Congestion Control in Machine Learning Clusters

Sudarsanan Rajasekaran
Massachusetts Institute of Technology

Manya Ghobadi
Massachusetts Institute of Technology

Gautam Kumar
Google

Aditya Akella
UT Austin

ABSTRACT

This paper argues that fair-sharing, the holy grail of congestion control algorithms for decades, is not necessarily a desirable property in Machine Learning (ML) training clusters. We demonstrate that for a specific combination of jobs, introducing unfairness improves the training time for *all* competing jobs. We call this specific combination of jobs *compatible* and define the compatibility criterion using a novel geometric abstraction. Our abstraction *rolls time* around a circle and *rotates* the communication phases of jobs to identify fully compatible jobs. Using this abstraction, we demonstrate up to 1.3× improvement in the average training iteration time of popular ML models. We advocate that resource management algorithms should take job compatibility on network links into account. We then propose three directions to ameliorate the impact of network congestion in ML training clusters: (*i*) an adaptively unfair congestion control scheme, (*ii*) priority queues on switches, and (*iii*) precise flow scheduling.

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, Manya Ghobadi, Gautam Kumar, and Aditya Akella. 2022. Congestion Control in Machine Learning Clusters. In *The 21st ACM Workshop on Hot Topics in Networks (HotNets '22), November 14–15, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3563766.3564115

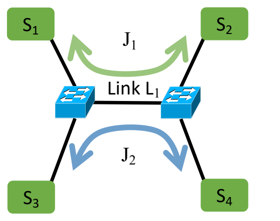
1 INTRODUCTION

The ever-growing increase in dataset and model sizes of deep neural networks (DNNs) has created a massive demand for efficient GPU clusters. Several studies have demonstrated that as the number of GPUs increases, the communication overhead of distributed Machine Learning (ML) training workloads quickly takes up a significant portion of training iteration time [6, 11, 20, 24, 29, 30, 38, 48].

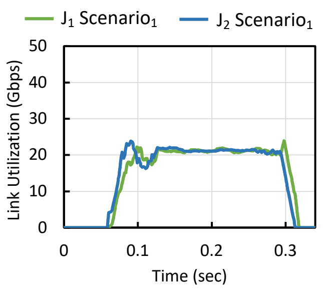
To alleviate the communication overhead of distributed ML training, many training platforms overlap the compute and communication phases of *a single job* using pipelining [30], smart memory management [37], or prioritized parameter transfers [19, 21, 32]. But these approaches tend to consider a job in isolation, and the impact of congestion control algorithms, when two or more training jobs share a bottleneck link is largely ignored. Today's systems for ML simply attempt to place workers of the same job close to each other to minimize the probability of congesting the network and rely on default TCP or RDMA protocols to handle congestion [18, 23, 26, 28, 31, 32, 34, 51]. For instance, Themis [26] uses a slowdown factor to give preference to place workers of the same job under the same Top-of-Rack (ToR) switch, but it does not take network congestion into account when workers are spread across ToRs.

Even with careful job placement, cross-job network contention is inevitable in large-scale GPU clusters. Today, when two or more ML jobs compete for bandwidth, congestion control approaches share the network resources *fairly*, but we demonstrate that for a specific combination of jobs, introducing unfairness creates a *desirable side effect* that improves the training time of all jobs competing for bandwidth (§2). Essentially, unfairness interleaves the computation and communication phases of different jobs, enabling them to claim the network bandwidth one-at-a-time, thereby improving the training time of all competing jobs. We refer to this set of jobs as *compatible*.

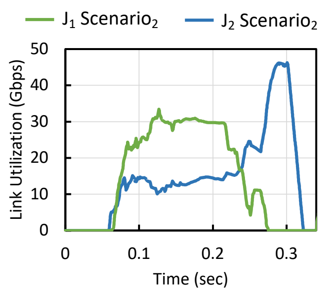
However, if jobs are incompatible, unfairness unnecessarily hurts those that are less aggressive. Identifying compatible jobs is challenging because it depends on the duration of compute and communication phases of competing jobs, and their network bandwidth requirements. To address this challenge,



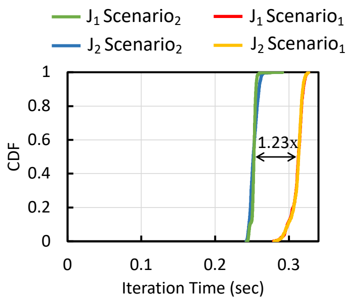
(a) Our testbed with four A100 GPU servers, 50 Gbps links, and Mellanox RDMA NICs



(b) First iteration in scenario$_1$: Fair bandwidth sharing



(c) First iteration in scenario$_2$: Unfair bandwidth sharing



(d) CDF of iteration times (J_1 and J_2 curves in each scenario overlap)

Figure 1: Impact of unfairness on training iteration time of two VGG19 training jobs sharing bottleneck link L_1.

we propose a geometric abstraction that leverages the periodic on-off pattern of DNN training (§3). The key idea of our abstraction is to *roll* time around a circle whose perimeter is proportional to the training iteration time of the ML jobs.

