Network Requirements for Distributed Machine Learning Training in the Cloud

by

James Salamy

BE(ECS)(Hons) & BSc, Monash University, 2018

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2022

Author .....
Department of Electrical Engineering and Computer Science
October 15, 2021

Certified by .....
Manya Ghobadi
TIBCO Career Development Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Network Requirements for Distributed Machine Learning Training in the Cloud

by

James Salamy

Submitted to the Department of Electrical Engineering and Computer Science
on October 15, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

In this thesis, I characterize the impact of network bandwidth on distributed machine learning training. I test four popular machine learning models (ResNet, DenseNet, VGG, and BERT) on an Nvidia A-100 cluster to determine the impact of bursty and non-bursty cross traffic (such as web-search traffic and long-lived flows) on the iteration time and throughput of distributed training. By varying the cross traffic load, I measure the impact of network congestion on training iteration times. I observe that with heavy web-search cross traffic (80% of link capacity), on average training iteration time is increased by up to 4 to 8×, for ResNet and BERT models, respectively. Further, I establish that the ring-all reduce communication collective is negatively impacted by network congestion even if the congestion is only affecting part of the ring. I also develop empirical models for the behavior of machine learning training in the presence of each type of cross traffic deployed. These results provide the motivation for developing novel congestion control protocols that are tailored for distributed training environments.

Thesis Supervisor: Manya Ghobadi
Title: TIBCO Career Development Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to extend my sincere gratitude to my advisor, Professor Manya Ghobadi, who has supported me across the last two years through many trials, and has constantly worked to make me a better researcher. You have inspired me, and without your support and belief I would not be where I am today.

Secondly, I would like to thank the members of the HiPerSys group and the wider Networks and Mobile Systems group at MIT CSAIL for their advice, friendship, and feedback. I would also especially like to thank Weiyang Wang, Mingran Yang, Sudarsanan Rajasekaran, Homa Esfahanizadeh, and Amir Farhat.

To the Heads of House and the Student Government of Sidney Pacific, thank you for creating a warm and welcoming community, and a home away from home for me during my time in Cambridge.

To Professor Leslie Kolodziejski and the entire EECS graduate office, and my academic advisor Professor Hae-Seung Lee, thank you for your advice, recommendations, and support throughout my time at MIT.

To my family and friends, especially my parents Andrew and Lynn, and my brother, Mark, thank you from the bottom of my heart for your unconditional support, advice, and proofreading skills. To my friends at MIT and in Australia, especially Kiyah and Katie, thank you for your organization skills, company, help, and as always for your patience, without which this process would have been impossible.

Finally, to my grandfather Malcolm. When I was considering grad school, you pushed me to take the opportunity in front of me. Thank you for teaching me, telling stories, and inspiring me to reach further.

This research was sponsored in part by MIT's Google Cloud Engine Program, the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Distributed Machine Learning (DL) has been a field of growing interest and importance for the last several years, in line with the growth of Machine Learning (ML) as a key tool in research and industry used to solve a wide range of technical challenges. The ever-growing demand for more accurate ML models has resulted in a steady increase in dataset and model sizes used in deep neural networks (DNN). ML researchers develop new models at a rapid rate [7, 8], leading to steady growth in computational and memory requirements.

Figure 1-1 demonstrates the rate of this increase by plotting published model parameter counts [1], transistor counts for CPU designs [2], and floating point operations per second (FLOPS) for ML accelerators [3]. This equates to an increase in model size of $\sim$100$\times$ every two years; this is 50$\times$ faster growth rate than the pace predicted by Moore's Law. DL has therefore become the focus of research efforts as modern ML models and datasets are too large and have been growing too quickly to be supported by developments in individual servers and accelerators [9, 10, 11].

The increase in ML training performance demands has been partly met by the development of specialized hardware accelerators and software stacks supporting efficient localized processing and distributed workloads. While hardware accelerators such as Nvidia's Ampere Architecture [12], or domain-specific examples such as Microsoft's Catapult FPGAs [13] or Google's TPUs [14], have provided a significant amount of speed-up compared to preexisting options, today's training tasks can still

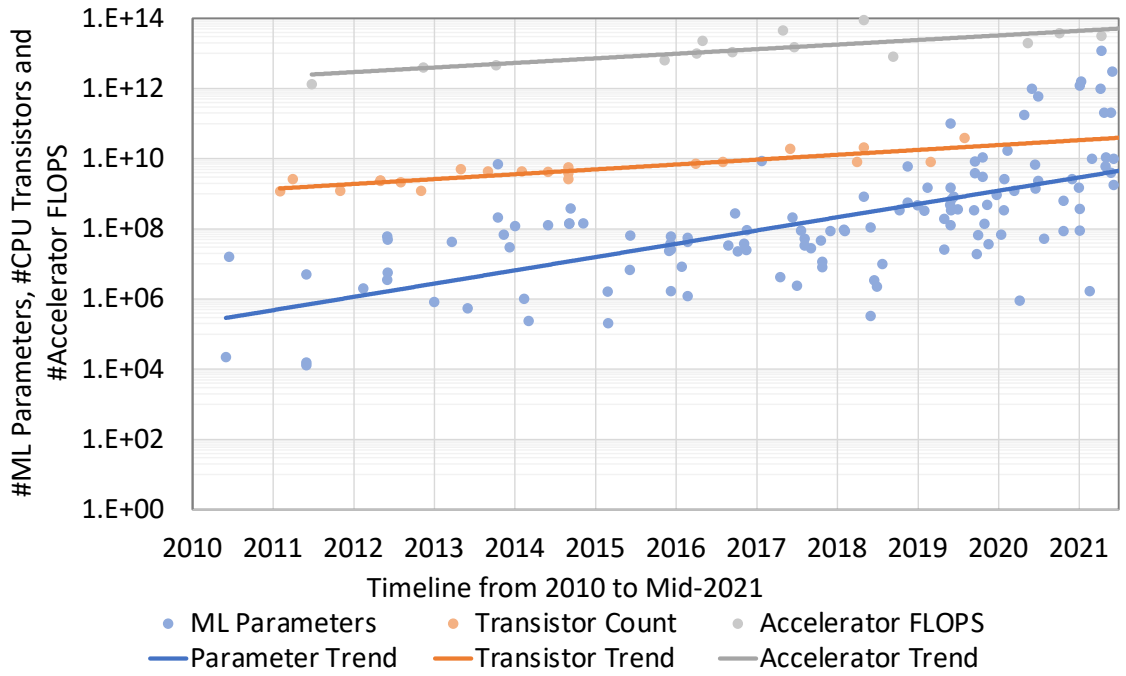

Figure 1-1: Visual timeline of the evolution of ML model size [1], CPU transistor count [2] and GPU/ML accelerator floating point operations per second (FLOPS) [3] in the last decade. Each trend is exponential, but the growth in ML parameter counts is more rapid.

take days and even weeks to complete [9, 10]. Future advancements to deep learning are significantly limited by the efficiency of large-scale distributed ML systems [15].

The primary challenge of DL training emerges from the tightly integrated nature of common aggregation algorithms, as individual worker nodes require their aggregate ML model updates to be supplied at precisely the right moment to make full use of the available accelerator resources [16]. A considerable body of work looks at how the training time of DL algorithms can be accelerated through scheduling, tensor operator choice, or new software frameworks [6, 9, 10, 11, 17, 18, 19, 20].

The scaling of DL training to increasingly large cluster sizes while maintaining linear performance gains has been a major focus of research and presents an ongoing challenge [6, 15, 17, 18, 21, 22]. The traditional method of DL scaling, increasing the per iteration throughput, has been found to have diminishing returns at large scale [11, 15]. This performance gap has led to the development of alternative scaling techniques that reduce iteration time, such as reducing the local throughput or splitting a model

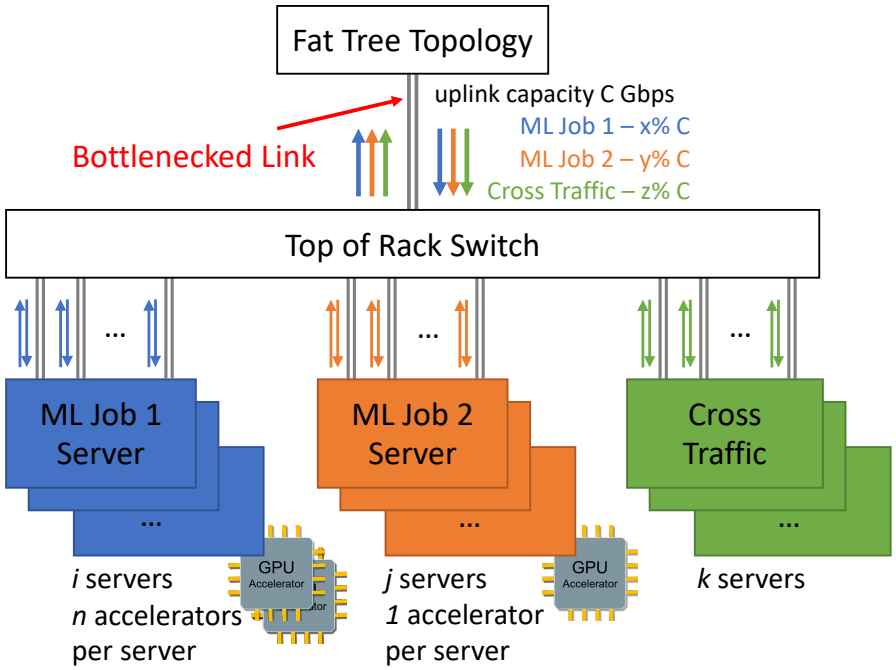

Figure 1-2: In a heterogeneous environment a range of ML jobs (blue, orange) and cross traffic (for example, MapReduce [4, 5], green) must share the bottleneck network resources to ensure both maximum utilization of local resources and the minimum possible job completion times. This poses the question: what is the most efficient allocation of C between these three arbitrary jobs sharing a common path out of a ToR?

between nodes - both of which require significantly higher bandwidths [15]. As both clusters and models continue to scale, the network performance of DL training is therefore a crucial bottleneck to improved overall performance.

The heterogeneous network conditions found in cloud environments represent a second challenge for DL acceleration, as the long-lived DL flows must coexist efficiently with competing traffic from a wide variety of applications, which predominately consist of small sized flows. This scenario is illustrated in Figure 1-2, in which I divide a group of servers under a typical data center top of rack (ToR) switch into three groups: two groups performing two distinct DL jobs, and the third group performing unrelated cross traffic jobs. As these servers all share the same ToR, they all utilize the same links between it and higher level switches within a standard topology, giving rise to a potential bottleneck.

From a user perspective, we wish our distributed framework to make the best use of the link capacity it is given, while from the perspective of the overall system, we want to obtain the most total work from all of the servers. Developing a characterization of how DL loads respond to different types of cross traffic is therefore a crucial step to understanding the trade-offs present in Figure 1-2, and therefore how to design (i) improvements to frameworks that are able to maximize throughput for a given bandwidth, and (ii) future congestion management systems that are DL aware and cater to its nuances.

My goal is to develop our understanding of the key network dynamics and parameters that impact distributed training, in order to provide a framework for answers to the key 'what-if' questions encountered when developing new efficient large-scale platforms. To do this, I study the impact of competing or 'cross' traffic on DL workloads. These observations can then be used in future work to help develop and characterize a DL aware congestion control protocol.

My key contributions form an evaluation of the performance of image recognition and natural language processing training in the presence of cross traffic, and can be summarized as:

- The impact of cross traffic on the utilization of accelerators;
- The effect of network bursts on ML training times and utilization;
- The consistency and variance of iteration times in the presence of competing network traffic;
- And the differences in intra-iteration communication demand observed from DL training task nodes.

In this thesis, I demonstrate that the variance and burst behavior of cross traffic plays a significant role in determining the efficiency of distributed ML training, in line with my theoretical predictions and the models I develop. At high cross traffic loads, I observe variations in the mean of 40% for bursty traffic and 8% for non-bursty traffic, as well as 99% tails on average 60% larger than the mean value of the iteration

time across all models tested. Further, I observe that when the Horovod platform competes with heavy cross traffic loads, the bottleneck link is underutilized by around 10%. I also establish that distributed learning frameworks do not consistently reduce their usage of non-bottlenecked links when they encounter cross traffic, and therefore they consume more bandwidth on these links than required to maintain their iteration time performance.

This thesis is organized as follows: First, I introduce key concepts and conduct a literature review in Chapter 2. Then, I describe the experimental activity and present the results and subsequent analysis in Chapter 3. Finally, I summarize key results in Chapter 4. I also include additional methodology notes in Appendix A, and supplementary results in Appendix B.

Chapter 2

Background and Related Work

In this chapter, I first contextualize ML techniques in Section 2.1.1, and then explore the four model families I have chosen to work with in Section 2.1.2. Next, I introduce the operation of Distributed ML (DL) training as an extension of ML training in Section 2.2, and cover the limitations imposed by the networking environment in Section 2.2.1. I discuss the available parameter synchronization strategies in Section 2.2.2 and parallelization strategies in Section 2.2.3, and survey state of the art developments in training frameworks in Section 2.3, before introducing the Horovod framework I utilize in this thesis in Section 2.3.1. I explore profiling tools that are available to assist in the characterization of DL in Section 2.4. Finally, I discuss how typical data center network workloads are structured in Section 2.5.

2.1 Machine Learning

2.1.1 Introduction to Key Techniques

A supervised ML model is a mapping from some raw input space to a set of labels or categories. A model is learned (trained) by repeatedly adjusting parameters so as to achieve a high accuracy on a dataset of examples whose correct labels are known. Once trained, the model can be used to predict the labels of unseen examples. As deep learning systems continue to advance the state of the art across many domains in-

cluding vision and language, the use of increasingly large models and datasets creates a growing computational burden [23, 24, 25, 26].

The primary algorithm used for much of ML training is Stochastic Gradient Descent (SGD), which calculates from a random sample of examples (referred to as a batch) the local prediction from the model, and its loss, or accuracy relative to the known data, which is used to determine the ideal direction to move down the optimization surface. These steps make up the forward pass of the algorithm. The backward pass calculates the gradients, and then updates the model with the new gradients [23, 24]. SGD can be applied in a distributed fashion, where, after the backward pass has calculated the local gradients, they are aggregated across all worker nodes before the global gradient is broadcast to all models [16, 27]. The hyperparameters that define the algorithm or its variants must be carefully designed for the specific use cases to achieve the best time-to-accuracy.

SGD itself supports both synchronous and asynchronous execution when running in a distributed fashion. SGD is a robust algorithm, however, it has been empirically observed that excessive batch size (taking overly many examples to average before recalculating the model) leads to degradation in the performance of models at inference, this is especially a problem in large-scale distributed training scenarios where the use of large batch sizes is a key acceleration strategy [11].

2.1.2 Popular DNN Models

In order to test and characterize network response to training activity in different scenarios, I analyze a variety of common models. I utilize four types of models: (i) light-weight communication with light-weight computation requirements, (ii) light-weight communication with heavy computation requirements, (iii) medium communication and computation requirements, and (iv) heavy-weight communication and computation requirements.

In this thesis I test with ResNet-50 [28], DenseNet-161 [29], and VGG-16 [30] models on the ImageNet [31] dataset, and the BERT-large [7] model on the imbd dataset, to ensure a selection of models with different characteristics and sizes are

Table 2.1: Summary of the key parameters of the ML models used in my distributed training experiments.

	ResNet [28]	DenseNet [29]	VGG [30]	BERT [7]
Model Version	50	161	16	large-uncased
# Parameters	25,557,032	28,681,000	138,357,444	366,428,986
Batch Size per GPU	64	64	64	8
Abstract Blocks	16 Bottlenecks	4 Dense Blocks	5 Groups	24 Transformers
Layer Size (squared)	64 to 1536	128 to 640	192 to 1536	1024
Input Size	3×224^2 pixels	3×224^2 pixels	3×224^2 pixels	512 tokens (max)
# Convolution Layers	53	184	13	0
# Linear Layers	1	1	3	146
# Embedding Layers	0	0	0	4
# Attention Heads	-	-	-	12

available for analysis. The key parameters of the models selected from each category are outlined in Table 2.1.

Residual Networks - ResNet

My selection for a light-weight communication light computation model is ResNet, a popular image recognition ML model. The ResNet series of models was developed in response to the ImageNet dataset challenge [28], and ResNet-50 is routinely used today for an initial trial scale for many experiments, as it has a reasonably complex structure without being prohibitively expensive to test. This model consists of a series of shortcut 'residual' paths that skip convolutional filters, which are followed by the standard pooling layers. This produces higher performance from fewer floating point operations [28]. ResNet is known to be relatively computationally intensive but not communication heavy [6, 10]. This model family, specifically ResNet-50, therefore serves as a good benchmark for a lightweight networking baseline.

