have been developed to meet this demand [21, 29, 39, 48, 57]. Alternatively, intra-job pipelining overlaps the compute and communication phases of the *same training job* [25, 30, 35, 35, 43, 45, 48]. These approaches only optimize a single job's performance, while we share resources across multiple jobs. Job placement schedulers try to minimize multi-job contention. Many focused on compute-optimization in how they assign workers to jobs, and only considered the network so far as to try and schedule workers for a job close together [18, 28, 38, 47, 50, 61, 65]. Our work complements these schedulers.

Resource Interleaving Muri [64] introduced the idea of multi-resource interleaving for DNN training, but required all jobs to share the same GPUs. Cassini [52] exploited the opportunity to overlap the computation and communication of different jobs using an ILP. Both of these works are centralized schedulers that would struggle to scale in a real system. In contrast, our implementation of MLTCP performs a distributed live optimization.

Unfairness in the network. Recently, there have been calls to introduce bandwidth unfairness to optimize flow completion times, energy efficiency, and DNN iteration time [6, 53, 63]. MLTCP automatically configures congestion control parameters to leverage unfairness to achieve these goals.

7 CONCLUSION

This paper introduces a technique to augment a family of congestion control algorithms to approximate optimal interleaving for DNN training jobs. The key idea is to dynamically adjust the flow congestion window (or sending rate) to induce a sliding effect so that the DNN jobs automatically converge to approximately optimal interleaving. We formalize our approach and show that the sliding effect iteration after iteration is equivalent to gradient descent, with the goal of improving communication interleaving.

Acknowledgements. We thank HotNets's anonymous reviewers for their valuable feedback. Thanks to Hari Balakrishnan for helpful suggestions and discussions. The authors are supported by DARPA FastNICs 4202290027, NSF SHF-2107244, NSF CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, and Sloan fellowship FG-2022-18504.

REFERENCES

[1] [n. d.]. NCCL Fast Socket. https://github.com/google/nccl-fastsocket.

[2] 2005. Pluggable congestion avoidance modules. https://lwn.net/Articles/128681/

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (San Jose, California) *(NSDI'10)*. USENIX Association, USA, 19.

[4] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference* (New Delhi, India) *(SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[5] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/2486001.2486031

[6] Serhat Arslan, Sundarajan Reneganathan, and Bruce Spang. 2023. Green With Envy: Unfair Congestion Control Algorithms Can Be More Energy Efficient. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (Cambridge, Massachusetts) *(HotNets '23)*. 8 pages.

[7] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2017. PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers. *IEEE/ACM Trans. Netw.* 25, 4 (aug 2017), 1954–1967. https://doi.org/10.1109/TNET.2017.2669216

[8] Andrea Baiocchi, Angelo Castellani, and Francesco Vacirca. [n. d.]. YeAH-TCP: Yet another highspeed TCP. ([n. d.]).

[9] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (Melbourne, Australia) *(IMC '10)*. Association for Computing Machinery, New York, NY, USA, 267–280. https://doi.org/10.1145/1879141.1879175

[10] L.S. Brakmo and L.L. Peterson. 1995. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* 13, 8 (1995), 1465–1480. https://doi.org/10.1109/49.464716

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 https://arxiv.org/abs/2005.14165

[12] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient Coflow Scheduling Without Prior Knowledge. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 393–406. https://doi.org/10.1145/2829988.2787480

[13] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. *SIGCOMM Comput. Commun. Rev.* 44, 4 (aug 2014), 443–454. https://doi.org/10.1145/2740070.2626315

[14] Wolfgang Dauscha, Heinz D. Modrow, and Alexander Neumann. 1985. On cyclic sequence types for constructing cyclic schedules. *Zeitschrift für Operations Research* 29 (1985), 1–30. https://api.semanticscholar.org/CorpusID:12356541

[15] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. 2005. Processor Sharing Flows in the Internet. In *Quality of Service – IWQoS 2005*, Hermann de Meer and Nina Bhatti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 271–285.

[16] M. Gerla, M.Y. Sanadidi, Ren Wang, A. Zanella, C. Casetti, and S. Mascolo. 2001. TCP Westwood: congestion window control using bandwidth estimation. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, Vol. 3. 1698–1702 vol.3. https://doi.org/10.1109/GLOCOM.2001.965869

[17] Joël Goossens. 2003. Scheduling of Offset Free Systems. *Real-Time Systems* 24 (03 2003), 239–258. https://doi.org/10.1023/A:1021782503695

[18] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 485–500. https://www.usenix.org/conference/nsdi19/presentation/gu

[19] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (jul 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[20] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[21] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. 2019. TicTac: Accelerating Distributed Deep Learning with Communication Scheduling. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 418–430. https://proceedings.mlsys.org/paper/2019/file/84d9ee44e457ddef7f2c4f25dc8fa865-Paper.pdf

[22] Stephen Hemminger. 2005. TCP infrastructure split out. http://lwn.net/Articles/128626/

[23] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. 2012. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.* 42, 4 (aug 2012), 127–138. https://doi.org/10.1145/2377677.2377710

[24] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. 2012. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Helsinki, Finland) *(SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 127–138. https://doi.org/10.1145/2342356.2342389

[25] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf

[26] V. Jacobson. 1988. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 314–329. https://doi.org/10.1145/52325.52356

[27] Akshay Jajoo, Y. Charlie Hu, and Xiaojun Lin. 2019. Your Coflow has Many Flows: Sampling them for Fun and Speed. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 833–848. https://www.usenix.org/conference/atc19/presentation/jajoo

[28] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. 2023. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 642–657. https://doi.org/10.1145/3600006.3613175

[29] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[30] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 26, 17 pages.