Using our geometric abstraction, we argue ML cluster operators can manage network congestion by ensuring fully compatible jobs are placed on the same network links, and introducing some form of unfairness to enable cross-job communication/computation interleaving across compatible jobs (§4). Toward this goal, we propose three potential future directions that use our geometric abstraction to alleviate congestion in ML training clusters : (*i*) an adaptively unfair congestion control scheme, (*ii*) priority queues on switches, and (*iii*) precise flow scheduling.

2 SURPRISING PAYOFF OF UNFAIRNESS

Distributed DNN training. A common approach for training large DNN models is data parallelism, where the training data is distributed across multiple accelerators. During each training iteration, accelerators need to synchronize their model weights. This step is called *allreduce* and can be performed using various techniques, such as broadcasting [53], parameter servers [25], ring-allreduce [1, 22, 44], tree-reduce [35], or hierarchical ring-allreduce [45, 46].

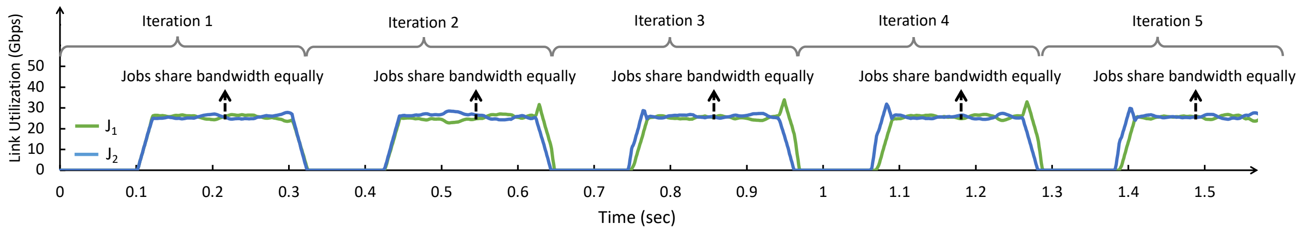
Compute and communication phases. To consider the impact of network congestion in data parallel training jobs, we refer to the forward pass as the *compute phase* and refer to the backpropagation and allreduce phases *together* as the *communication phase* because congestion impacts any period of time when data is being injected into the network.

Pipeline parallelism. To speed up training, many platforms pipeline the computation in the backpropagation step with the communication in the allreduce step [19–21, 30, 32, 37]. Pipelining techniques are effective at overlapping the computation and communication phases of the same job, but they ignore the interaction between multiple jobs when they share a bottleneck link.

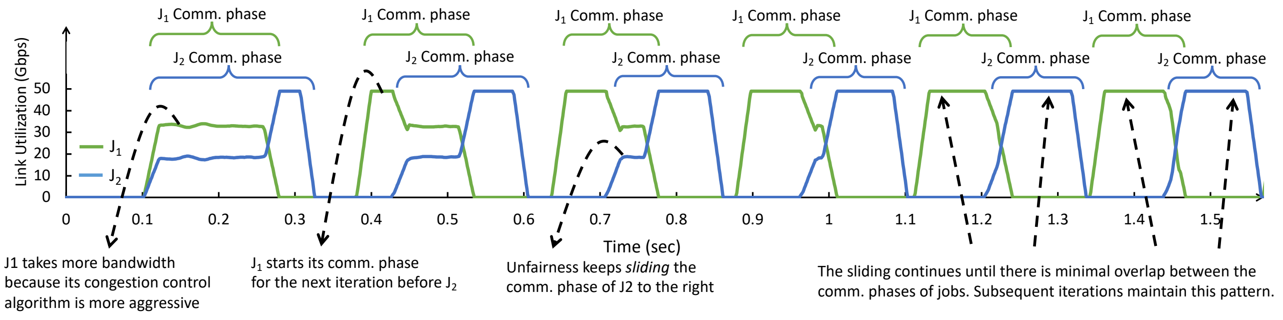
Goal. We consider GPU training clusters where large DNN models are distributed across GPUs. Our ultimate goal is to avoid network congestion to slow down the training time of jobs sharing a link. We argue that achieving this goal does not always require augmenting the network bandwidth, and *compatible jobs* can share network links without experiencing any slowdowns, as if the jobs are running with dedicated network resources.