Dense Networks - DenseNet, and the VGG Model

After ResNets, DenseNets are the next step up in terms of network depth, featuring more layers, more complexity, but a manageable number of parameters through feature reuse. Another key identifying feature of this type of network is that every layer is connected to every other layer [29]. These models therefore represent a midpoint between the smaller earlier models and the larger NLP style models in terms

of scale, and thus I selected DenseNet-161 to act as the light-weight communication model with heavy computation requirements. I also test with VGG-16 [30], a simple network predating ResNet, but one which has a significantly larger communication load relative to either ResNet or DenseNet, and so is well suited to act as the medium intensity model.

Bidirectional Encoder Representations from Transformers - BERT

I also wish to test with a type (iv) model, with heavy computation and communication requirements, to investigate the effects of scaling, and of the additional communication requirements larger models demand. A good candidate for this is the BERT family [7], and other transformer-based models such as GTP-2 and GTP-3, which have hundreds of millions to hundreds of billions of parameters [7, 8]. These models achieve good performance on natural language processing (NLP) applications, such as rapid automatic translation [32], and at time of writing are some of the largest common models. Due to the size of these NLP models, they cannot be implemented effectively without some form of distribution, due to the memory and computational requirements. I selected BERT-large, with 336m parameters, to use as a representative of this class of models, it is approximately 14× larger than the smaller ResNet-50 model, as outlined in Table 2.1, and therefore provides insight into network impacts for large, heavy communication models.

2.2 Distributed ML Training

A common approach to accelerate the training time of large ML models is distributed training, which uses a large number of parallel, independent calculations, performed in an interleaved fashion, with communication between processes required to ensure model convergence [22, 33, 34]. Multiple frameworks have been devised to enable increasingly efficient distributed training for a variety of scenarios [6, 15, 17, 18, 19, 20, 21, 35, 36, 37, 38, 39].

The key distribution block underlying every framework is the `all-reduce` algo-

rithm [9, 40], which is typically implemented as a ring [6], and is available as part of Nvidia's NCCL communication library [41] and MPI. It can be broken down into a series of both `reduce-scatter` and `all-gather` operations. `reduce-scatter` operations share the work of computing a subset of a DL gradient update across a subset of worker nodes. An individual node reduces (or aggregates) all gradient updates for a particular set of parameters. An `all-gather` operation then collects these aggregated updates from each node to create the final result [6, 42].

2.2.1 Networking Limitations

The network itself is still a bottleneck for large DNN training jobs [6, 10, 17, 18, 21, 22]. As an example, the Microsoft ZeRO (Zero Redundancy) optimizer [21] has scaling inefficiencies above hundred billion parameter models on a 400 GPU DGX-2H cluster [43]. ZeRO is able to achieve super-linear scaling in GPU numbers up to this model size by redesigning the memory management of DL training to reduce the memory footprint. However at the maximum model size tested of 170 billion parameters, the efficiency of the system degraded to only 53% of its sustained performance at lower sizes, demonstrating the scaling challenge still to be met [21]. This degradation occurs because while increasing the cluster size typically either reduces or maintains the computation required per worker, it does not significantly affect the bandwidth required for the computation phase, as it is dependent on model size and the parallelism employed [6, 9, 11]. Increasing the batch size of a data parallel training job can potentially compensate for the performance degradation experienced from scaling, but due to limits in model convergence with arbitrarily large batch sizes, there is an upper bound on reducing training time by increasing batch sizes [11, 44].

In very large-scale environments, the total training time becomes dominated by the communication time, which results in a diminishing return if more workers are added. As workers become more computationally efficient, this set of conditions becomes more common, which drives the motivation for designing efficient large-scale systems for training ML workloads at massive scale that remove the network as a potential bottleneck altogether.

2.2.2 Parameter Synchronization Strategies

Distributed ML training can be categorized into synchronous, asynchronous, or partially synchronous settings. These designs are built into various software frameworks, discussed in Section 2.3, that support distributed training. Synchronous designs maintain a strict lockstep across all workers per iteration, benefiting from guarantees on convergence [16, 22, 33, 45]. Asynchronous designs allow some workers to lag the leading iteration, allowing anything from a degree of staleness (or slack) in updates, but implementing limits to achieve partial synchronous behavior and guarantees [36, 46, 47, 48, 49], to operating entirely without synchronization steps to achieve faster run times or resource utilization at the expense of convergence quality [16, 50, 51]. Recent developments such as Hoplite improve the utility of asynchronous systems by redesigning the communications layer to provide efficient collective communication and fault tolerance [37]. SGD itself converges robustly, even in the presence of slack-asynchronous systems such as Hogwild! [46, 52]. Due to the presence of these complex timing requirements, and the non-uniform access times present in cloud settings [53], distribution often requires expert tuning to achieve the maximum performance of the hardware [54, 55]. Removing the reliance on expert tuning therefore requires a full and automated understanding of the nuances of timing requirements and scheduling decisions. I utilize synchronous designs in this thesis for simplicity to allow focus on network interactions in the worst case. This choice also removes some complexities expected from asynchronous systems due to changes in training accuracy convergence behavior.

2.2.3 Parallelization Strategies

We can also classify ML algorithms into three types by considering the parallelization used - distributed systems can use data parallelism, model parallelism, or a hybrid of the two.

Data Parallelism

The first and most common classification in use in DL frameworks is data parallelization, which repeats the model across each worker, and sends a different segment of data (a shard) to each worker to work on in parallel. This allows a larger quantity of computational resources to be leveraged to share the same problem, directly increasing the effective throughput. The individual worker's gradient updates are pooled and aggregated, before being fed back to the workers for the next step [19, 45]. This approach lifts the I/O limits on individual machines, and enables larger batch sizes to be used, provided SGD convergence limits are respected [11]. The networking load is proportional to the throughput of examples, the size of the model, and the size of the cluster.

Examples of data parallelization implemented as a framework include the default Pytorch and TensorFlow distribution options, the Litz framework, and state of the art distributed frameworks such as Horovod [6, 35, 45, 56, 57]. In this thesis, I focus on data parallelism, as it is the most common form of parallelism, and therefore provides a good base point for comparisons of the effects of cross traffic.

Model Parallelism

As an alternative strategy, frameworks can use model parallelism to manage large models that do not fit within individual accelerator memories [54]. Here, we divide a model's neurons between multiple accelerators, with the same work (minibatch) passed to each process. This reduces the memory required to process the DNN locally, as it is split between multiple processors, but requires a considerably larger communication bandwidth as any fully-connected layer requires all-to-all communication, and the connecting edges between accelerators must communicate with each other at low latency. Its use in convolutional layers is less efficient, and it suffers from efficiency bottlenecks if the model is spread to widely separated workers [19, 33].

Hybrid Parallelism

Frameworks using a pipe-lining strategy make an attempt to use the best features of both data and model parallelism in a hybrid form. We can break a model of interest into multi-layer slices, with dependencies that can be used to form a pipeline. Training is thus conducted over a minibatch in a pipe-lined fashion, increasing utilization and reducing waiting time for communications, which may be efficiently overlapped [10]. The Pipedream framework is one example of a hybrid parallel framework of this form, PipeSwitch as an alternative, which makes use of pipelining to enable spare inference cycles to be returned to training utilization without affecting the inference service level agreement latencies [58]

2.3 ML Training Frameworks

The community has been developing various ML training and scheduling platforms to support both fast iteration times and model convergence, often by overlapping the communication between nodes and computation on accelerators [59], efficiently using resources [6, 39], or improving the scheduling or optimization of the task graphs [19, 60, 61]. This section provides a short survey of today's ML training frameworks.

One of the commonly used distribution frameworks for ML training is a parameter server. It is the default option in many frameworks, including Tensorflow and Pytorch [45]. In a parameter server training, all nodes send their updates to a central node, which performs aggregation and returns the calculated gradient update for the back propagation step as a broadcast message [45, 62]. This system works well for small clusters, as it is simple to reason about and debug, and has significant simplicity advantages for checking timing behavior between workers as the controller has a global view. However, it is not the most efficient solution, as the server process is idle during computation, and the workers are idle during aggregation. At scale, these problems compound, and it is unable to keep up with the increased demand and remain efficient, necessitating other options [6].

Another approach is to accelerate the training time of multiple ML training jobs

through smart cluster scheduling schemes. An example of this approach is TicTac [59], a research platform that provides improvements to iteration times compared to standard out of the box implementations by generating a tighter overlap of communication and computation by operating on the graphs of popular ML frameworks like TensorFlow and Pytorch to discover optimizations. TicTac then schedules communications on a priority based on the level of dependency in the computation graph. A variant of this approach is to work directly on the scheduling problem, such as the recent Apathetic Future Share (AFS) algorithm [61], which provides elastic scheduling and maintains an awareness of future job demands compared to a greedy approach, which particularly penalizes ML, to achieve better job completion times overall. Pollux and ByteScheduler both aim for a similar goal, but achieve this by measuring the throughput and statistical efficacy together to balance cross traffic and dependency proxies, and through a Bayesian optimization approach respectively [57, 63]. Other schedulers make use of accelerator heterogeneity [64], sub-GPU scale allocation of resources [65], an auction mechanism to achieve a finish time fair distribution [66], or a placement based on trial execution and careful leverage of relaxations on consolidations [67] to deliver fairness and high resource utilization.

Another common approach is to distribute the computational graphs of DNN models across several accelerators. For example, FlexFlow [19] improves training by distributing DNN computation graphs across several dimensions, including Sample, Operator, Attributes, and Parameter. The FlexFlow training framework covers both data and model parallelism. FlexFlow's simulator searches for parallelization strategies using Markov chain Monte Carlo (MCMC) techniques to generate candidates, allowing an optima to be found, beating the performance of expert-designed strategies. Other related approaches seek reductions in the complexity of searching for equivalent operators to simplify graph execution [60, 68, 69].

A more recent development, the PET system [70], takes this further by searching for and applying partially equivalent tensor operators to achieve computational optimizations. Coupled with correction tensors that return the operation to the original statistical behavior, this leads to an overall improvement in the system capacity.

The Daydream platform [71] takes an alternate approach to FlexFlow's simulation, utilizing seen examples to predict and therefore accelerate future behavior as opposed to running an MCMC simulation. Simulation backed approaches are feasible as, due to the significant costs associated with running training, using a small amount of computational effort to improve the process before launch can lead to big savings overall [19, 71].

While simulation is often used to model scheduling performance, virtual machine resource utilization [72], or data center topology [73], for general tasks outside an ML context, these networking simulations do not map well to ML training scenarios, as they do not account for the specifics of ML operations [71]. For example, an ML sensitive network simulation could make use of the transmission patterns inherent in the NCCL libraries and algorithms chosen, such as ring all-reduce [42], as opposed to the random utilization model [74] common in data center used to construct simulators. Combining knowledge of network response to ML into existing ML acceleration frameworks would lead to better estimates in communication dominated configurations.

A growing number of research platforms approach the challenge of accelerating ML by focusing on the spare capacity in the network itself to support in-network aggregation using a variety of networking devices such as programmable switches for SwitchML or the ATP system [39, 75], or FPGAs as in the PANAMA system [35]. These approaches succeed in reducing the communication load associated with gradient updates by performing aggregation on partial gradients associated with specific ML jobs at line rate as they pass through networking devices. This reduces the number of upstream packets required to carry the aggregated information, reducing the impact of bottlenecks and simplifying the congestion problem. An alternate approach detailed in the PLink proposal [76], which makes use of network probes to establish locality information, then used to develop dynamic aggregation schemes to make use of the topology and react to changes.

In OmniReduce [38], the underlying all-reduce algorithm is modified to take into account the sparsity of the values presented for aggregation. The system only sends

non-zero tensor blocks on the network, drastically reducing the communication demand and improving communication time, especially for models with a high degree of sparsity.

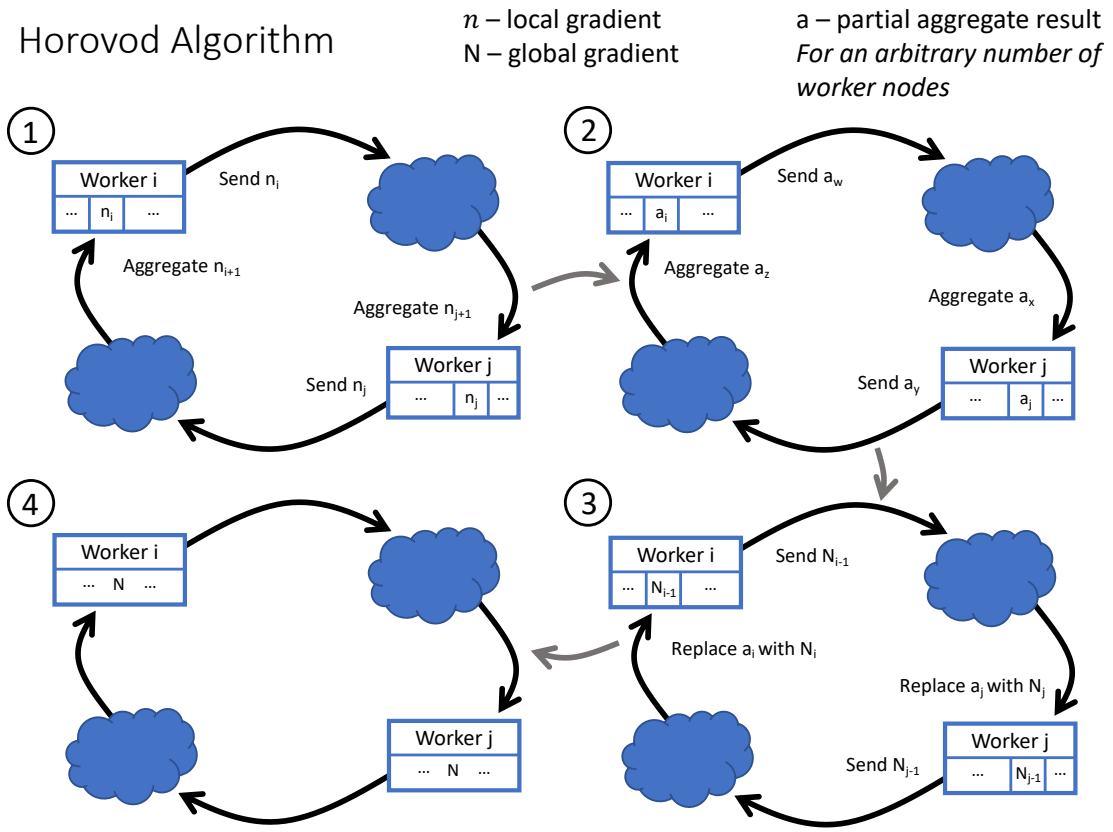

Figure 2-1: Horovod ring all-reduce algorithm, demonstrating the distribution of gradient updates across the worker nodes. In Step 1, the first partial gradient is sent to a neighboring worker. Step 2 represents a `reduce-scatter` step, where a partial aggregate is sent and added to the local corresponding value. Step 3 represents a `all-gather` step, where partial final gradient values are sent around the ring to build the full result at each worker as shown in Step 4 [6].

2.3.1 Horovod

The distribution system I focus on in this thesis is Horovod [6] from the Linux Foundation AI & Data Foundation, which is based on the logical structure of a ring and designed for production environments. This fully distributed architecture does not incur the bottleneck penalties of a parameter server when scaling, as no broadcast steps are required, and is bandwidth optimal in reducing the load placed on the net-

work by the all-reduce algorithm [42]. This algorithm makes a series of exchange and aggregate steps, where a portion of the gradient vector, or its intermediate aggregated values, are passed around in a ring-like fashion until all workers possess a fraction of the complete update, these are illustrated in Figure 2-1. These updates are then passed around to produce an updated model, replacing the unaggregated model parameters in sequence. A total of $2(X-1)$ steps are required for a process with X workers [6]. This approach is effective, but it is vulnerable to stragglers, as any single point of failure holds up the ring.

2.4 Profiling Tools

Profiling tools are a critical component of understanding the performance and bottlenecks of distributed ML training systems. Hardware profiling tools provide information on physical ML accelerators and the associated cores, memory, and storage, while software profiling tools provide top level overviews, graph execution, and overall performance. Hardware manufacturers, such as Nvidia, provide profilers that make use of counters embedded in the hardware drivers themselves to produce fine grained typical low-level information on the operation of their components. These tools typically contain hundreds of low level counters such as speeds, clock rates, and in/out flows [77]. However, these tools are limited as they do not leverage ML-specific knowledge, because they are designed for general purpose applications. An example of this type of proprietary tool is Nvidia's Nsight [77], which provides information on multi GPU synchronization behavior, communication and compute overlap, GPU utilization, and memory accesses, allowing for performance monitoring and tweaks. Other examples of these tools include nvprof [78], an earlier version of Nsight, and Intel's vTune Profiler [79], which serves a similar purpose around Intel CPU/GPU and FPGA IP blocks, allowing bottlenecks to be discovered and addressed within programs.

Hardware tools are not able to monitor network utilization directly. Network monitoring tools, such as the MLNX_OFED NIC driver package [80], are able to

monitor incoming/outgoing traffic, but these tools do not correlate network traffic to the accelerator processes. As a result, fine-grained analysis of network traffic during training is challenging.