[31] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev.* 32, 4 (aug 2002), 89–102. https://doi.org/10.1145/964725.633035

[32] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[33] Matheus Ladeira, Emmanuel Grolleau, Fabien Bonneval, Gautier Hattenberger, Yassine Ouhammou, and Yuri Hérouard. 2022. Scheduling Offset-Free Systems Under FIFO Priority Protocol. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022) (Dagstuhl Artifacts Series, Vol. 231)*. Modena, Italy. https://doi.org/10.4230/DARTS.8.1.4

[34] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. 2017. DX: Latency-Based Congestion Control for Datacenters. *IEEE/ACM Transactions on Networking* 25, 1 (2017), 335–348. https://doi.org/10.1109/TNET.2016.2587286

[35] Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. 2021. TeraPipe: Token-Level Pipeline Parallelism for Training Large-Scale Language Models. arXiv:2102.07988 [cs.LG]

[36] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* 20, 1 (jan 1973), 46–61. https://doi.org/10.1145/321738.321743

[37] Shao Liu, Tamer Başar, and R. Srikant. 2008. TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation* 65, 6 (2008), 417–440. https://doi.org/10.1016/j.peva.2007.12.007 Innovative Performance Evaluation Methodologies and Tools: Selected Papers from ValueTools 2006.

[38] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

[39] Kshiteej Mahajan, Ching-Hsiang Chu, Srinivas Sridharan, and Aditya Akella. 2023. Better Together: Jointly Optimizing ML Collective Scheduling and Execution Planning using SYNDICATE. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 809–824. https://www.usenix.org/conference/nsdi23/presentation/mahajan

[40] Luo Mai, Chuntao Hong, and Paolo Costa. 2015. Optimizing Network Performance in Distributed Machine Learning. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/mai

[41] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.* 27, 3 (jul 1997), 67–82. https://doi.org/10.1145/263932.264023

[42] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 537–550. https://doi.org/10.1145/2829988.2787510

[43] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 579–596. https://www.usenix.org/conference/osdi22/presentation/mohan

[44] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 221–235. https://doi.org/10.1145/3230543.3230564

[45] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP'19)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3341301.3359646

[46] Rui Pan, Yiming Lei, Jialong Li, Zhiqiang Xie, Binhang Yuan, and Yiting Xia. 2022. Efficient flow scheduling in distributed deep learning training with echelon formation. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/3563766.3564096

[47] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 14 pages. https://doi.org/10.1145/3190508.3190517

[48] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A Generic Communication Scheduler for Distributed DNN Training Acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 16–29. https://doi.org/10.1145/3341301.3359642

[49] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 307–318. https://doi.org/10.1145/2619239.2626309

[50] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P.

Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 1–18. https://www.usenix.org/conference/osdi21/presentation/qiao

[51] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language Models are Unsupervised Multitask Learners. (2018). https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[52] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1403–1420. https://www.usenix.org/conference/nsdi24/presentation/rajasekaran

[53] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. 2022. Congestion Control in Machine Learning Clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. 235–242.

[54] Linus Schrage. 1968. Letter to the Editor—A Proof of the Optimality of the Shortest Remaining Processing Time Discipline. *Operations Research* 16, 3 (1968), 687–690. https://doi.org/10.1287/opre.16.3.687

[55] Linus E. Schrage and Louis W. Miller. 1966. The Queue M/G/1 with the Shortest Remaining Processing Time Discipline. *Operations Research* 14, 4 (1966), 670–684. https://doi.org/10.1287/opre.14.4.670

[56] Paolo Serafini and Walter Ukovich. 1989. A Mathematical Model for Periodic Scheduling Problems. *SIAM Journal on Discrete Mathematics* 2, 4 (1989), 550–581. https://doi.org/10.1137/0402049 arXiv:https://doi.org/10.1137/0402049

[57] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 593–612. https://www.usenix.org/conference/nsdi23/presentation/shah

[58] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. 2012. Deadline-aware datacenter tcp (D2TCP). In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Helsinki, Finland) *(SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 115–126. https://doi.org/10.1145/2342356.2342388

[59] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 739–767. https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang

[60] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 50–61. https://doi.org/10.1145/2018436.2018443

[61] 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. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[62] Lisong Xu, K. Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE INFOCOM 2004*, Vol. 4. 2514–2524 vol.4. https://doi.org/10.1109/INFCOM.2004.1354672

[63] Adrian Zaplatel and Fernando Kuipers. 2023. Slowdown as a Metric for Congestion Control Fairness. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (Cambridge, Massachusetts) *(HotNets '23)*. 8 pages.

[64] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. 2022. Multi-Resource Interleaving for Deep Learning Training. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 428–440. https://doi.org/10.1145/3544216.3544224

[65] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 703–723. https://www.usenix.org/conference/nsdi23/presentation/zheng

[66] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484