A surprising observation. We begin our argument with an observation from a testbed with A100 GPUs and ConnectX-5 Mellanox NICs, shown in Figure 1a. The capacity of each NIC is 50 Gbps. We run two distributed DNN training jobs, called J_1 and J_2, across the servers such that they share a bottleneck link L_1. We run each job for 1,000 iterations under two scenarios. In the first scenario, two VGG19 [41] training jobs start simultaneously and share L_1 fairly under the default RDMA-based DCQCN congestion control algorithm [57]. DCQCN has a parameter T that corresponds to the time period of its rate increase. The default value of T in our testbed is 125 μs. Figure 1b shows both jobs achieve roughly 21 Gbps bandwidth (i.e., half of L_1's capacity) during the first iteration. This is not surprising, as DCQCN is designed to divide the capacity equally between jobs [58]. In the second scenario, we artificially introduce unfairness by adjusting DCQCN's T parameter on J_1's servers to 100 μs, making it more aggressive. As a result, in the very first iteration, J_1 achieves roughly 30 Gbps of bandwidth, whereas J_2 achieves 15 Gbps, as shown in Figure 1c. At first blush, it appears continuous unfairness will hurt J_2's iteration time in subsequent iterations. But we find that as training progresses, unfairness helps both J_1 and J_2. Figure 1d plots the CDF of training iteration times for both scenarios, demonstrating that the unfairness in scenario$_2$ accelerates the median iteration time of both jobs by 1.23×.

Why would unfairness in congestion control help ML training? DNN training jobs have a unique on-off pattern [26, 32, 34, 51] where the off period corresponds to the compute phase and the on period represents the communication phase.



(a) Fair bandwidth sharing



J1 takes more bandwidth because its congestion control algorithm is more aggressive

J_1 starts its comm. phase for the next iteration before J_2

Unfairness keeps *sliding* the comm. phase of J2 to the right

The sliding continues until there is minimal overlap between the comm. phases of jobs. Subsequent iterations maintain this pattern.

(b) Unfair bandwidth sharing

Figure 2: Comparing the iteration times with fair and unfair bandwidth allocations.

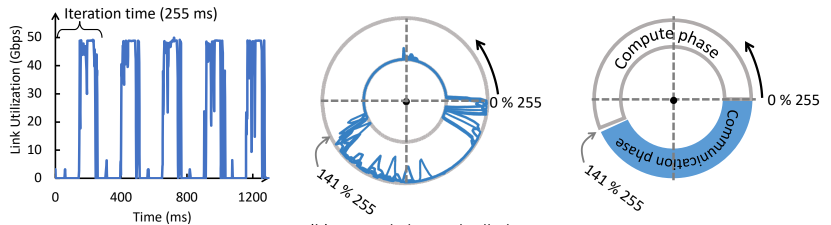
Intuitively, when two training jobs share a network link, fair bandwidth sharing slows down both jobs by prolonging their communication phases. In contrast, unfair bandwidth sharing speeds up one job while slowing down the other, creating a side effect that *slides* the on-off pattern of the two competing jobs to fit into each other after a few iterations.

Figure 2 demonstrates the sliding impact for the two scenarios in Figure 1 by showing the link utilization of back-to-back iterations. For clarity of presentation, we assume both jobs start at the same time, and we smooth out the plots to help with the visualization. Figure 2a shows that when the bottleneck link is shared fairly, both jobs continue to occupy ≈50% of the available bandwidth across all iterations. In contrast, Figure 2b shows that for the very first iteration, unfairness gives more bandwidth to J_1, allowing it to accelerate and complete the first iteration at $t = 0.28$ sec, whereas J_2 takes longer and completes its first iteration at $t = 0.32$ sec. This imbalance means the second communication phase of J_1 starts earlier (at $t = 0.38$ sec) and utilizes the full bandwidth temporarily, while the second communication phase of J_2 starts later (at $t = 0.42$ sec). Similarly, in the second iteration, when both jobs are communicating, due to unfairness, J_1 occupies a bigger share of the bandwidth. Interestingly, the region where both jobs compete for network communication gradually reduces as we move from the first iteration to the fourth iteration. By the fourth iteration, unfairness pulls apart the communication phases of the jobs and interleaves the computation phase of J_1 with the communication phase of J_2 perpetually. Hence, the iteration times of both jobs become almost equal to what they would have been had the jobs been running in a dedicated cluster; i.e., faster than fair sharing.