In contrast to pure hardware profiling tools, the Google Cloud Platform provides the TPU tools [14] to allow detailed analysis of ML program execution on TPU accelerators. This allows association of the low level metric features of hardware profiling with the high level graph view of the DNN being trained, which is more common in purely software profiling tools. It also monitors the intra-TPU cluster network.

Software training frameworks such as MXNet and Pytorch have built in profiling tools [81, 82], which provide high level performance information and graphical analysis. However, as these frameworks do not have knowledge of the low level profiling details such as packet drops, cache hit rates, or memory speeds, and so they are less able to point at improvements from these hardware layers [71]. Daydream [71] proposes a system that is able to build acceleration efficacy estimates for different ML implementations by both utilizing both kernel-level dependency information from dedicated profilers and mapping low-level traces to specific DNN application layers. This provides an alternate pathway to answer interconnected 'what-if' questions that are the goal of this type of profiling work.

2.5 Data Center Workload

To be able to study the impact of network conditions on ML training, I must first identify the common types of flows in data center environments. Typically, data center workload analysis is performed on publicly released measurements provided by large online service providers, such as those supplied by Google in 2011 [83] and 2019 [74], Facebook in 2015 [84], or Microsoft in 2017 [72, 85, 86].

We look to these studies to understand how the system functions as a connected entity overall. Important points identified by Google in the initial analysis of the 2019 Google data, in comparison to the 2011 traces, include an increase in scheduling

costs, the dominance of 'hog' tasks, with 1% of tasks consuming 99% of the available resources, and an increase in average utilization of about 30% [74]. In all of these traces, a high level of heterogeneity in the workload is present, in terms of the lengths, sizes, and types of tasks run [83]. The performance of ML jobs in these large scale cloud clusters can be improved by scheduling improvements focusing on fair resource sharing, handling of long term tasks, and relative sub-task placement [66]. Network paths outside the core are typically underutilized on average, however, losses still occur due to burst traffic [87]. Therefore, bursts can be a significant factor in the performance of DL tasks running on a spatially concentrated group of nodes that do not need to cross the core switch fabric.

Well understood traffic models have been established based on these publicly released measurements, such as the pFabric model [88] which I utilize in this thesis. This models web-search traffic with a CDF providing an average flow size of 2.4 kB, which compares well with the known data center traffic observations of the majority of flows being short lived of size 10 kB or less [85].

In order to make use of this model in the test-bed I utilize for my experimental work, I co-developed a custom-designed multi-threaded TCP flow generation script in C++ with Sudarasanan Rajasekaran. This script acts in a similar way to the `iperf` tool, by sending TCP flows between a client and a series of servers, but in contrast to `iperf`, it uses a microsecond timing mechanism to launch flows on the client side in response to the specifications laid out in a configuration file; for flow start time, flow destination and flow size. The data sent is default dummy text, which is received and discarded at the server. Both client and server are implemented as multi-threaded programs to increase the capacity of the tool, allowing it to saturate a 25 Gbps link.

Configuration files are generated according to either the straight web-search traffic model [85, 88], or a modified version of the same that provides periodic bursts on demand, which for this thesis are defined as 1 ms 300% demand spikes every 10 ms. The details of the algorithms that generate both these traffic types are provided in Appendix A.1. Loads for consistent demand are generated using Algorithm 1, while burst loads are generated using Algorithm 2.

Chapter 3

Experimentation and Analysis

3.1 Configuration and Methodology

In this thesis, I use the Horovod framework [6] to train distributed DNN models developed using Pytorch [82] and Tensorflow [45] packages. The primary all-reduce technique in Horovod is ring all-reduce. Iteration times are recorded from Horovod's outputs. I use the `tcpdump` and `iperf` tools to capture network traffic and generate iperf cross traffic respectively. I also implemented a separate script to provide logging from Mellanox's Network Interface Card (NIC) packet counters. Finally, to generate web-search cross traffic, I used a custom-designed multi-threaded TCP flow generation script (described in Section 2.5), with loads generated as specified in the pFabric and DCTCP papers [85, 88].

Testbed Infrastructure

All experiments in this thesis are performed on a testbed with four ASUS ECS4000A-E10 servers. Each server is equipped with an AMD EPYC 7502P 32-core processor [89] and an Nvidia A100 accelerator with 40 GB of HBM2e memory [12]. Networking is provided by Mellanox ConnectX-5 100 Gbps NICs [80] on each server directly connected to a Juniper MX480 SDN Switch [90]. The switch is configured in 25 Gbps mode for all experiments in this thesis. As a result, each NIC is limited to 25 Gbps. I made this choice to explore the impact of high network congestion levels

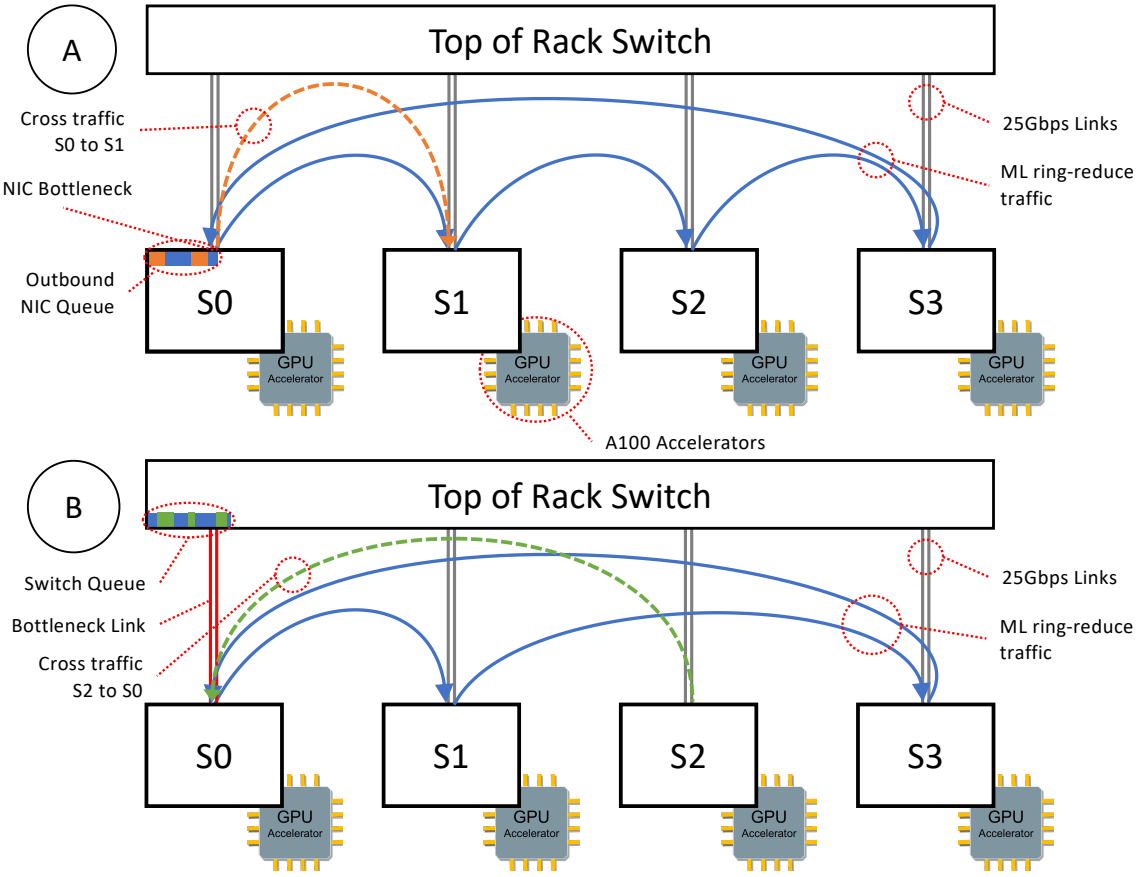

Figure 3-1: In configuration A, cross traffic is introduced into the outbound queue of the NIC on server S_0 and a Horovod ring all-reduce loop connects the 4 servers. This creates a bottleneck in the outbound queue of the S_0 NIC. In configuration B, cross traffic is generated at a separate server, in this example S_2, which feeds into the bottleneck link between the ToR and S_0 at the switch end. All links in this network have a maximum capacity of 25 Gbps.

between reasonably sized DL loads and cross traffic at a significant link capacity, without needing to be concerned about additional bottlenecks or confounding factors present at the full 100 Gbps system capability. I utilize Ubuntu 18.04, CUDA v11.1, cuDNN v8.0.5 and NCCL v2.7.8 [41], Horovod v0.21.3 [91], Pytorch v1.7.1 [92] and Tensorflow v2.4.1 [93].

Topology

I configure the topologies described in Figure 3-1 to study two alternate toy network scenarios for cross traffic competing with Distributed Machine Learning (DL) traffic, depending on whether server ingress or egress is the bottleneck. In configuration A,

cross traffic is introduced into the outbound queue of the NIC on server S_0. DL training is performed on the four servers, with its associated traffic, carried by the Horovod framework, passing in a ring all-reduce loop between these servers as shown in Figure 3-1(a). This creates a bottleneck in the outbound queue of the S_0 NIC.

In configuration B, cross traffic is generated at a separate server, in this example S_2, which feeds into the bottleneck link between the ToR and S_0 at the top of rack switch (ToR) end of the link. The other three servers perform DL training using the ring-all reduce communication pattern as shown in Figure 3-1(b). Configuration B also describes the scenario in which the bottlenecked link is between two switches at a higher aggregation level in the full topology (above the ToR shown in Figure 3-1), as the servers S_0 and S_2 must respond to congestion in the network without control of all outbound competing flows.

Models

I compare the iteration times of each model running on the infrastructure described in the previous section to understand the relative dominance of the computational and communication requirements of each model. The achieved iteration times and theoretical networking loads calculated from the ring all-reduce paper [42] are listed in Table 3.1, using equation 3.1, where P is the number of parameters of the given ML model, and N is the number of workers training the ML model. $N = 4$ for configuration A, while $N = 3$ for configuration B, as one node is set aside in this configuration to provide cross traffic.

$$D = 2 \times \frac{N-1}{N} \times P \times 32 \text{ bits/parameter} \tag{3.1}$$

Table 3.1 ranks the four models utilized in this thesis in order of decreasing required data per iteration. The BERT, VGG, and ResNet models, as listed in this table, follow a consistent trend where the required data per iteration decreases along with the iteration time observed. This is expected as the number of parameters is directly proportional to the data required, by equation 3.1, and the number of pa-

Table 3.1: Mean achieved iteration time of each model used in my thesis (carried by RDMA) with no cross traffic compared with the theoretical communication demand per iteration as calculated from equation 3.1.

Model	Data per Iter (Theory)	Mean Iter Time	BW Required
BERT-large	2011 MB	0.84 s	19.15 Gbps
VGG-16	830 MB	0.37 s	17.95 Gbps
DenseNet-161	172 MB	0.38 s	3.63 Gbps
ResNet-50	153 MB	0.16 s	7.69 Gbps

rameters is expected to be roughly proportional to the iteration time, provided the complexity of applying each parameter to incoming data is constant.

I divide the data required per iteration by the mean time per iteration to establish the estimated theoretical bandwidth required by each model, as listed in Table 3.1. This is a key difference between the different models as it determines the network capacity required to support their training. The largest model, BERT-large, requires 19.15 Gbps to train without experiencing congestion, and in general, this value decreases in line with model size. However, the DenseNet structure does not follow this pattern: DenseNet-161 contains a similar number of parameters to ResNet-50, as listed in Table 2.1, but takes 2.4× as long to complete an iteration (as shown in Table 3.1). This reduces the relative bandwidth required to train DenseNet without experiencing a bottleneck to only 3.63 Gbps compared to 7.69 Gbps for ResNet, as the computational complexity per parameter of this model (which I have estimated as the time taken per parameter) is higher, at $O(10^{-8})$ compared to $O(10^{-9})$ for the other models in this thesis.

3.2 Performance Against iperf Cross Traffic

I utilize the cluster in configuration A for this series of experiments. I generate cross traffic with the `iperf` tool to generate[1] a programmable bandwidth of long lived TCP

[1]The general form of iperf command used was `iperf -s` on the server side using the default ports, and `iperf -c` IP `-b` BW `-t` 180 `-i` 1 `-P` $pthr$ on the client size. Client parameters were the server's IP, a BW in Gbps appropriate for the test case, a 3 minute run-time in seconds, printing every second for logging, and $pthr$, typically ranged between 1 and 10 indicating the number of parallel threads sufficient to reach as close as possible to the requested BW load when competing with DL traffic.

flows between a single or several pairs of ports. This traffic then competes on the same link as the remote direct memory access (RDMA) Horovod traffic between neighbor GPUs. When the NIC bottleneck is highly utilized by both DL and iperf traffic, it is increasingly difficult to use `iperf` to force the NIC to allocate above approximately 81% or more of the total available bandwidth to itself, due to the practical limit on the number of flows and the behavior of the competing Horovod/NCCL traffic.

I begin each experiment in the series by verifying that `iperf` is able to achieve 25 Gbps on the link, before running the relevant DL job without competition, building the cross traffic, and increasing the number of parallel flows utilized by `iperf` until saturation is achieved. All training experiments are run for 180 s or until a minimum of 100 training iterations occur. The average number of iterations across all models is 1880.

Expected impacts of iperf traffic: Considering the iperf results as a whole, my hypothesis is that we expect to observe a minimal change in the iteration time from zero cross traffic up to the point at which there is insufficient bandwidth to transport the parameters of the given model in time for the next iteration to begin.

Figure 3-2 plots the mean recorded iteration times against the level of iperf cross traffic applied to the network. The 99^{th} percentile tail of each iteration time data point, calculated from all of the measured iteration times in that experiment, is shown for each series in Figure 3-2 as a vertical bar extending above the mean value. Figure 3-2(a) demonstrates the hypothesized relationship showing two distinct regions for all series. We observe that the expected minimal change in iteration time for low traffic levels is also true for the DenseNet and ResNet series. These series, which remain flat through the low demand region of Figure 3-2(a), are calculated to require 3.6 Gbps and 7.7 Gbps respectively in Table 3.1, and thus experience no degradation to performance up to their respective congestion points (defined later in this section) at 19.7 Gbps and 18.1 Gbps of cross traffic respectively. Once the cross traffic is higher than the respective congestion points, the remaining bandwidth can not satisfy the zero load demand, and we enter the high load region of the plot, where both models' iteration times increase sharply, as the accelerators must wait for the aggregation

stage to complete. The growth in iteration time is super exponential.

The larger models, with measured network demands of 18.2 Gbps for VGG-16 and 19.2 Gbps for BERT-large, show initial degradation in Figure 3-2(a), as the idle bandwidth is a minor component of the 25 Gbps link capacity for these series. The rate of the initial degradation in iteration time increases significantly as the bandwidth available to the model is more severely cut, leading to a similar non-linear (and therefore super-exponential) characteristic across both regions of the plot.

In Figure 3-2(b), I re-scale the cross traffic axis to understand the underlying mathematical function in operation. I expect an asymptote to occur in the iteration time data at a cross traffic level of 25 Gbps as this is the capacity of the bottleneck. Therefore, I apply a hyperbola transformation in Microsoft Excel, described in equation 3.2, with its asymptote set at 25 Gbps, to linearize the data points as they approach this value. I also normalize the values, so that a cross traffic level of zero (the initial no-load point) corresponds to a value of one.

$$BW_{rescale}(BW) = \frac{25}{25 - BW} \tag{3.2}$$

Figure 3-2(b) has two distinct sets of trends, corresponding to the small and large size model series shown in Figure 3-2(a). The small models remain flat before the congestion point, and now grow linearly past this. I observe linear growth for the larger models (BERT and VGG) across the entire cross traffic range, suggesting a linear fit, which is plotted in Figure 3-2(b) as well as in in Appendix B.2, would be appropriate for these models. The form of this fit, for coefficients $A = 0.618, 0.254$ and $B = 9.95, 10.75$ (for BERT, VGG respectively), is given in equation 3.3, and achieves an average $R^2 = 0.99$ for the two series in Figure 3-2(b).

$$t = Ae^{\frac{B}{25-BW}} \tag{3.3}$$

To analyze the performance of DL training against iperf traffic, I first extract the congestion point of each model under iperf cross traffic. I define the congestion point to be the point on a DL iteration time against cross traffic plot where congestion

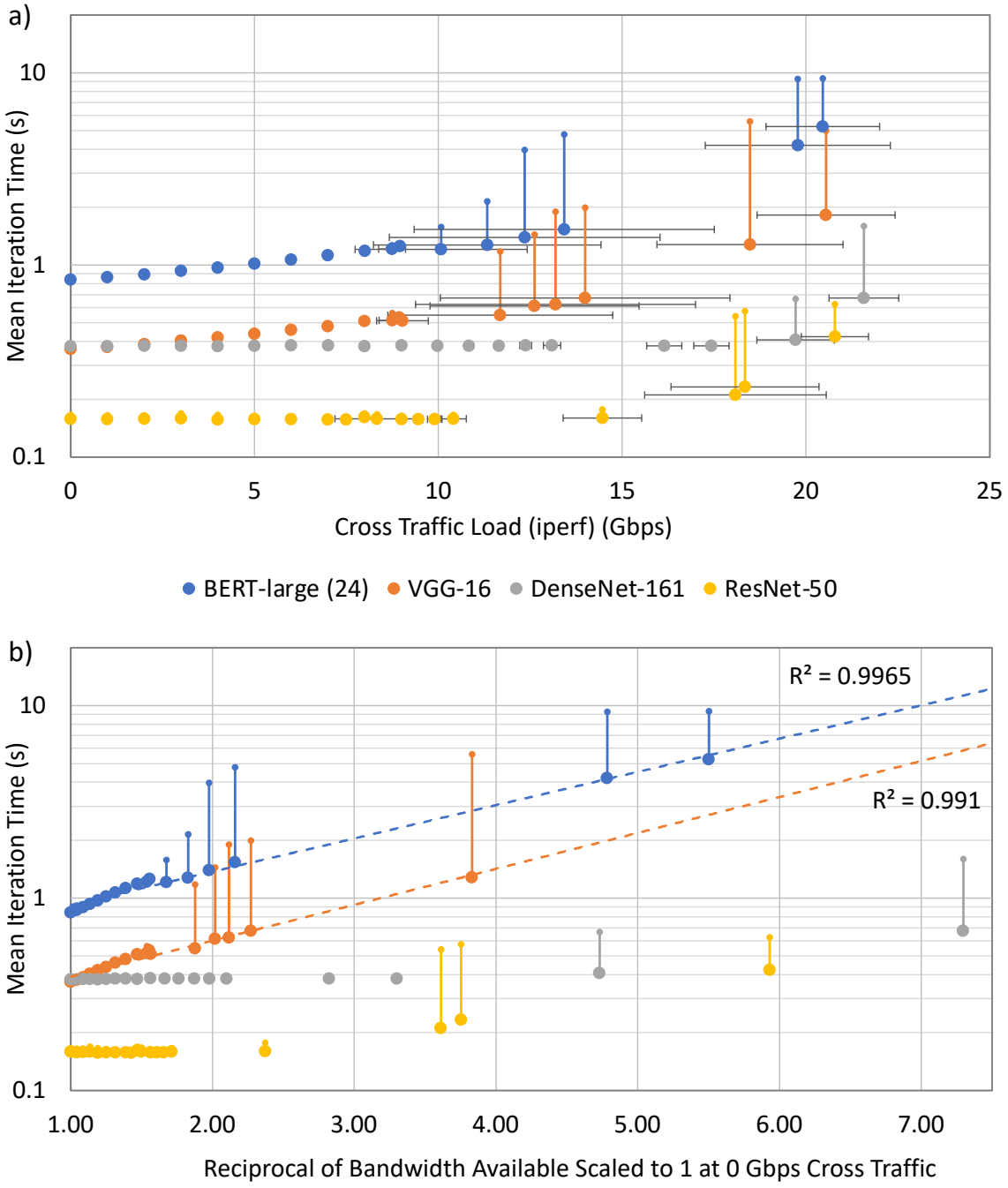

Figure 3-2: Mean behavior of DL tasks in the presence of iperf cross traffic plotted on a logarithmic scale. The maximum link capacity is 25 Gbps. The vertical bars on each data point represent the measured 99% tail iteration time of that experiment. Graph (a) plots the degradation of iteration performance against the quantity of iperf cross traffic, while graph (b) re-scales the x-axis using a hyperbola to produce a linear relationship between iteration time and the reduction in available DL bandwidth. Graph (b) also includes exponential lines of best fit and R^2 values for the BERT-large and VGG series.

Table 3.2: Comparison of the data required for a single DL iteration derived from theory and calculated from the congestion point observed in Figure 3-2 using iperf cross traffic.

Model	Theoretical Demand	Experimental Value of Congestion Point	% Difference
BERT-large	2011.0 MB	2216.2 MB	10%
VGG-16	830.1 MB	928.7 MB	12%
DenseNet-161	172.1 MB	268.8 MB	56%
ResNet-50	153.3 MB	182.5 MB	19%

first dominates the curve's behavior. Below this point, there is sufficient bandwidth to sustain both DL and cross traffic jobs without interference. Above this point, the two traffic types must compete, leading to congestion. I calculate the congestion point in two steps; first, by finding the intersection of two lines: one fitted to the sufficient (spare) bandwidth region, and one fitted to the first several points (tangent to the curve) of the congestion region; and second by then multiplying the available DL bandwidth by the iteration time at this point. The full procedure is outlined in Appendix A.2. For example, the congestion point for VGG-16 occurs at $(13.99, 0.68)$ in Figure 3-2(a).

I then compare the theoretical demand for each model, calculated as per equation 3.1, with this experimental congestion point in Table 3.2. I can now analyze systematic differences between the theoretical and empirical values for each loading scenario. I observe that the calculated congestion point is larger than the theoretical point for every model. This systematic difference occurs as the calculated congestion point assumes all bandwidth unused by the cross traffic is carrying productive DL traffic, which is not necessarily the case, and will lead to an overestimate of the congestion point's data value. The difference between the theoretical and congestion point estimate is small, around 10% for BERT-large and VGG-16, but larger for the smaller models, especially for the computationally heavy model DenseNet, with a 56% difference. This difference is likely due to the computational performance of the model still making a notable contribution to the iteration time despite the presence of congestion during model communication.

Table 3.3: Comparison of the theoretical and experimental bandwidths required by DL training in the most NIC bottlenecked test case from Figure 3-2 using iperf cross traffic.

Model	Theoretical	Experimental	Difference
BERT-large	3.06 Gbps	4.54 Gbps	1.48 Gbps
VGG-16	5.19 Gbps	6.52 Gbps	1.33 Gbps
DenseNet-161	2.04 Gbps	3.42 Gbps	1.38 Gbps
ResNet-50	2.90 Gbps	4.21 Gbps	1.31 Gbps

I extrapolate the ideal (theoretical) bandwidth required to serve the DL training from the most NIC bottlenecked sample point of each model series. The ideal bandwidth can be calculated by dividing the theoretical data transfer per iteration by the experimentally observed iteration time, for example, BERT-large's theoretical consumption is calculated in equation 3.4:

$$BW(\text{BERT Exp.}) = 2011 \text{ MB} \div 5.257 \text{ s} = 3.06 \text{ Gbps} \tag{3.4}$$

I compare this ideal figure with the experimental bandwidth used in Table 3.3 for an alternative insight into the performance of DL when competing with iperf cross traffic. This provides a quantitative sense of how much of the bandwidth available to the DL training is being used efficiently, for example, BERT requires 3.06 Gbps as calculated in equation 3.4, but I observe it utilizes 4.52 Gbps, leaving a difference of 1.48 Gbps available for improvements.

Takeaways: The key takeaways from my analysis of iperf traffic are that: (i) the congestion points of a series of DL models tested against increasing levels of cross traffic were well predicted by the theoretical demand calculated by equation 3.1. (ii) As expected, cross traffic negatively impacts the iteration time of distributed DL training when the degree of congestion exceeds the required bandwidth for training. For example, when the cross traffic load is 20.5 Gbps, BERT's iteration time is increased by 5.3× compared to the zero cross traffic case.

3.3 Performance Against Web-Search Cross Traffic

I utilize the cluster in configuration B for this series of experiments. I generate cross traffic between two servers as shown in Figure 3-1(b), with the TCP flow generation script described in Section 2.5. I apply a web-search profile load, either with a consistent demand level, or with millisecond long bursts in the form of a 1 ms 300% demand spike every 10 ms. The traffic competes with the incoming RDMA or TCP Horovod traffic on the incoming link. I find that in most cases, high cross traffic levels are easier to achieve than with the `iperf` tool. Each model's low cross traffic cases are compared against those obtained from the low `iperf` cross traffic results, as at low link utilization, there should be no congestion, allowing for verification of correct operation at this level.

3.3.1 Consistent Web-Search Cross Traffic

I begin each experiment in the series by verifying that the TCP flow generation script is able to achieve a minimum of at least 22 Gbps on the bottleneck link and that all links are in 25 Gbps mode using `iperf`. I then run the relevant DL jobs with stepped growth in cross traffic demand for a minimum of 120 s, or approximately 100 iterations, to minimize disruption due to socket errors, and potential memory and handling issues with large quantities of parallel communications from the TCP flow generation script at heavy loads. The average number of iterations across all models is 880.

Expected impacts of consistent traffic: As in the iperf cross traffic scenario, my hypothesis is that we should expect to observe a minimal change in the iteration time from zero cross traffic, up to a point at which there is insufficient bandwidth to transport the parameters required, with an additional note: I expect the use of consistent web-search flows as the source of cross traffic to lead to a greater degree of interference between the DL task and the cross traffic, and thus worse iteration times. This expectation is drawn from the behavior of congestion control; if all flows are relatively large and long lived, as in the iperf traffic case, this is an easy problem

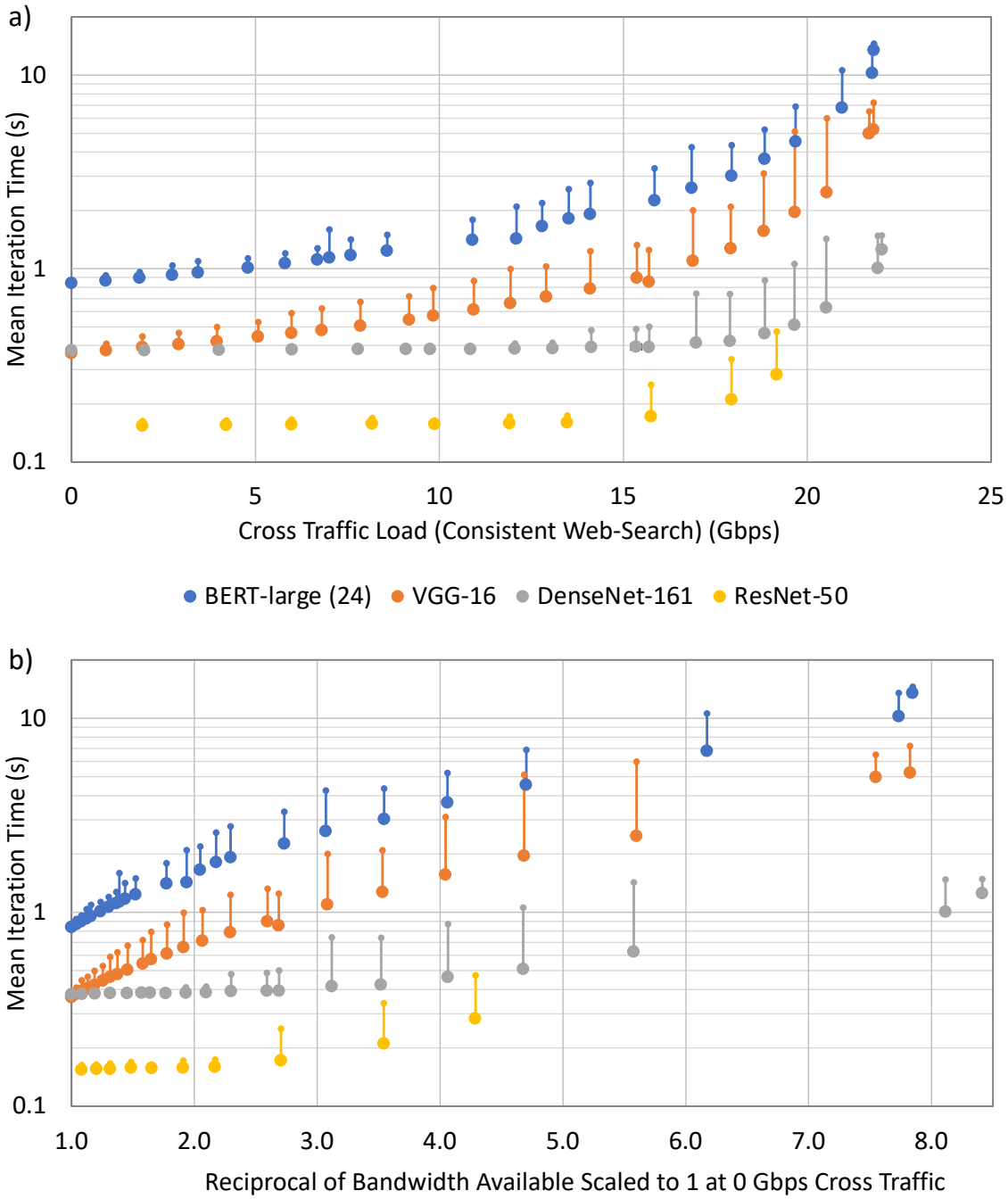

Figure 3-3: Mean behavior of DL tasks in the presence of consistent web-search cross traffic. The vertical bars on each data point represent the measured 99% tail iteration time of that experiment. Graph (a) plots the degradation of iteration performance against the quantity of consistent web-search cross traffic, while graph (b) re-scales the x-axis using a hyperbola to produce a more linear relationship between degradation and the reduction in available DL bandwidth.

to solve. However, if the flows are numerous, small, and unpredictable, like the consistent web-search traffic profile, the algorithm must attempt to control flows that may have already ended, making a fair share and full utilization harder to achieve.

Figure 3-3 plots the mean recorded iteration times against the level of consistent web-search cross traffic applied to the network. The 99^{th} percentile tail of each iteration time data point, calculated from all of the measured iteration times in that experiment, is shown for each series in Figure 3-3 as a vertical bar extending above the mean value. Figure 3-3(a) again demonstrates the hypothesized relationship for DenseNet and ResNet, with minimal changes in iteration time for the region up to their respective congestion points at 17.9 Gbps and 15.8 Gbps of cross traffic. The BERT and VGG model series in Figure 3-3(a) showed a more significant early increase in iteration time in this cross traffic scenario.

The increase in early degradation to iteration time is especially noticeable in the visibly higher 99% tail iteration times in Figure 3-3(a), shown by the vertical bars attached to each series' data points, even at low cross traffic levels. Tail iteration time performance is an important consideration for overall job completion time in synchronous DL, as the next step of the computation phase cannot proceed until all workers have completed their previous iteration. The rate of the iteration time increase in Figure 3-3(a) across all models increases with the level of cross traffic, as in the iperf traffic scenario, leading to a similar non-linear (and therefore super-exponential) characteristic function.

In Figure 3-3(b), I re-scale the cross traffic axis to understand the underlying mathematical function in operation. I apply a hyperbola transformation, described in equation 3.2, as before. Figure 3-3(b) has two distinct sets of trends, corresponding to the smaller two and larger two sized model series shown in Figure 3-3(a). The behavior of the small models in Figure 3-3(b) still falls into the flat and non-linear two region behavior observed in the iperf traffic results, but the large size models are now non-linear. A close visual inspection of Figure 3-3(b) suggests that the iteration time decay is slower than in the iperf cross traffic case. After the hyperbola transform has been applied, the BERT and VGG model series iteration times increase at a

Table 3.4: Comparison of the data required for a single DL iteration derived from theory and calculated from the observed congestion point in Figure 3-3 when competing with consistent web-search cross traffic.

Model	Theoretical Demand	Experimental Value of Congestion Point	% Difference
BERT-large	1787.6 MB	2296.4 MB	28%
VGG-16	737.9 MB	990.7 MB	34%
DenseNet-161	153.0 MB	355.2 MB	132%
ResNet-50	136.3 MB	198.8 MB	46%

decreasing rate along the re-scaled x-axis. This implies that the remaining function is highly likely to be of the form x^k where $0 < k < 1$. Testing the simplest choice, $k = 1/2$, gives a excellent fit, as shown in Figure B.2(b) in Appendix B.2, with an average $R^2 = 0.99$ across both series. The general form of the fit equation, with fit coefficients A and B (for example $A = 0.204,\ B = 1.44$ for BERT) is given in equation 3.5.

$$t = Ae^{\frac{B}{\sqrt{25-BW}}} \tag{3.5}$$

To analyze the performance of DL training against consistent web-search traffic, I first calculate the volume of data required by each model per iteration in configuration B from Figure 3-1, and find the congestion point of each model under consistent web-search cross traffic, for example, it occurs for VGG-16 at (15.70,0.85) in Figure 3-3(a). These values are presented in Table 3.4. As the observed volume of data required to be transmitted is larger than the theoretical ideal value, a greater share of bandwidth is required for a longer period than necessary to complete the transfer. This implies that the DL task is unable to utilize the network as efficiently as is theoretically possible. Further, as expected, I observe a larger gap between the theoretical demand and the experimental congestion point, at an average of 31% for BERT and VGG, suggesting the DL task is more highly impacted by the consistent cross traffic, compared to the iperf cross traffic. The difference for DenseNet is also significantly larger in this case, suggesting this model is unable to respond efficiently to update starvation, a logical consequence of computational intensity, compared to the other model types.