Jobs competing for bandwidth (batch size)	Fairness iter. time	Unfairness iter. time	Unfairness speed-up	Fully compatible
BERT (8)	183 ms	157 ms	1.17×	✗
VGG19 (1200)	297 ms	315 ms	0.94×	
DLRM (2000)	1301 ms	1001 ms	1.3×	✓
DLRM (2000)	1300 ms	1019 ms	1.28×	
BERT (8)	320 ms	216 ms	1.48×	
VGG19 (1400)	494 ms	466 ms	1.06×	✗
WideResNet (800)	466 ms	505 ms	0.92×	
WideResNet (800)	295 ms	273 ms	1.08×	✓
VGG16 (1400)	294 ms	274 ms	1.07×	
VGG19 (1400)	389 ms	329 ms	1.18×	
VGG16 (1700)	389 ms	329 ms	1.18×	✓
ResNet50 (1600)	167 ms	165 ms	1.01×	

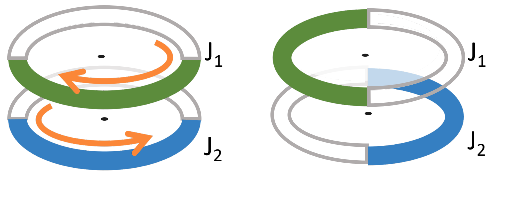
Table 1: Unfairness speeds up only fully compatible jobs.

Is unfairness helpful for all ML jobs? It turns out that the above desirable side effect of unfair bandwidth sharing can only help jobs whose communication and computation phases can fit perfectly into each other. As Table 1 shows, unfairness only helps a specific combination of jobs. Each row represents a different group of popular DNN training jobs (and batch sizes). We first measure the average iteration time when each group of jobs competes for bandwidth using the default (fair) DCQCN algorithm. Then, for each group of jobs, we make the DCQCN algorithm unfair such that the order of aggressiveness is based on the jobs' order of appearance in the table, with each job more aggressive than subsequent jobs in its row. The first group shows that when BERT [15] and VGG19 [41] models share a link, making BERT more aggressive ends up negatively hurting the iteration time of VGG19. But the second group shows when two DLRM [2]



(a) Network demand (b) Network demand rolled around a circle (c) Geometric abstraction

Figure 3: Our geometric abstraction.



(a) Communication phases are colliding (b) Communication phases are interleaved

Figure 4: Jobs J_1 and J_2 are compatible.

models share a link, making the first DLRM more aggressive accelerates the average iteration time of *both* jobs by 1.28 × −1.3×, compared to fair bandwidth sharing. The green color indicates the set of jobs for which unfairness leads to faster iteration times than fair bandwidth sharing. We refer to a group of jobs for which unfairness results in faster iterations time for all the jobs in the group as *compatible jobs*.

3 GEOMETRIC ABSTRACTION

To determine whether a set of jobs competing on a link is compatible, we seek to answer the following question: "*Is there a way to slide the communication pattern of the jobs such that their communication phases have almost no overlap with each other?*

Roll time around a circle. To answer above question, we propose a novel geometric abstraction. Consider the time-series representation of the network demand for a job running in a dedicated cluster with no congestion. Given the periodic on-off pattern of DNN training, the duration of the compute and communication phases remains more or less the same across training iterations. Consequently, if we *roll* time around a circle whose perimeter is equal to the training iteration time, the communication phases of all iterations will appear approximately on the same arc of the circle. For instance, Figure 3a illustrates the time-series network demand of VGG16 with a training iteration time of 255 ms where the first 141 ms are pure computation (i.e., forward pass). Figure 3b shows a circle with perimeter 255 time units and the time-series data plotted around it in a counter-clockwise direction. The compute phase of all iterations occupies the arc starting at 0 and ending at 141 time units, and the communication phases span the rest of the circle. This representation demonstrates that the compute and communication phases of different iterations always cover the same arcs of the circle. We design our geometric abstraction to capture this circular property. Figure 3c shows our geometric abstraction. The circle's perimeter represents the iteration time, set to 255 time units. The compute phase spans 141 time units, represented by the uncolored arc, and the communication phase, represented by the colored arc, occupies the remainder of the circle.

Rotate the circle to avoid congestion. To determine the compatibility of two (or more) jobs, we place each job on its corresponding circle and overlay the circles on top of each other. Congestion happens when the communication phases collide, as shown in Figure 4a. To avoid congestion, we rotate the circles to find a position where the communication arcs do not collide, as shown in Figure 4b. If such a rotation is found, the jobs are deemed compatible. Note that rotating the circles clockwise and counterclockwise is equivalent to the sliding effect of unfairness. Moreover, overlapping the computation times of each job (the uncolored regions) is acceptable, as we assume jobs are not sharing the compute resources with each other.

How to capture jobs with different iteration times? The above technique is only applicable when circles have the same perimeter. To generalize our geometric abstraction to the case where jobs have different iteration times, we place each job on a *unified circle* whose perimeter is equal to the Least Common Multiple (LCM) of the iteration times of all jobs competing on the link. For instance, consider two jobs J_1 and J_2 competing on a bottleneck link with iteration times 40 ms and 60 ms, respectively. To determine the compatibility of these two jobs, we place them on a circle with a perimeter equal to $LCM(40, 60) = 120$ units. Figure 5a shows J_1 on this unified circle. Given that the perimeter of the circle is 3× J_1's iteration time, there are three communication and computation phases in the figure. Similarly, Figure 5b shows J_2 on the unified circle. We then overlay the unified circles (shown in Figure 5c) and rotate them to determine whether the jobs are compatible. Figure 5d shows that when J_1 is rotated 30° counterclockwise, the colored areas on the circles do not collide; i.e., the jobs are fully compatible.

Optimization formulation. We use an optimization formulation to determine whether a set of jobs is fully compatible and if so, what the best angle of rotation is for each job such that the communication phases do not overlap. Our formulation searches for rotation angles such that there is no region on the unified circle where more than one job is communicating. For scalability, we discretize the circle into small sectors and add constraints capping the number of jobs

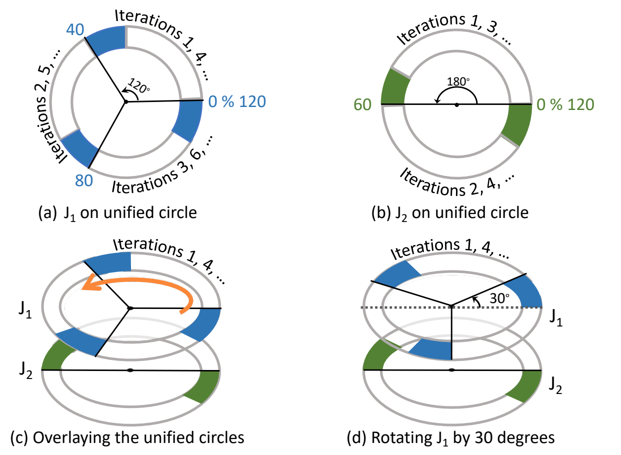


Figure 5: Geometric abstraction for jobs with different training iteration times using a unified circle.

communicating in each sector at one. If the optimization formulation finds the rotation angles satisfying the constraints, the jobs are deemed compatible.[1]

4 CONGESTION-FREE ML CLUSTERS

Today's ML cluster scheduling techniques consider end-hosts that are topologically near each other as the main criterion for reducing network congestion [18, 23, 26, 28, 31, 32, 34, 51]. This section argues for two requirements to move toward congestion-free ML clusters. First, ML schedulers should be augmented to take compatibility into account and to place compatible jobs on network links. In other words, the problem of job placement should be related not only to available resources on servers but also to compatibility on links. Second, once compatible jobs are placed on the same link(s), service providers need to artificially create the desirable side effect of unfairness to enable compatible jobs to occupy the entire link bandwidth without slowing each other down.

Placing compatible jobs on links. To place compatible jobs on network links, the ML scheduler should first profile each ML training job in isolation to measure its iteration time, communication pattern, and bandwidth demand for different hyper-parameters. Next, the scheduling algorithm should become aware of network routes (e.g., ECMP routing decisions) for each job. Once the routes are known, the scheduler runs our optimization formulation for the set of jobs placed on a network link to determine their compatibility. If the jobs are incompatible, the scheduler should look for alternative placements. We believe this approach applies to today's ML schedulers, including BytePS [23, 32], Themis [26], Pollux [34], and Muri [56].