Table 3.5: Comparison of the theoretical and experimental bandwidths required by DL training in the most congested test case from Figure 3-3 using consistent web-search cross traffic.

Model	Theoretical	Experimental	Difference (%)
BERT-large	1.06 Gbps	3.18 Gbps	2.12 Gbps (67%)
VGG-16	1.13 Gbps	3.19 Gbps	2.06 Gbps (65%)
DenseNet-161	0.98 Gbps	2.97 Gbps	1.99 Gbps (67%)
ResNet-50	3.85 Gbps	5.83 Gbps	1.98 Gbps (34%)

I then compare the ideal and experimental bandwidths in Table 3.5. Compared to the iperf cross traffic case, both experimental and ideal values are lower, as higher cross traffic loads were reachable using consistent web-search traffic, leading to a lower available bandwidth at the most bottlenecked test case. However, the difference between the ideal and observed bandwidths was larger for each model than in the iperf traffic case, with, for example, BERT-large utilizing 3.18 Gbps when it theoretically required 1.06 Gbps, a difference of 2.12 Gbps (67%). This supports the analysis that the DL task is unable to make the most efficient possible use of the network resources it has available when competing against consistent web-search cross traffic.

Takeaways: The key takeaways from my analysis of consistent web-search traffic are that: (i) DL training is less efficient in its use of network bandwidth when competing with consistent web-search cross traffic in comparison to iperf cross traffic. (ii) The bandwidth difference between theoretical utilization and experimental observation for the most bottlenecked test case under consistent web-search cross traffic is larger than the equivalent case under iperf cross traffic, with an average difference of 2.03 Gbps (58%) between ideal and experimental results. (iii) Empirical functions for the iteration time under cross traffic load are able to predict the iteration time behavior of DL tasks.

3.3.2 Burst Web-Search Cross Traffic

As in the previous case, I verify that the TCP flow generation script is able to achieve a minimum of at least 22 Gbps on the link and that all links are in 25 Gbps mode using `iperf`. I generate burst loads using Algorithm 2, which allows the width, height,

and repeating pattern of bursts to be controlled. The mechanics of the algorithm are covered in detail in Appendix A.1. For the burst web-search traffic experiments, I select a width of 1 ms, a period of 10 ms, and a height of 3× the consistent mean level. This equates to, for a target average load of 6 Gbps as an example, a 1 ms period of 300% of nominal traffic (averaging 2.3 MB - 18 Gbps) followed by a 9 ms period of 78% of nominal traffic (averaging 5.3 MB - 4.7 Gbps). Figure B.3 presents another example of this traffic for a 9 Gbps average load. Traffic from the TCP flow generation script is graphed to verify it met the required network specifications. I select this scale of burst behavior to ensure a clear difference is evident between burst and consistent web-search traffic, and to be in a reasonable range to represent the considerable magnitude of burst flows in data center networks [87]. The burst segments of the traffic impede the DL tasks if they both occur at the same time, even if the overall average bandwidth is insufficient for this to happen otherwise. I set the burst period to be significantly shorter than the shortest iteration time recorded, here ResNet-50 at 0.16 s, to ensure that multiple bursts occur within an iteration and thus the predicted impedance occurs.

After validation, I sweep each DL job with stepped growth in burst cross traffic for a minimum of 180 s, with additional runs captured to ensure the minimum number of valid iterations is at least 50. There are two exceptions to this, the 25 Gbps runs of VGG and BERT, which contain fewer runs due to the length of the observed single iteration times. The average number of iterations captured across all models and runs is 1216, weighted towards low level cross traffic cases.

Expected impacts of burst traffic: As per both previous scenarios, my hypothesis is that we expect to observe a minimal change in the iteration time from zero cross traffic, up to a point at which there is insufficient bandwidth to transport the parameters required. The difference in this scenario is that I expect the congestion point to occur sooner in the computation light models than in previous cases, as their iteration times are more sensitive to the communication efficiency, as there is less computation available to mask delays due to events such as bursts on the network. Further, I expect that burst traffic should be more difficult to manage than consistent

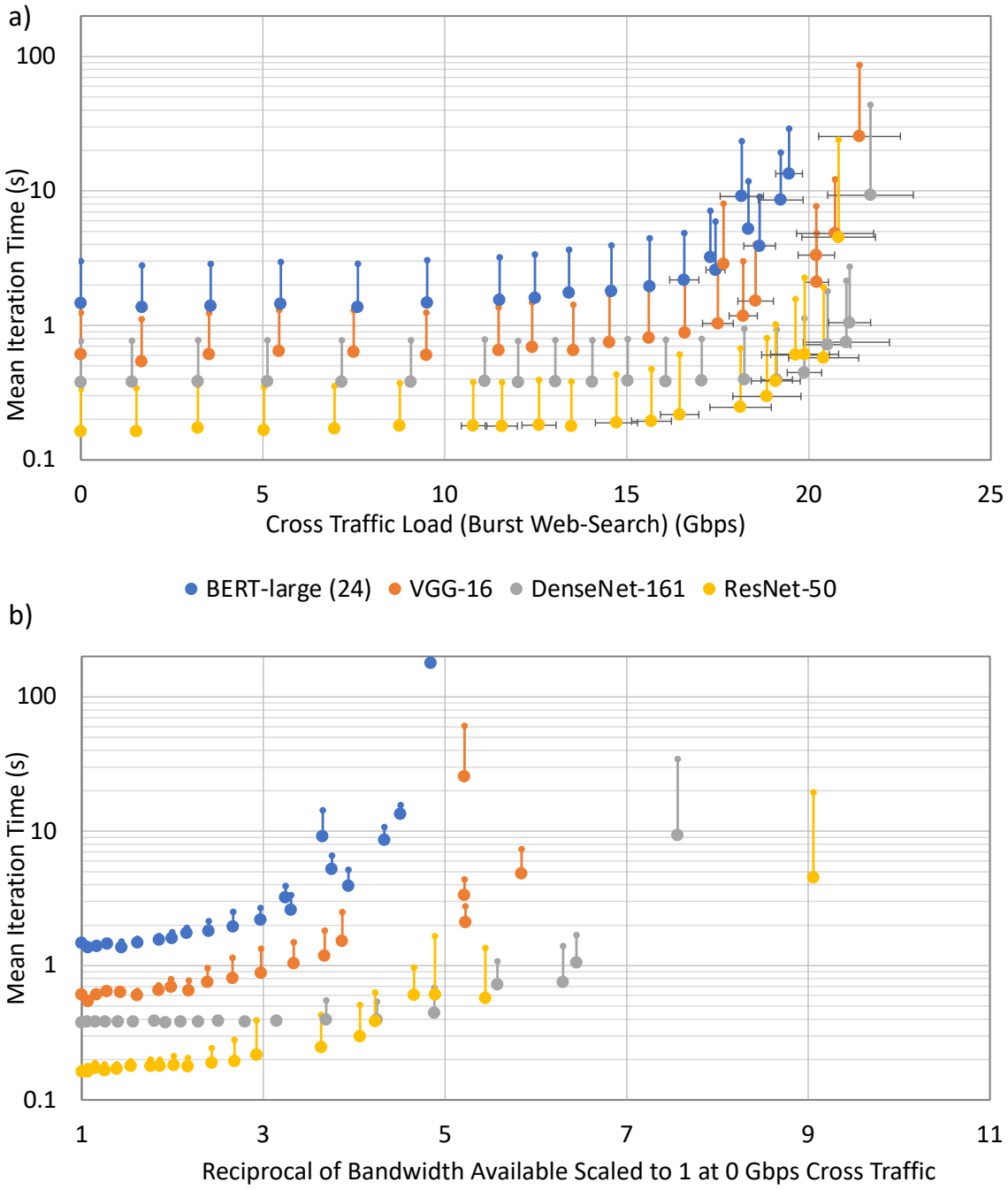

Figure 3-4: Mean behavior of DL tasks in the presence of burst dominated web-search cross traffic. The maximum link capacity is 25 Gbps. The vertical bars on each data point represent the measured 99% tail iteration time of that experiment. Graph (a) plots the degradation of iteration performance against the quantity of burst web-search cross traffic, while graph (b) re-scales the x-axis using a hyperbola to produce a more linear relationship between degradation and the reduction in available DL bandwidth.

web-search traffic for congestion control, as bursts are both large and short.

Figure 3-4 plots the mean recorded iteration times against the level of burst web-search cross traffic applied to the network. The 99^{th} percentile tail of each iteration time data point, calculated from all of the measured iteration times in that experiment, is shown for each series in Figure 3-4 as a vertical bar extending above the mean value. Figure 3-4(a) demonstrates the hypothesized relationship showing two distinct regions, separated by the congestion point, for all series. I observe the expected minimal change in iteration times for low traffic levels is true for large models as well as small models in this scenario as the DL traffic is carried over TCP rather than RDMA. This is in contrast to the previous scenarios, where a distinction could be made between the larger VGG and BERT models and the smaller DenseNet and ResNet models. This network difference is discussed in detail in Section 3.4.2.

As in the previous traffic scenarios, the DenseNet and ResNet model series in Figure 3-4(a) remained flat with minimal effect on iteration time up to the congestion point, with a super exponential increase in the second region. Across the three load types, I observe the distance between the congestion points of the smaller models, DenseNet-161 at 19.9 Gbps, and ResNet-50 at 18.4 Gbps, to be the lowest in this case. Therefore the computational difference between these models has the least effect on bandwidth requirements when cross traffic is periodically bursty.

The BERT and VGG model series in Figure 3-4(a) show a flat characteristic before they reach their congestion points at 14.6 Gbps and 16.6 Gbps respectively. At high cross traffic levels, significantly steeper curves are observed in Figure 3-4(a) for both series in comparison to the previous scenarios.

In Figure 3-4(b), I re-scale the cross traffic axis to understand the underlying mathematical function in operation. I apply a hyperbola transformation, described in equation 3.2, as in the previous scenarios. As expected from the consistent form of all series in Figure 3-4(a), the transformed series in Figure 3-4(b) all appear to have the same non-linear functional form. A close visual inspection of Figure 3-4(b) suggests that the iteration time decay is faster than in the iperf cross traffic case. After the hyperbola transform has been applied, the BERT and VGG model series

Table 3.6: Comparison of the data required for a single DL iteration derived from theory and calculated from the discontinuity observed in Figure 3-4 when competing with burst web-search cross traffic. The observed discontinuity is then compared to the equivalent consistent web-search cross traffic result.

Model	Theoretical Demand	Experimental Value of Congestion Point	% Difference
BERT-large	1787.6 MB	2343.5 MB	31%
VGG-16	737.9 MB	925.3 MB	25%
DenseNet-161	153.0 MB	284.9 MB	86%
ResNet-50	136.3 MB	203.2 MB	49%

iteration times increase at an increasing rate along the re-scaled x-axis. This implies that the remaining function is highly likely to be of the form x^k where $k < 1$. Testing the simplest choice, $k = 2$, gives a good fit, as shown in Figure B.2(c) in Appendix B.2, with an average $R^2 = 0.83$ across all series, and an $R^2 = 0.91$ for BERT-large. The fit is excellent for values below 17 Gbps, but is less suited above this level of cross traffic, as discussed further in Section 3.4.2. The general form of the square fit equation, with constants A and B, is given in equation 3.6.

$$t = Ae^{\frac{B}{(25-BW)^2}} \tag{3.6}$$

To analyze the performance of DL training against burst web-search traffic, I reuse the calculated the volume of data required by each model per iteration in configuration B, and find the congestion point of each model under burst web-search cross traffic, for example, it occurs for VGG-16 at (16.60,0.88) in Figure 3-4(a). These values are presented in Table 3.6. I observe similar congestion points in the burst case as in the consistent web-search case, with congestion point values within 100 MB of the consistent web-search scenario. As expected, the differences between theory and measured congestion points followed the trends established for the previous traffic scenarios, with a relatively modest 28% difference for the large models, similar to the consistent web-search results, and a much larger difference for the smaller models, especially for DenseNet-161, although it is smaller for this load compared to consistent web-search.

Table 3.7: Comparison of the theoretical and experimental bandwidths required by DL training in the most congested test case from Figure 3-4 using burst web-search cross traffic. Note BERT and ResNet were both tested up to 24.7 Gbps of cross traffic, while VGG and DenseNet were only tested up to approximately 21.5 Gbps.

Model	Theoretical	Experimental	Difference (%)
BERT-large	0.078 Gbps	0.347 Gbps	0.27 Gbps (78%)
VGG-16	0.232 Gbps	3.601 Gbps	3.38 Gbps (94%)
DenseNet-161	0.131 Gbps	3.306 Gbps	3.18 Gbps (98%)
ResNet-50	0.095 Gbps	0.347 Gbps	0.25 Gbps (73%)

I next compare the ideal and experimental bandwidths in Table 3.7. Compared to both previous cases, the experimental values are higher and ideal values are significantly lower, demonstrating a larger difference between average available and theoretically utilized bandwidth. This is expected as burst traffic, especially at high cross traffic volumes, should prevent DL training tasks from utilizing available bandwidth more effectively than other forms of cross traffic, as bursty traffic is by definition not at a consistent level. This result also demonstrates that in settings where burst traffic is dominant, there is significantly more room to improve utilization, with a range of 73 to 98% available headroom on a 25 Gbps link, even at very heavy cross traffic (24.7 Gbps) levels, compared to around 34-67% available headroom for consistent web-search traffic.

Takeaways: The key takeaways from my analysis of burst web-search traffic are that: (i) DL training competing with burst web-search cross traffic is least effective at utilizing the available network capacity, compared to the previous scenarios. This is especially notable, as these results are closest to real-world traffic. (ii) The bandwidth difference between theoretical utilization and experimental observation under heavy cross traffic loads is largest in this scenario, and represents an opportunity to increase the efficiency of DL tasks through networking improvements. (iii) As noted previously, empirical functions for the iteration time under cross traffic load are able to predict the iteration time the behavior of DL tasks.

Table 3.8: Mean under-performance by DL tasks (spare capacity) over bottleneck for the range of test cases where the link is saturated. Values are calculated as the difference (delta) between the bandwidth required to theoretically sustain the measured iteration time, and observed available bandwidth. Note: Very heavy cross traffic points in the BERT and ResNet series are excluded.

Model	iperf	Consistent Web-Search	Burst Web-Search
BERT-large	1.24 Gbps	2.63 Gbps	2.62 Gbps
VGG-16	1.12 Gbps	2.39 Gbps	2.51 Gbps
DenseNet-161	1.64 Gbps	2.84 Gbps	2.75 Gbps
ResNet-50	1.21 Gbps	2.25 Gbps	2.52 Gbps

3.4 Analysis of Cross Traffic Types

I now seek to compare the impact each cross traffic type has on the DL task it is in competition with. We know the amount of bandwidth available to the DL task for each experiment. We can calculate the ideal bandwidth required from the iteration time recorded for the same experiment, which we can then compare with the recorded available bandwidth to determine the size of the difference. Calculating this difference, or delta, for the test cases where the link is saturated is therefore a good criterion for comparison between the types of cross traffic, as it measures how far from the ideal full utilization of the link the DL training is operating (how much more bandwidth is being consumed to produce the same level of performance), and thus how well the congestion control algorithm is performing.

The bandwidth deltas for each model and cross traffic type are listed in Table 3.8. We observe a consistent step size of between 1 to 1.4 Gbps between the iperf and consistent web-search cross traffic types, and no significant change between the two types of web-search traffic. This implies that all models are uniformly affected by the more dynamic congestion environment provided by either variety of web-search cross traffic compared to iperf cross traffic, while the effects of burst traffic are not significant to the mean bandwidth requirements for the saturated region, quoted in Table 3.8, despite having a pronounced effect on the most congested test cases.

As seen in the previous sections, the most heavily bottlenecked cases have spare

capacities above, in percentage terms, those in Table 3.8. Working from the most bottlenecked examples, the spare capacity on offer, if able to be utilized, would provide gains of between 34-67% per model series for consistent web-search, and up to 73-98% per model series for burst web-search cross traffic, demonstrating the effects this traffic type has at the highest congestion levels. To give an example, BERT-large at its most congested (24.7 Gbps) against burst cross traffic could run approximately 5× faster (at 37 s compared with 184 s per iteration) if it utilizes all of the bandwidth theoretically available. At a more moderate point of 19.8 Gbps of cross traffic, BERT-large could achieve a speed-up of approximately 2× times (from 5.2 s to 2.6 s per iteration) in the same manner.

Takeaway: The key takeaway from this analysis is that the bandwidth differences listed in Table 3.8 represent the space for potential improvement in congestion control algorithms to better utilize the network for these traffic types.