Creating unfairness for compatible jobs. After placing compatible jobs on network links, the next step is to avoid colliding the communication phases of jobs, so that they can co-exist on the same link while using the link bandwidth one at a time. This property can be achieved by: (*i*) deploying an unfair congestion control algorithm; or (*ii*) using packet priorities to achieve unfairness; or (*iii*) scheduling the communication phases in the appropriate time slots. Below we summarize each approach.

Using an unfair transport protocol. Intuitively, deploying an unfair congestion control algorithm throughout a cluster seems like a bad idea. In particular, as shown in Table 1, when incompatible jobs share a link, our unfair transport protocol favors more aggressive jobs and slows down the less aggressive ones without creating any desirable side effects. However, we argue that an *adaptively* unfair congestion control algorithm can achieve the desired side effect of unfairness when the jobs are compatible without negatively impacting incompatible jobs. For instance, the DCQCN algorithm [57, 58] uses the following equation to determine the increase in its target sending rate: $R_T = R_T + R_{AI}$, where R_T represents the target sending rate, and R_{AI} denotes the Additive Increase step. To enable adaptive unfairness, we adjust R_{AI} from a constant to $R_{AI}(1 + \frac{Data_{sent}}{Data_{comm.\ phase}})$, where $Data_{sent}$ represents the amount of progress in the communication phase. Hence, a job closer to completing its communication phase is more aggressive than a job just about to start its communication phase ($Data_{sent} = 0$), enabling interleaving of compatible jobs. Meanwhile, incompatible jobs continue to take turns in becoming the aggressive party to claim the bandwidth, and they end up sharing the bandwidth fairly in steady state.

Using priority queues. An alternative approach is to use priority queues in network switches. This approach is similar to prior techniques using application-aware semantics in datacenters to achieve differential performance [3, 4, 12–14, 16, 17, 55]. For ML workloads, the assigned priority for jobs can be arbitrary as long as the jobs competing for the same link are compatible and have a *unique* priority. In this approach, the scheduler assigns the priority value to each job sharing the same link. Then, the end-hosts mark packets with the assigned priority, allowing the switch [36, 42] to divide the link bandwidth accordingly, thereby mimicking the desirable side effect of unfairness. This approach does not require any changes to the congestion control algorithm. However, a potential challenge is that today's switches support a few priority queues; thus, maintaining unique priorities when there is a large number of jobs becomes challenging.

Flow scheduling. Instead of creating explicit unfairness in the congestion control algorithm, service providers can use the centralized scheduler to schedule flows for each job at precise time intervals. Concretely, the output of our optimization formulation provides an angle of rotation for each job such that the communication phases do not collide. This angle

[1]The details of the optimization formulation are omitted for brevity.

corresponds to a time-shift for the communication phase of a job. Using this time-shift, the scheduler can schedule flows at appropriate times to avoid colliding the communication phases of jobs sharing network links. This approach is similar to flow-scheduling techniques in datacenters [33, 39, 47]. However, it is challenging to schedule short transfers at precise times without a high-resolution clock synchronization across the cluster.

5 DISCUSSION

Cluster-level compatibility. In large-scale clusters, jobs are likely to traverse multiple links, and they may compete with different jobs on different links. Given the interdependence of all servers participating in a training job, service providers must ensure compatibility is preserved across all links. A potential solution to address this is to expand the perimeter of the unified circle to become the LCM of the iteration times of the jobs sharing at least one link with other jobs and solving the optimization formulation to find a unique rotation angle for each job.

GPU multi-tenancy. For simplicity, we assume GPUs are dedicated resources for each job, and different jobs do not share the same GPU – this is not far from how many production clusters run today to ensure predictable and high throughput training performance. Thus, our geometric abstraction only considers the network links as shared resources and allows the compute phases of different jobs to overlap. Recent proposals demonstrate the feasibility of multi-tenancy on GPUs [5, 49, 51, 52, 56]. We note that capturing GPU multi-tenancy is possible by adding more constraints in our optimization formulation, but we omit the details for brevity.

Impact of hyper-parameters. The iteration time and communication demand of a job are affected by the batch size, the number of workers, and the all-reduce algorithm. If these hyper-parameters are changed during the lifetime of a job, its geometric abstraction changes accordingly, and the scheduler should take the new abstraction into account. This also provides an opportunity for the scheduler to adjust the hyper-parameters to improve the compatibility of jobs sharing links while making job placement decisions.