3.4.1 Variance of Cross Traffic Effects by Loading

While I have so far looked at the behavior of the mean iteration times observed by DL training, this is potentially insufficient to develop a full picture of the behavior of a single data point, as the shape of the cumulative density function (CDF) must also be considered to understand the variance, dominance or lack of a tail, and any other features of note of the distribution. Therefore, I consider three representative models, ResNet-50, VGG-16, and BERT, as the most network dependent choices, in two network settings, with low levels of cross traffic (around 8 Gbps or 30%) and with high levels of cross traffic (around 20 Gbps or 80%). These are displayed in Figure 3-5(a) and (b) respectively. The values for 8 and 20 Gbps were drawn from the true measured bandwidths calculated from data from the `tcpdump` tool, rather than the requested input values. I made this choice as while the difference between these points is typically small, at high levels of cross traffic it can be on the order of 1-2 Gbps, and as the iteration times form a super exponential characteristic, selecting the wrong pair of series for comparison would produce misleading results.

Each model series is presented for consistent and burst web-search cross traffic, to

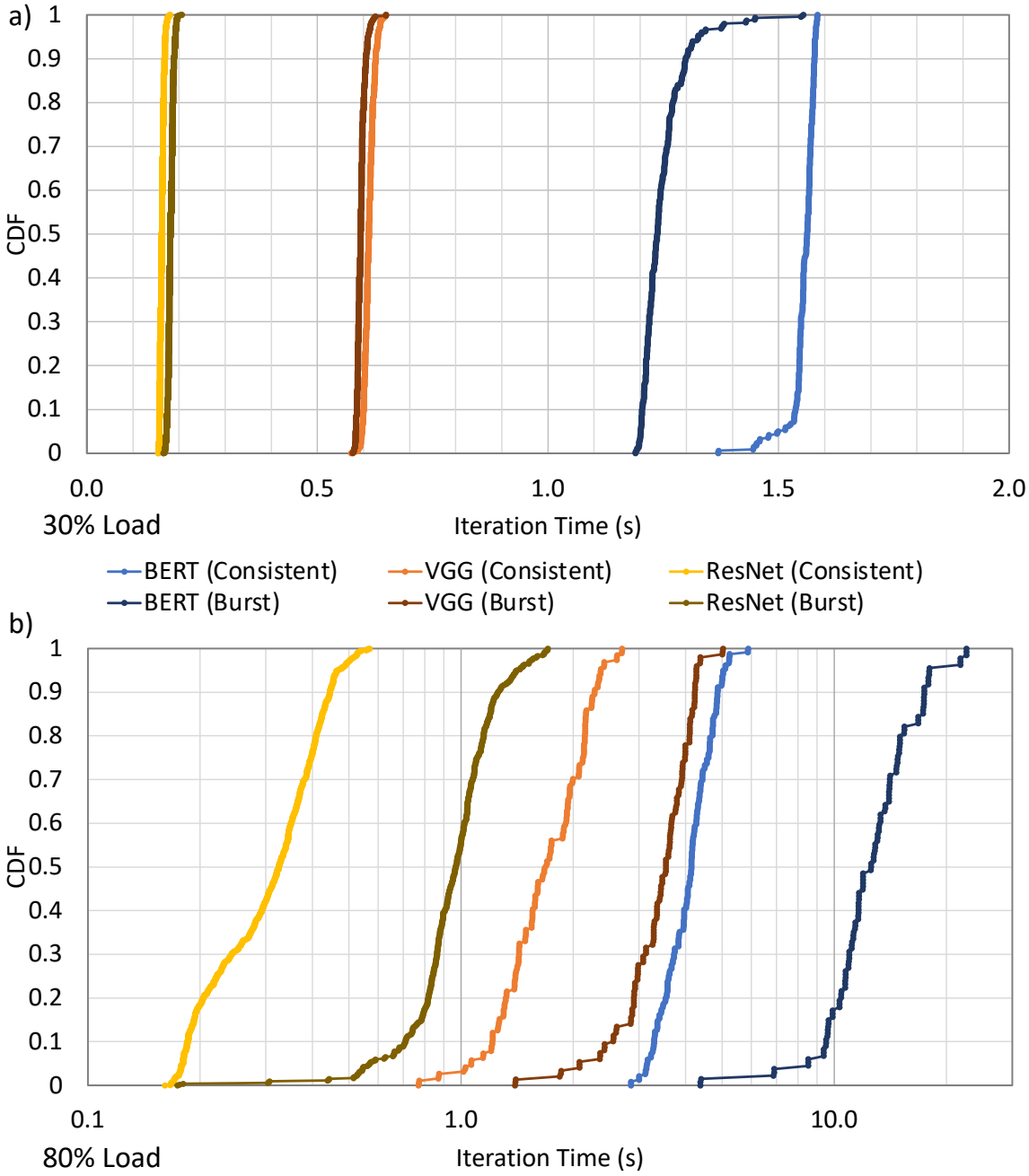

Figure 3-5: CDF of DL iteration times for three models, BERT, VGG and ResNet, comparing burst and consistent web-search cross traffic. Figure (a) shows a light (30%) cross traffic load, while (b) shows a heavy (80%) load. Note the time axis in graph (a) is considerably shorter than the logarithmic axis in graph (b).

visualize the effects of burst traffic on the distribution of training iteration completion times. All of the series in both graphs of Figure 3-5 use TCP for DL traffic, to remove this as a possible cause of difference between the series. I have included a CDF of DL performance for consistent web-search under Remote Direct Memory Access (RDMA) and TCP performance by way of comparison in Appendix B.1 as Figure B-1.

First I discuss Figure 3-5(a) in detail. At this comparatively low level of cross traffic, there is a negligible gap between the profiles of VGG-16 or ResNet-50 for burst and consistent traffic. Further, the distributions are almost vertical, indicating a very small variance and a minimal to non-existent tail. This is the expected result, as there is room on the link for both tasks to proceed without mutual interference, leading to consistent iteration times for the DL tasks under consideration. The BERT burst task is slightly faster than the consistent task, which is unexpected. When compared with the results obtained in Section 3.4.2 and Figure 3-6, both the burst and consistent series jitter throughout the low load regime, despite recording a minimum of 250 points per average in this range and having similar averages over all data points in this region. This instability may be due to the data handling and loading process required by the TCP communication pathway, as it is not visible in the RDMA sample in Figure 3-6, although the RDMA/TCP CDF, Figure B-1, does show a significant tail. This is discussed in depth in Section 3.4.2. The variance of the BERT curves is also significantly larger than the other two models considered.

Now, we consider the highly loaded case, Figure 3-5(b), in detail. Here, as expected, all of the series utilizing burst web-search cross traffic perform worse than the consistent web-search cross traffic series. Further, the burst loads also have significantly stronger tail behaviors than the consistent loads. All models, as one would expect from a more bottlenecked environment, experience a greater range of iteration times, and thus have a higher variance, and thus lower slope, than the low load case. Note that this plot uses a logarithmic scale to demonstrate the scale of difference, as the BERT burst web-search traffic series, in particular, is considerably above its consistent equivalent.

Takeaways: The key takeaways from this analysis are that: (i) At 30% load, there

is minimal difference in impact on a DL task's iteration time between consistent web-search and burst web-search cross traffic. (ii) At 80% load, burst web-search cross traffic has a considerably higher impact on the iteration time of DL tasks. (iii) the tails of the recorded iteration time distributions are considerably worse and less consistent under heavy cross traffic loads.

3.4.2 Effect of Cross Traffic Type on Model Performance

I now consider the impacts of each cross traffic type used in this thesis on ResNet-50, as an example of a small model, and BERT-large, as an example of a large model, in order to understand the differences in how the models respond to the different load types in a direct comparison. I present this comparison in Figure 3-6, with iperf (DL on RDMA), consistent web-search (DL on RDMA), consistent web-search (DL on TCP) and burst web-search (DL on TCP) as cross traffic series. Remote Direct Memory Access (RDMA) traffic refers to the protocol that Horovod uses to aggregate parameters during the communication phase of an iteration. As opposed to traditional TCP, RDMA uses the UDP transport layer, and allows for direct copying between system memory and the NIC buffers, bypassing the CPU. In the context of DL training, this represents the removal of a significant performance bottleneck, as aggregation results are not dependent on the CPU and may proceed directly to GPU memory [94, 95].

RDMA and TCP

Firstly, I consider the difference between RDMA and TCP DL traffic on iteration time when competing against consistent web-search cross traffic. In Figure 3-6(a), the BERT model is clearly able to iterate more quickly at low cross traffic levels when running in RDMA mode, as one would expect, as it is a large model dependent on significant data transport per iteration. The initial recorded means for RDMA and TCP DL are 0.84 s and 1.45 s respectively, supporting this. However, we can see that the RDMA case is immediately network limited, with the recorded speeds of

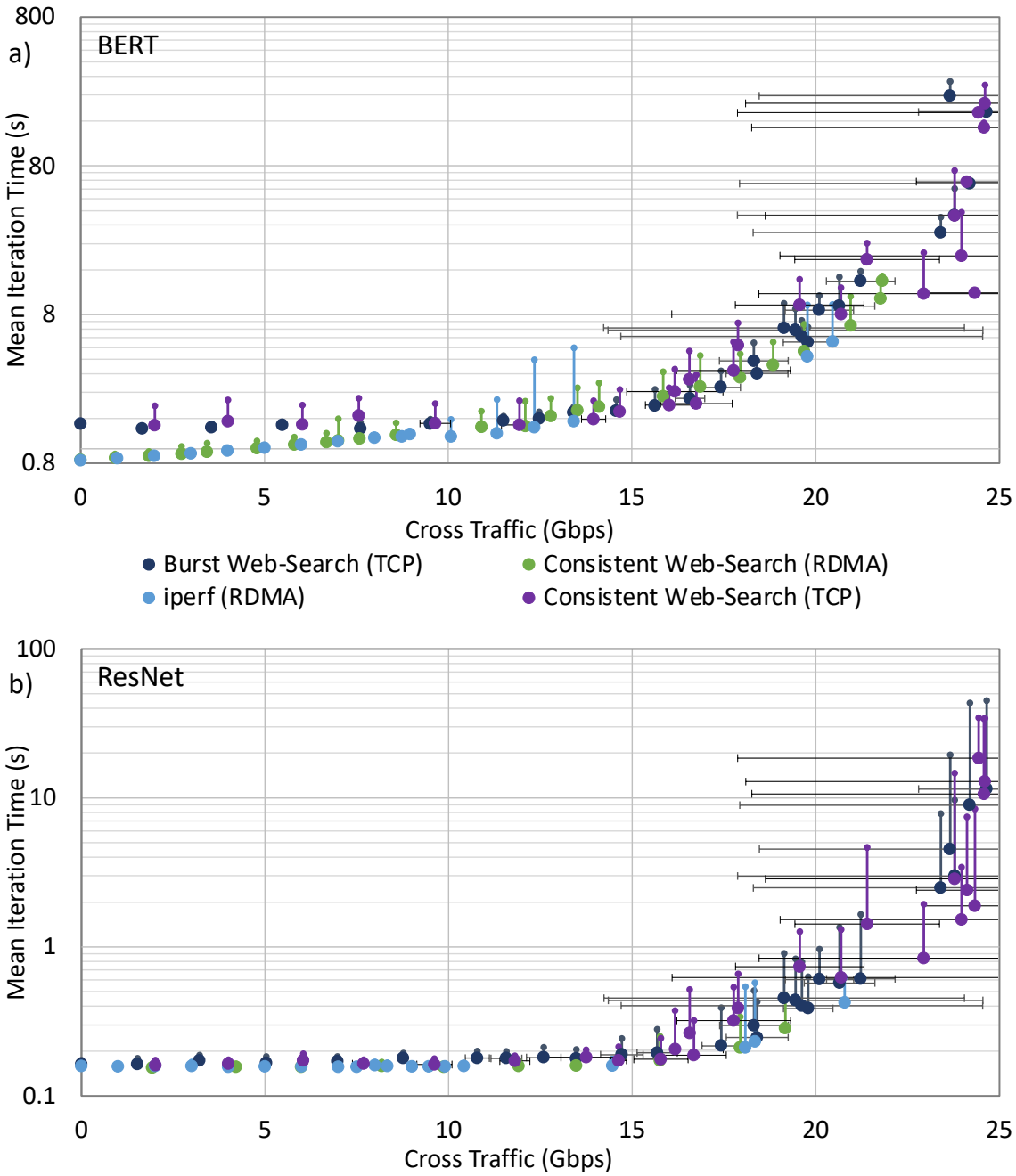

Figure 3-6: Mean iteration times of (a) BERT-large and (b) ResNet-50 DL tasks in the presence of iperf, consistent web-search (DL on RDMA), consistent web-search (DL on TCP) and burst web-search (DL on TCP) cross traffic. The maximum link capacity is 25 Gbps. The vertical bars on each data point represent the measured 99% tail iteration time of that experiment.

Table 3.9: Numeric fits for the functions modeling the degradation of BERT-large and ResNet-50 iteration times for DL tasks running RDMA and TCP communications in the presence of consistent web-search cross traffic. The corresponding graphs are presented in Figure B-3.

Model	RDMA	TCP
BERT-large	$t = 0.200 \exp\left(\frac{7.281}{\sqrt{25-BW}}\right)$	$t = 0.564 \exp\left(\frac{3.667}{\sqrt{25-BW}}\right)$
ResNet-50	$t = 0.140 \exp\left(\frac{22.793}{(25-BW)^2}\right)$	$t = 0.0645 \exp\left(\frac{3.311}{\sqrt{25-BW}}\right)$

iterations decaying from the first presence of cross traffic on the bottleneck link. In contrast, the TCP case does not respond to increasing network demand until after the RDMA case's iteration time advantage has been lost. This is because the CPU is performing processing in the TCP case, which delays traffic but also provides a buffer to absorb the initial network slowdown before it impacts performance. At medium and high levels of cross traffic, while the points of the two series are statistically distinct from each other, they trace almost identical super exponential curves, and thus have a similar effect on network behavior.

In Figure 3-6(b), the ResNet model, which is considerably smaller, has a flat characteristic under both RDMA and TCP DL until the congestion point is reached. This is expected as any gains to the system provided by RDMA would be significantly smaller, as the CPU load required by TCP would also be much smaller. Small degradation is not expected in this case as the base requirement for network bandwidth by the DL task is small compared to the total available bandwidth, so degradation is only expected (and observed) at high to very high cross traffic levels. At these high settings, the behavior of the RDMA and TCP-backed DL task again follow very similar curves across the range of analysis, as the network environment becomes the controlling factor. This behavior matches that observed by the Horovod development team comparing the performance of RDMA and TCP [91].

In order to quantify these observations, I perform numerical fits to the RDMA and TCP consistent web-search data-sets, using the models derived in Section 3.3.1 and Section 3.3.2. I fit the consistent web-search model to each curve, using least-squares error and the Microsoft Excel solver add-in [96], and list the result in Table 3.9. The

graphical fits are provided in Figure B-3 in Appendix B.2. The ResNet-50 RDMA series contains a short enough cross traffic range that the burst web-search model provides a better fit, so I utilize this model for this series. Overall, I achieve an average $R^2 = 0.97$ for the RDMA fits and an average $R^2 = 0.96$ for the TCP fits. The TCP fits have a lower coefficient of determination as they extend closer to the asymptote at the link capacity. This analysis supports the previous qualitative observations, showing small differences across the full experimental range.

Takeaways: The key takeaways from this analysis are that: (i) While individual points do not agree, small DL models are effectively indistinguishable regardless of load. (ii) At heavy loads, the key point of interest to my thesis, all DL models behave similarly with each type of protocol with this configuration. (iii) The numerical fits support these observations, confirming that there is little difference between DL models carried by RDMA and TCP from the point of view of their network responses.

iperf Traffic, Consistent and Burst Web-Search Traffic

Secondly, I consider the impacts of the different load types on the iteration performance. From the analysis on individual traffic types performed to this point, we would expect the order of traffic types, ranked by their efficacy in slowing DL tasks, to be: iperf, consistent web-search, burst web-search. The iperf traffic, denoted by the light blue line series in Figure 3-6, performs equivalently to slightly behind the purple consistent web-search series across the full range of bandwidth values. The lack of separation at low loads is not surprising, and is mirrored by the burst traffic results, as at low demand, bottlenecks do not occur. The DL task for iperf cross traffic, carried by RDMA, approximates the consistent cross traffic RDMA task for Figure 3-6(a) as expected. While the level of iperf traffic performance is worse than initially expected, by following the curve outlined by the existing points, we observe that at high loads iperf traffic allows the DL task to perform better than consistent web-search traffic. Using an alternative iperf traffic generator above the level of cross traffic achieved would show this separation more clearly.

In both graphs of Figure 3-6, the dark blue burst curve is somewhat above the

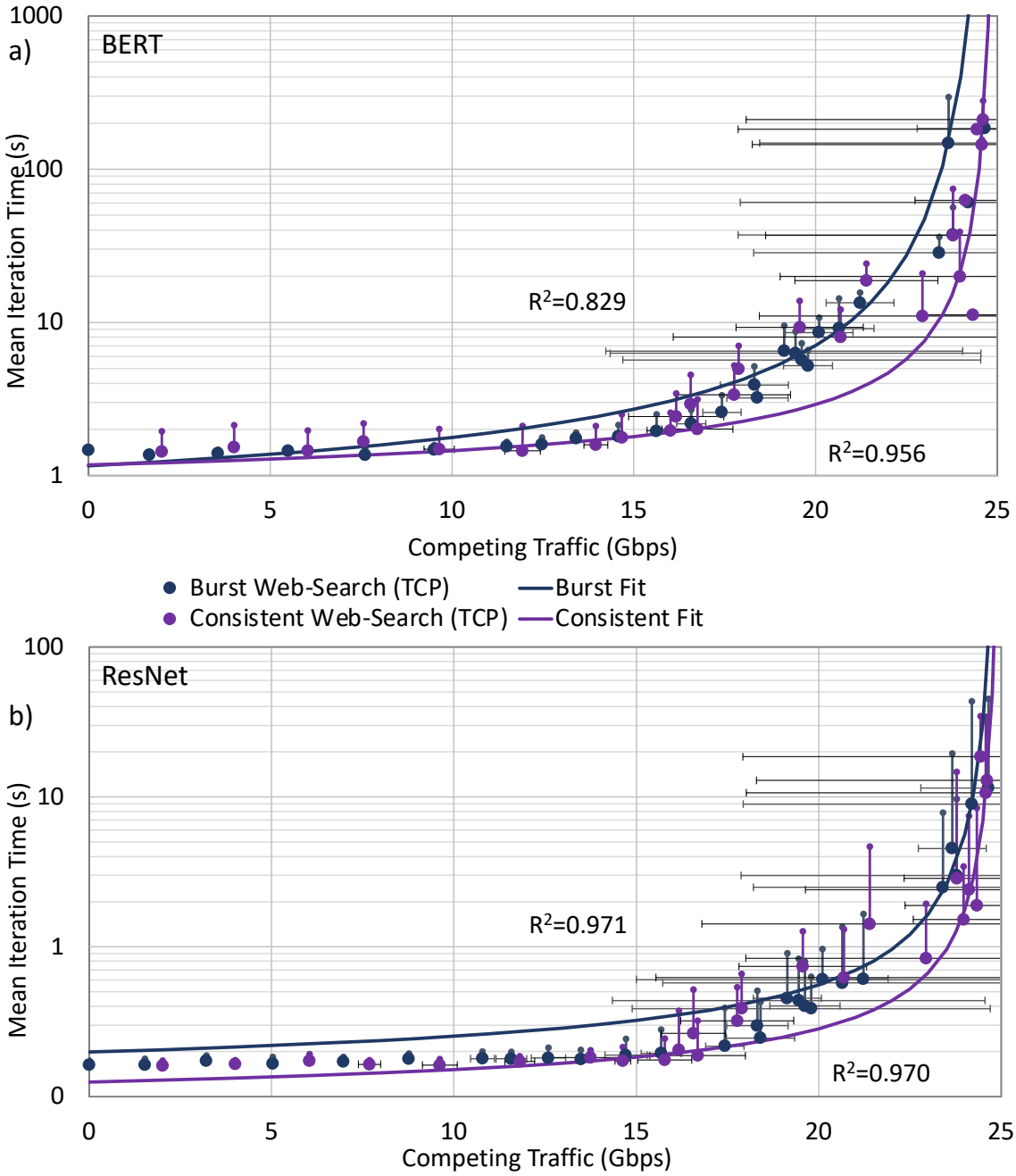

Figure 3-7: Mean iteration times of (a) BERT-large and (b) ResNet-50 DL tasks in the presence of consistent web-search and burst web-search cross traffic. The maximum link capacity is 25 Gbps. Fits based on equation 3.5 are included for all series in both graphs (a) and (b). Coefficients of determination are presented for each fit, and the vertical bars on each data point represent the measured 99% tail iteration time of that experiment.

Table 3.10: Numeric fits for the functions modeling the degradation of BERT-large and ResNet-50 iteration times in the presence of consistent and burst web-search cross traffic.

Model	Consistent Web-Search	Burst Web-Search
BERT-large	$t = 0.564 \exp\left(\frac{3.667}{\sqrt{25-BW}}\right)$	$t = 0.268 \exp\left(\frac{7.310}{\sqrt{25-BW}}\right)$
ResNet-50	$t = 0.0645 \exp\left(\frac{3.311}{\sqrt{25-BW}}\right)$	$t = 0.0862 \exp\left(\frac{4.174}{\sqrt{25-BW}}\right)$

other traffic classes, and therefore more effective at disrupting DL tasks. In Figure 3-7 I have performed a least-squares error fit of the empirical equation 3.5, derived for the consistent web-search traffic case in Section 3.3.1, to the curves for consistent and burst traffic. This produces the functions listed in Table 3.10. I made this choice as the burst traffic model derived in the previous section, while more accurate to around 17 Gbps, is unable to handle the discontinuity due to the maximum possible traffic load of 25 Gbps. The high level of uncertainty in the high bandwidth test cases limits the quality of the fits obtained, and also the level of precise analysis possible.

Takeaways: The key takeaways from this analysis are that: (i) The empirical models demonstrate that the iteration time performance is more severely degraded by the burst load than the consistent load. (ii) The general form of empirical model, equation 3.7, established by this work is broadly applicable to DL tasks competing with different cross traffic types at different load levels.

$$t(BW) = A \exp\left(\frac{B}{(C - BW)^n}\right) \tag{3.7}$$

where t is the iteration time, BW is the bandwidth, A, B are fit coefficients, n specifies the functional family and C is the bottleneck capacity.

Cross Traffic Variance

In order to understand the reasoning behind the large standard deviations observed in the measurements of bandwidths at high cross traffic levels in Figure 3-6, I consider the histogram presented in Figure 3-8. The two distributions, taken from burst and consistent web-search traffic at a 24 Gbps cross traffic level, have considerably different

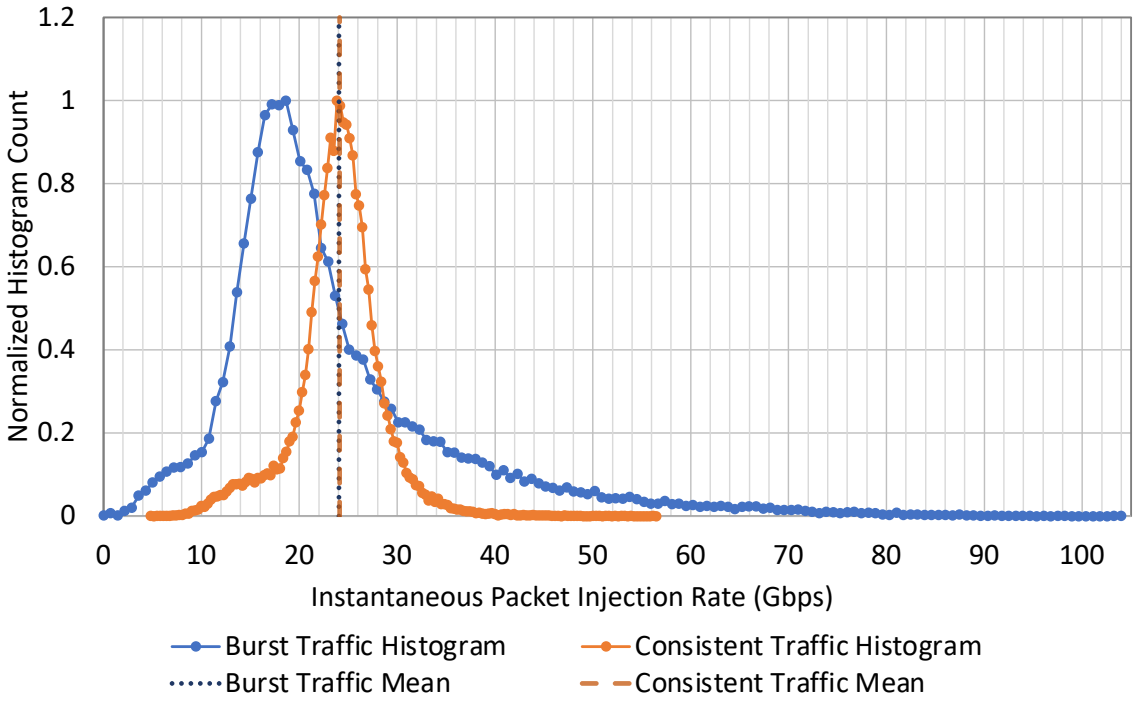

Figure 3-8: Histogram of 60,000 instantaneous measurements of the packet injection rate for burst web-search traffic and consistent web-search traffic, both for the requested load of 24 Gbps. A significant tail is present in the burst traffic histogram that is not present in the symmetric consistent traffic. The height of both distributions is normalized to the highest peak, and the mean of each distribution is shown with the dotted vertical line.

profiles, with the burst histogram exhibiting a lower peak, but a considerable tail and skew, as we would expect from a periodic bursty flow. In comparison, the consistent histogram is symmetric about its mean. Both histograms display a relatively high peak width, due to the stochastic sampling of the flow sizes and times leading to varying demand over time.

These relatively high variances increase the complexity of analysis, but the consistency of the means between traffic types means comparisons are still practical. The histogram widths are logical consequences of the way the selected load generation algorithm operates. Real-world traffic, which is not designed to have a specific mean, would likely have at least this width, if not larger, hence this behavior is not detrimental to building an understanding of the behavior of DL in response to cross traffic.

Takeaway: The key takeaways from this analysis are that: (i) precise values at high load points are difficult to achieve due to the width and skew of the sample peaks, limiting the maximum confidence in fits obtained in this region. (ii) The histogram behavior indicates that over the experimental times under consideration, the profiles of the two traffic types are distinctly different, stochastic, and adhere to the key characteristics expected of each traffic type.

3.5 Iteration Scale Network Interactions

To complete the picture developed in this thesis of network interaction with DL training, I consider the way a single iteration is affected by low (30%) and high (80%) cross traffic settings in detail. These results were captured from packet traffic on network configuration B in three key locations: a node only participating in DL (referred to as DL only), the node generating the cross traffic (referred to as cross traffic only), and the node receiving both the cross traffic and DL traffic (referred to as DL shared). I filter for outbound traffic with non-zero payload length to clarify flow directions and to prevent double counting. From this filtering, I separate three key flows, which are shown in Figure 3-9. Flow 1 captures the behavior of the DL ring when the link is not locally congested, but the ring is congested elsewhere. Flow 2 shows the effect of local congestion, as server S_0 is unable to advance its calculations until it has received all relevant updates through the congested inbound path. Flow 3 tracks the cross traffic incoming to server S_0. I made this choice to ensure any difference between DL load behaviors in the two tested load scenarios is visible. I utilize consistent web-search cross traffic to provide a reasonable level of traffic complexity without needing to account for the added complications of burst traffic in an iteration by iteration analysis. My hypothesis is that under heavy cross traffic loads, a large variation in profile should exist between flows 1 and 2, as flow 2 should experience a significantly different congestion environment, despite both carrying the same data per time.

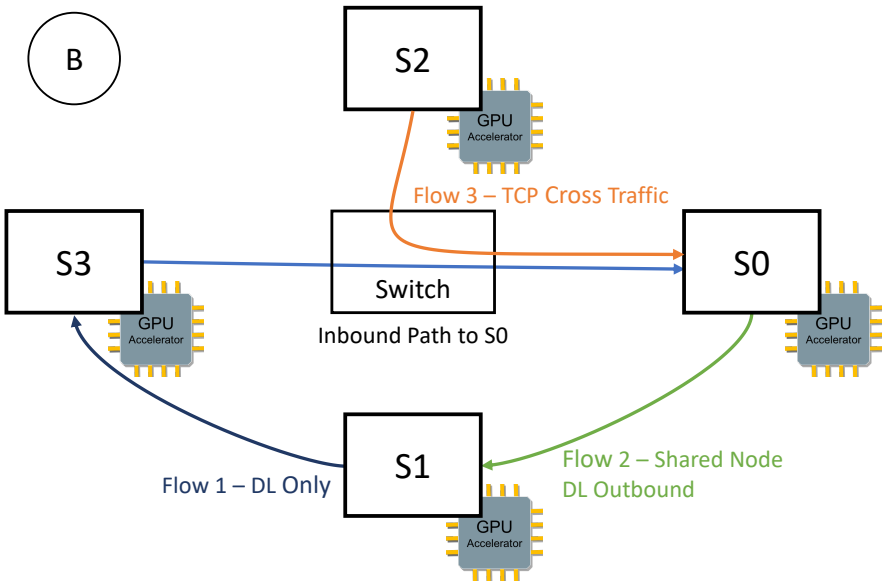

Figure 3-9: IP logical view of Configuration B from Figure 3-1. Flows 1, 2 and 3 are monitored using `tcpdump` and are reported in Figure 3-10 and Figure 3-11. The Horovod training ring consists of servers S_0, S_1 and S_3, with switch S_2 feeding cross traffic into S_0. Although all servers use the switch, only the inbound path to switch S_0, the location of the bottleneck link, is shown.

ResNet Analysis: Figure 3-10 demonstrates the behavior of ResNet-50 in each cross traffic setting. Figure 3-10(a) graphs two complete iterations, showing the peaks and troughs expected from an average 8 Gbps consistent web-search load, with the DL traffic point density adapting around the peaks. The DL traffic is sampled utilizing up to the full bandwidth available, with the density of samples spread across the full range of available bandwidths. I use short moving average filters to process the large quantity of raw data to provide better estimates of the instantaneous bandwidth without losing too much of the fine-grained time information represented by the individual samples. This does mean some estimated bandwidth points occur above the 25 Gbps link capacity, which is not physically possible. Uncertainty in the inter-arrival reporting time of packets over short time frames leads to this overestimate, which can be reduced with longer windows at the expense of event-time accuracy.

Visually, both the DL unique and DL shared nodes have their samples spread across the full range of bandwidths at approximately the same densities. This is in contrast to Figure 3-10(b), in which the DL only node is skewed higher than the DL

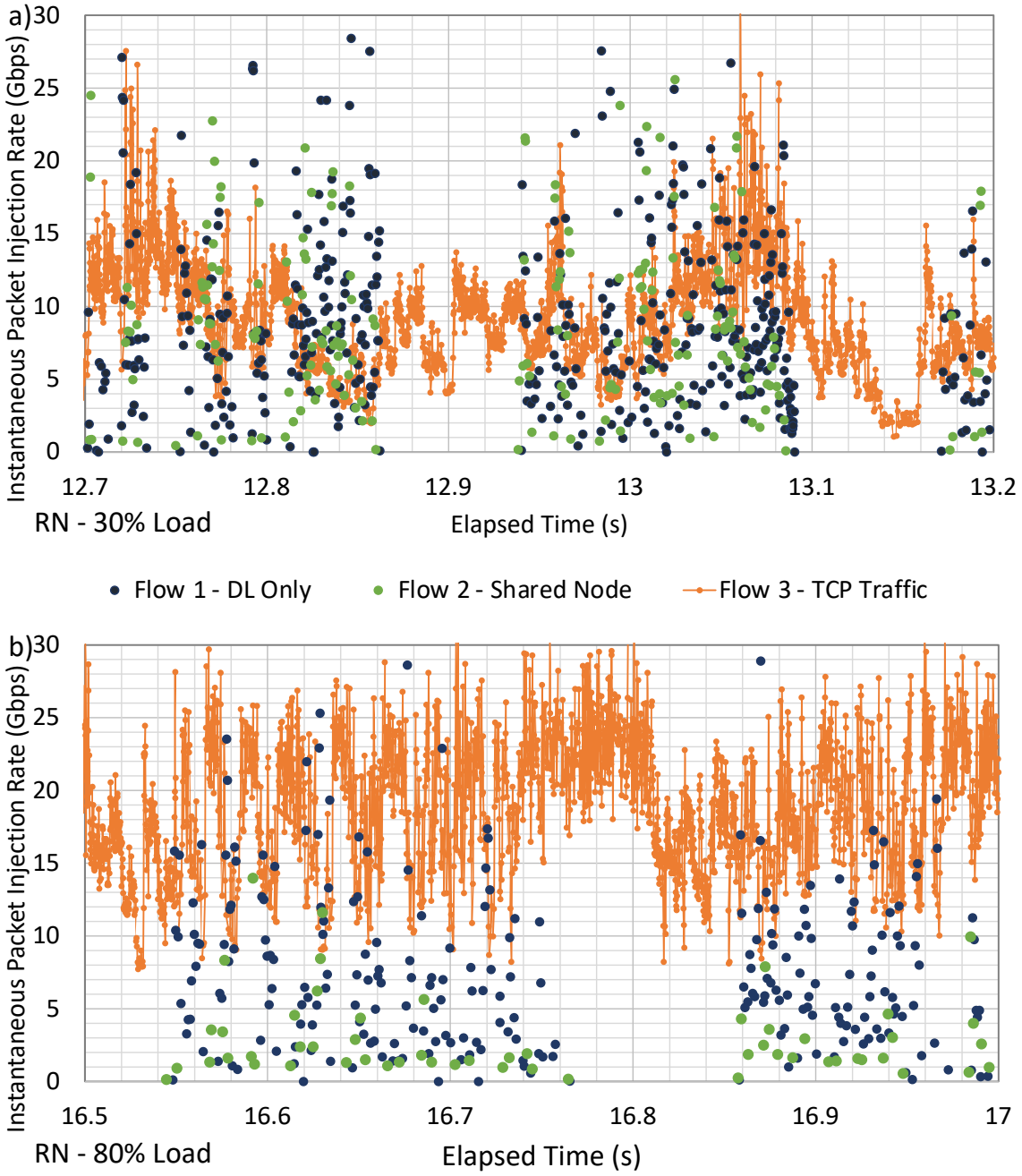

Figure 3-10: Instantaneous synchronized packet injection rate readings for several iterations of ResNet-50 with (a) 8 Gbps and (b) 20 Gbps of consistent web-search cross traffic. The distance between two subsequent iterations is clearly visible at several times in each graph. The three flows shown correspond to the flows labeled in Figure 3-9. Each graph uses a moving average filter of width (a):30/(b):50 to increase clarity.