6 RELATED WORK

Congestion control for ML. RDMA is currently the standard technology used in ML clusters. Congestion control algorithms for RDMA include DCQCN [57, 58], IRN [27], and RoCC [43]. These schemes strive to achieve fairness across all flows sharing a link and do not leverage the unique properties of ML workloads, such as periodicity and predictable network demand. Xia et al. [50] leverage the loss-tolerance of ML training and propose a bounded-loss tolerant transport as a new congestion control paradigm for ML training workloads. In contrast, our approach does not require bounding the loss.

Scheduling techniques. DNN scheduling algorithms decide where jobs are placed in the cluster. Today's schedulers, such as Gandiva [51], Themis [26], Pollux [34], Tiresias [18], ByteScheduler [23, 32], and Optimus [31], try to minimize congestion by placing workers of the same job as close to each other as possible. But these approaches do not consider placing compatible jobs on network links to avoid sharing the network bandwidth. Flow-scheduling approaches such as Sincronia [3], Orchestra [9], and Coflows [7, 8, 10, 39, 40, 54] provide differential treatment for flows instead of sharing the network fairly. But these approaches are designed for legacy datacenter traffic and do not leverage the periodic behavior of ML traffic. In contrast, our approach is custom-designed for ML workloads. Muri [56] and Synergy [28] recently proposed DL scheduling techniques to interleave critical resources (e.g., GPU, CPU, network, storage) of DL training jobs. However, Muri considered a restrictive setting where resource interleaving is only possible for jobs distributed across the exact same set of servers, and Synergy only considered GPU, CPU, and memory as critical resources. Our approach captures a more generic case where network links are shared across jobs, irrespective of the set of GPUs they use.

7 CONCLUSION

We present a surprising finding: unfair bandwidth allocation helps certain combinations of DNN jobs achieve congestion-free performance even though the network is shared. We formalize our finding by defining a novel geometric abstraction to capture job compatibility and argue ML schedulers should use an optimization formulation to take job compatibility into account. We discuss potential approaches to systematically unlock this opportunity to optimize network sharing for compatible jobs.

Acknowledgments. We would like to thank HotNets' anonymous reviewers for their valuable feedback. Special thanks to Zhizhen Zhong for help with the optimization formulation, Weiyang Wang for help with the testbed, Mingran Yang for help with the Tofino switch, Benoit Pit–Claudel, Pantea Karimi, Dousabel Tay, Kapil Vaidya, Nandita Dukkipati, and Amin Vahdat for giving valuable suggestions. The MIT-affiliated authors are supported by ARPA-E ENLITENED PINE DE-AR0000843, DARPA FastNICs 4202290027, NSF CNS-2008624, NSF SHF-2107244, NSF ASCENT-2023468, NSF CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, Meta faculty award, Google faculty award, and Sloan fellowship FG-2022-18504. The UT Austin author is supported by NSF CNS-2105890, NSF CNS-2106199 and gifts from Meta and Cisco.

REFERENCES

[1] Baidu, 2017. https://github.com/baidu-research/baidu-allreduce.

[2] Deep Learning Recommendation Model for Personalization and Recommendation Systems, 2021. https://github.com/facebookresearch/dlrm.

[3] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat. Sincronia: Near-optimal network design for coflows. SIGCOMM '18, page 16–29, New York, NY, USA, 2018. Association for Computing Machinery.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 435–446, New York, NY, USA, 2013. ACM.

[5] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu. Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency. *SIGPLAN Not.*, 53(2):503–518, mar 2018.

[6] M. A. Chang, A. Panda, D. Bottini, L. Jian, P. Kumar, and S. Shenker. Network evolution for dnns. *SysML*, 2018.

[7] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, page 31–36, New York, NY, USA, 2012. Association for Computing Machinery.

[8] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 393–406, New York, NY, USA, 2015. Association for Computing Machinery.

[9] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 98–109, New York, NY, USA, 2011. Association for Computing Machinery.

[10] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 443–454, New York, NY, USA, 2014. Association for Computing Machinery.

[11] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengil, M. Liu, D. Lo, S. Alkalay, and M. Haselman. Accelerating persistent neural networks at datacenter scale. In *Hot Chips*, volume 29, 2017.

[12] R. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, 1995.

[13] R. L. Cruz. Service burstiness and dynamic burstiness measures: A framework. *J. High Speed Netw.*, 1(2):105–127, apr 1992.

[14] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19(4):1–12, aug 1989.

[15] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[16] N. Figueira and J. Pasquale. Rate-function scheduling. In *Proceedings of INFOCOM '97*, volume 3, pages 1063–1071 vol.3, 1997.

[17] N. R. Figueira and J. Pasquale. Leave-in-time: A new service discipline for real-time communications in a packet-switching network. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '95, page 207–218, New York, NY, USA, 1995. Association for Computing Machinery.

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

[19] S. H. Hashemi, S. Abdu Jyothi, and R. Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 418–430, 2019.

[20] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS*, 2019.

[21] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko. Priority-based parameter propagation for distributed dnn training. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 132–145, 2019.

[22] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018.

[23] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo. A unified architecture for accelerating distributed DNN training in heterogeneous gpu/cpu clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479. USENIX Association, Nov. 2020.

[24] M. Khani, M. Ghobadi, M. Alizadeh, Z. Zhu, M. Glick, K. Bergman, A. Vahdat, B. Klenk, and E. Ebrahimi. Sip-ml: High-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, pages 657–675, New York, NY, USA, 2021. Association for Computing Machinery.

[25] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. OSDI'14, pages 583–598. USENIX Association, 2014.

[26] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 289–304, Santa Clara, CA, Feb. 2020. USENIX Association.

[27] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery.

[28] J. Mohan, A. Phanishayee, J. J. Kulkarni, and V. Chidambaram. Looking beyond gpus for dnn scheduling on multi-tenant clusters. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI 2022)*, July 2022.

[29] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao. Software-hardware co-design for fast and scalable training of deep learning recommendation models, 2021.

[30] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP'19, pages 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings*

of the Thirteenth EuroSys Conference, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[32] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 16–29, New York, NY, USA, 2019. Association for Computing Machinery.

[33] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 307–318, New York, NY, USA, 2014. Association for Computing Machinery.

[34] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 1–18. USENIX Association, July 2021.

[35] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[36] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 239–250, New York, NY, USA, 2003. Association for Computing Machinery.

[37] S. Rashidi, M. Denton, S. Sridharan, S. Srinivasan, A. Suresh, J. Nie, and T. Krishna. *Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms*, page 540–553. IEEE Press, 2021.

[38] A. Sergeev and M. D. Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[39] M. Shafiee and J. Ghaderi. Scheduling coflows in datacenter networks: Improved bound for total weighted completion time. In *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '17 Abstracts, page 29–30, New York, NY, USA, 2017. Association for Computing Machinery.

[40] M. Shafiee and J. Ghaderi. Scheduling coflows with dependency graph. *IEEE/ACM Trans. Netw.*, 30(1):450–463, feb 2022.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[42] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 44–57, New York, NY, USA, 2016. Association for Computing Machinery.

[43] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster, and T. Edsall. *RoCC: Robust Congestion Control for RDMA*, page 17–30. Association for Computing Machinery, New York, NY, USA, 2020.

[44] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, Feb. 2005.

[45] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, Feb. 2005.

[46] Y. Ueno and R. Yokota. Exhaustive study of hierarchical allreduce patterns for large messages between gpus. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 430–439, 2019.

[47] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical tdma for datacenter ethernet. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 225–238, New York, NY, USA, 2012. Association for Computing Machinery.

[48] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. Kewitsch. Topoopt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.

[49] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, Renton, WA, Apr. 2022. USENIX Association.

[50] J. Xia, G. Zeng, J. Zhang, W. Wang, W. Bai, J. Jiang, and K. Chen. Rethinking transport layer design for distributed machine learning. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, APNet '19, page 22–28, New York, NY, USA, 2019. Association for Computing Machinery.

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

[52] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548. USENIX Association, Nov. 2020.

[53] P. Xie, J. K. Kim, Y. Zhou, Q. Ho, A. Kumar, Y. Yu, and E. Xing. Lighter-communication distributed machine learning via sufficient factor broadcasting. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 795–804, Arlington, Virginia, USA, 2016. AUAI Press.

[54] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 160–173, New York, NY, USA, 2016. Association for Computing Machinery.

[55] L. Zhang. Virtualclock: A new traffic control algorithm for packet-switched networks. *ACM Trans. Comput. Syst.*, 9(2):101–124, may 1991.

[56] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 428–440, New York, NY, USA, 2022. Association for Computing Machinery.

[57] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery.

[58] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye. Ecn or delay: Lessons learnt from analysis of dcqcn and timely. In *CoNEXT'16*, September 2016.