Table 3.11: Average estimated bandwidths for the iterations graphed in Figure 3-10 and Figure 3-11 from the point of view of a DL only node and a DL shared node.

Model	Load	DL Only (Flow 1)	DL Shared (Flow 2)
BERT-large	30%	9.82 Gbps	9.46 Gbps
BERT-large	80%	3.62 Gbps	3.27 Gbps
ResNet-50	30%	8.42 Gbps	8.40 Gbps
ResNet-50	80%	7.24 Gbps	4.13 Gbps

shared node. This is confirmed by the average figures listed in Table 3.11, with the DL only node recording higher bandwidths than the DL shared node in both load cases, and significantly more in the high load case.

Takeaways: The key takeaways from this analysis of ResNet-50 iterations are that: (i) if one node experiences high levels of cross traffic, the other participating nodes do not back off their transmissions to the bandwidth sufficient to meet the new relaxed iteration timing requirements. (ii) Therefore, a more efficient training platform could release this bandwidth to productive use of other cluster tenants without degrading the effect of DL, as the performance of DL iteration times is tied to the worst bottleneck experienced.

BERT Analysis: Individual iterations of BERT are considerably longer than ResNet. In order to view fine details clearly and the transition between iterations, I plot the junction between two iterations and a neighboring region in Figure 3-11. As expected, the two DL traffic samples at each load level ((a) and (b)) have a similar profile, but higher bandwidths are recorded for the DL only node, again aligning with the average values for this time window recorded in Table 3.11.

I observe a smaller difference in performance between the DL only and DL shared observations in Figure 3-11(b), compared to the high load scenario for ResNet-50 explored in Figure 3-10(b). The difference between the two observations remained roughly constant, at approximately 0.4 Gbps from the averages in Table 3.11, despite the overall observed bandwidth available decreasing by a factor of 3 due to the increased competition from cross traffic. This difference is most likely due to the difference in the underlying ML model details. Both BERT and ResNet iteration times are observed to decrease by a factor of two between the low and high cross traffic

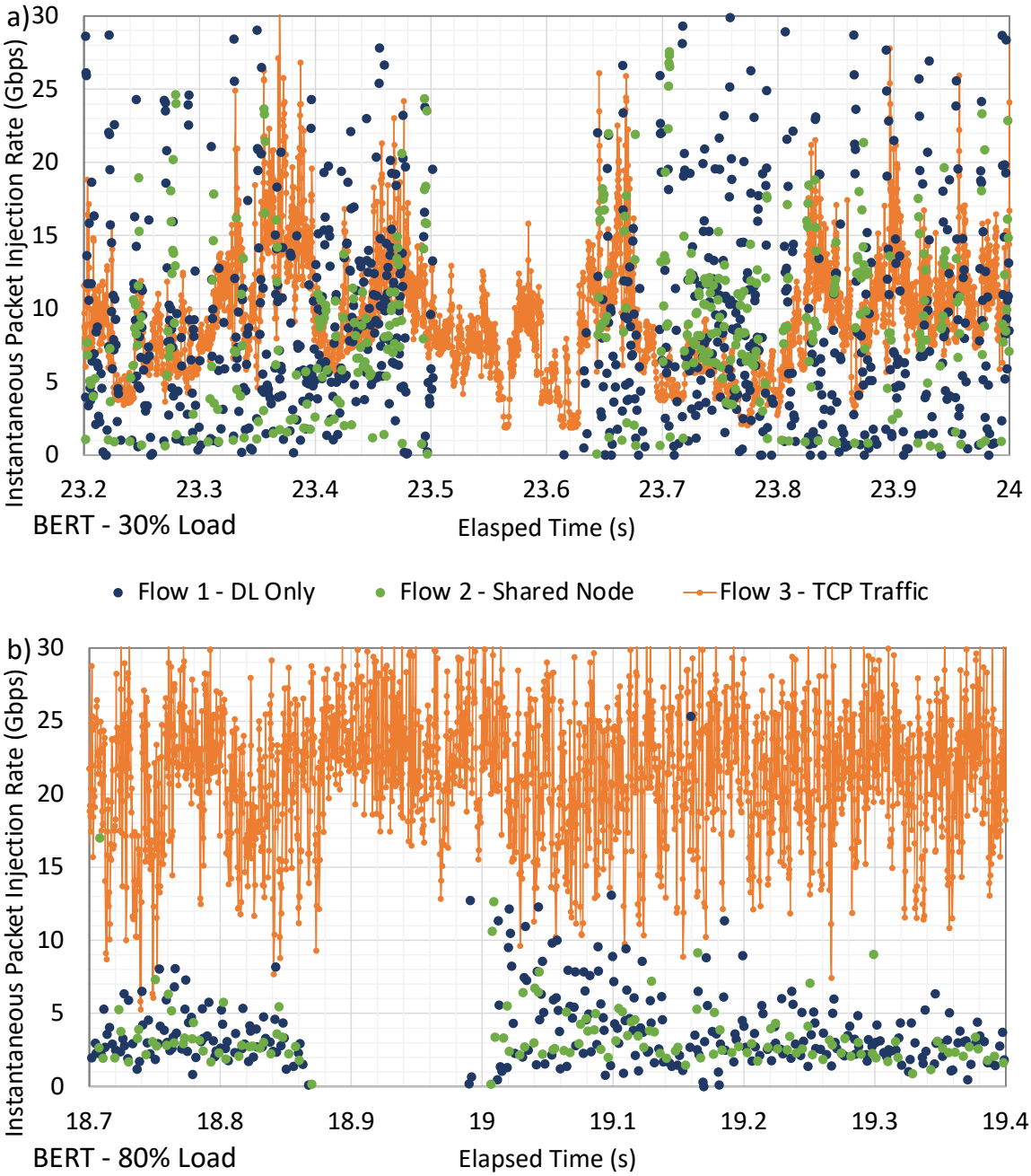

Figure 3-11: Instantaneous synchronized packet injection rate readings for the transition between two iterations of BERT-large (24) with (a) 8 Gbps and (b) 20 Gbps of consistent web-search cross traffic. The transition between the two subsequent iterations is clearly visible at 23.5 and 18.9 s respectively. The three flows shown correspond to the flows labeled in Figure 3-9. Each graph uses a moving average filter of width (a):30/(b):50 to increase clarity.

scenarios tested, but the amount of traffic required by BERT is still over 14× larger, implying a considerably higher network utilization and demand in both scenarios that must be managed.

Takeaways: The key takeaways from this analysis of BERT iterations are that: (i) Both flows 1 and 2 decreased their bandwidth usage by a factor of 3 between the low and high test cases, without observing the significant difference noted in the ResNet experiment. (ii) Therefore, Horovod running an NLP such as BERT appears to send the model parameters piece-wise, leading to a network demand with lower peaks but a longer duration per iteration.

Chapter 4

Conclusion

In this thesis, I characterize the network impacts to and from training Distributed Machine Learning (DL) models utilizing the Horovod framework. In order to accomplish this, I co-develop a custom-designed multi-threaded TCP flow generation script and use it to test the iteration time of DL models against competing similar, consistent web-search and burst-based web-search traffic. This analysis forms an important step towards understanding the impact of network congestion on distributed training workloads. My key results are that:

- The variance and burst behavior of cross traffic plays a significant part in determining the efficiency of distributed ML training in congested network environments. As expected, when congestion is below a theoretically determined threshold, the effects are minor.

- Network bursts have a significant impact on DL training when compared to consistent loads when run at highly congested settings. The presence of competing burst traffic reduces the maximum effective bottleneck utilization to around 90% of the theoretical bandwidth.

- At low loads, minimal variance occurs in recorded iteration times in each series tested, with an average variance of 0.05% compared to mean values, and with no significant tails present with the exception of the BERT model. At high loads, the variance is significant, at an average of 7.8% for consistent and 39.3%

for burst mean values. 99% tails on average 60% larger than the mean value are also observed.

- Considering individual iterations, nodes that communicate via bottlenecked links use less bandwidth for DL training than nodes that are able to communicate without experiencing a bottleneck. This effect is especially pronounced in ResNet-50 and in highly congested settings.

- I develop a model to predict the increase in iteration times due to network congesting and verify its accuracy for the test cases in this thesis.

Building on this thesis, I identify two interesting possibilities for future work: (i) improve and optimize the effective throughput of DL traffic when sharing links with cross traffic at high utilization through customized congestion control, as this would allow either more DL traffic to pass in the same traffic conditions, or would allow the same iteration time to be achieved with less bandwidth required, and (ii) reduce the oversupply of bandwidth to nodes that are unable to progress due to a network delay in another node in the ring. Reassigning this bandwidth would allow additional work to occur on these nodes without affecting the job completion time of the DL task. Taken as a whole, these observations point at the opportunities for improvement a DL aware congestion control protocol would enjoy.

Appendix A

Additional Methodology and Set up Instructions

A.1 Web-Search Load Generation Algorithms

To create the range of cross traffic types required to compete with the distributed ML task I study in this thesis, I utilize a known size distribution drawn from existing work [85, 88] in order to generate web-search profile traffic with arbitrary parameters. The basic version, Algorithm 1, generates a consistent level of randomized traffic by drawing flow sizes from the known weighted distribution, and flow inter-arrival times from a Poisson distribution. The parameter of the Poisson distribution is set such that the expected inter-arrival time multiplied by the expected data per flow gives the requested target data rate.

Algorithm 1 Consistent Web-Search Traffic Generation

Require: target_load, total_time, WS_dist

$data\_req \leftarrow target\_load \times total\_time$

$samples \leftarrow \text{floor}\left(\frac{data\_req}{ave\_data\_per\_sample}\right)$

$\lambda \leftarrow \frac{total\_time}{samples}$

$flow\_inter\_t \leftarrow \text{Poisson}\,(\lambda, samples)$

$flow\_t \leftarrow \text{CumSum}(flow\_inter\_t)$

$sizes \leftarrow \text{RandChoice}(WS\_dist, WS\_weights, samples)$

return flow_t, sizes

The procedure used to generate burst web-search traffic, outlined in Algorithm 2, follows a similar logic, except using a repeating pattern formed of two elements. These are a burst element and a consistent element, which are then repeated to generate the full traffic file. As the target burst duration for this thesis of 1000 μs only required a burst element with one or two flows for most target loads in the experimental range (0-25 Gbps), I set up Algorithm 2 to force the burst phase to have 2 flows, with a total data size proportionate to 3/10 of the total data to be transmitted in each repeating period. The data split between the two burst flows, and their arrival time, are then randomly sampled from a normal distribution and an exponential distribution respectively.

The consistent element is generated as per Algorithm 1, however, to ensure a sufficiently representative number of flows occur during the consistent phase, a minimum of 8 flows are generated. In order to ensure that the correct total volume of data is generated (7/10 of the pattern's total in 9/10 of the time), I scale down the flow size distribution accordingly before applying Algorithm 1.

Algorithm 2 Burst Web-Search Traffic Generation

Require: target_load, brst_t, total_time, WS_dist

$scale \leftarrow 1$
$brst\_period \leftarrow 10 \times brst\_t$
$num\_brst \leftarrow \frac{total\_time}{brst\_period}$
$brst\_list \leftarrow$ randomize_burst$(brst_period, num_brst)$

$no\_burst\_data \leftarrow target\_load \times \frac{total\_time}{num\_brst}\left(1 - \frac{3}{10}\right)$
$no\_brst\_sp \leftarrow \text{floor}\left(\frac{no\_burst\_data}{ave\_data\_per\_sample}\right)$ ▷ ave_data is a known constant
if no_brst_sp is less than 8 **then**
    $no\_brst\_sp \leftarrow 8$
    $ave\_data\_req \leftarrow \frac{no\_burst\_data}{8}$
    $scale \leftarrow \frac{ave\_data\_req}{ave\_data\_per\_sample}$
end if

for i **in** num_brst **do**
    $flow\_t, sizes$ append $brst\_list[i]$
    $flow\_t, sizes$ append draw_Poisson$(no_brst_sp, brst_t, scale, WS_dist)$
end for
return flow_t, sizes

A.2 Congestion Point Identification

To obtain this measurement, I first construct a line along the linear portion of the graph. Second, I construct a tangent to the curved portion of the graph, as close to the interception with the linear portion as possible. Both these lines are shown as dotted for the two series in consideration in Figure A-1. Once this is done, I find the closest data point to this measurement, and take this iteration time and bandwidth as the experimental values for the congestion point. This is shown in Figure A-1 using thin solid lines.

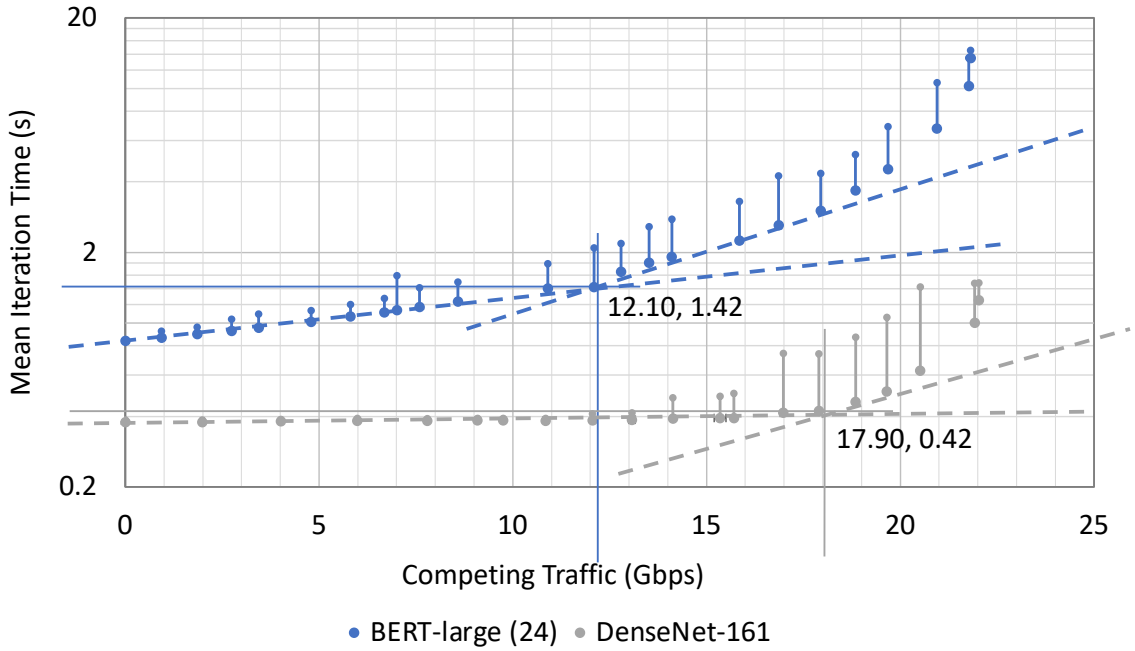

Figure A-1: Method for calculating the congestion point on an iteration time against bandwidth graph (sample graph taken is web-search traffic).

Appendix B

Additional Results and Details

B.1 DL Performance utilizing RDMA and TCP for Transport

To provide an alternate view of the analysis in Section 3.4.2, Figure B-1 compares the distributions of samples taken from DL tasks utilizing RDMA and TCP for communication with (a) light and (b) heavy loads provided by consistent web-search cross traffic. As expected, in the case of light loading, the large models (VGG, BERT) perform better using RDMA, as they are able to take greater advantage of the direct memory access. In contrast, the RDMA ResNet series is slower throughout its distribution than the TCP version in light loading.

In Figure B-1 (b) I observed the same trend as discussed in Section 3.4.2, as the differences between each protocol are significantly reduced for all models compared to the low load case. At this level, the distributions of the two protocols are closely comparable, which aligns with the key takeaway reached in the main body of this thesis. As a subtlety, the tail behavior of the RDMA mode is measurably worse, which would affect the 99^{th} percentile and tail measurements, but the gap is not so dramatic as to invalidate the comparison.

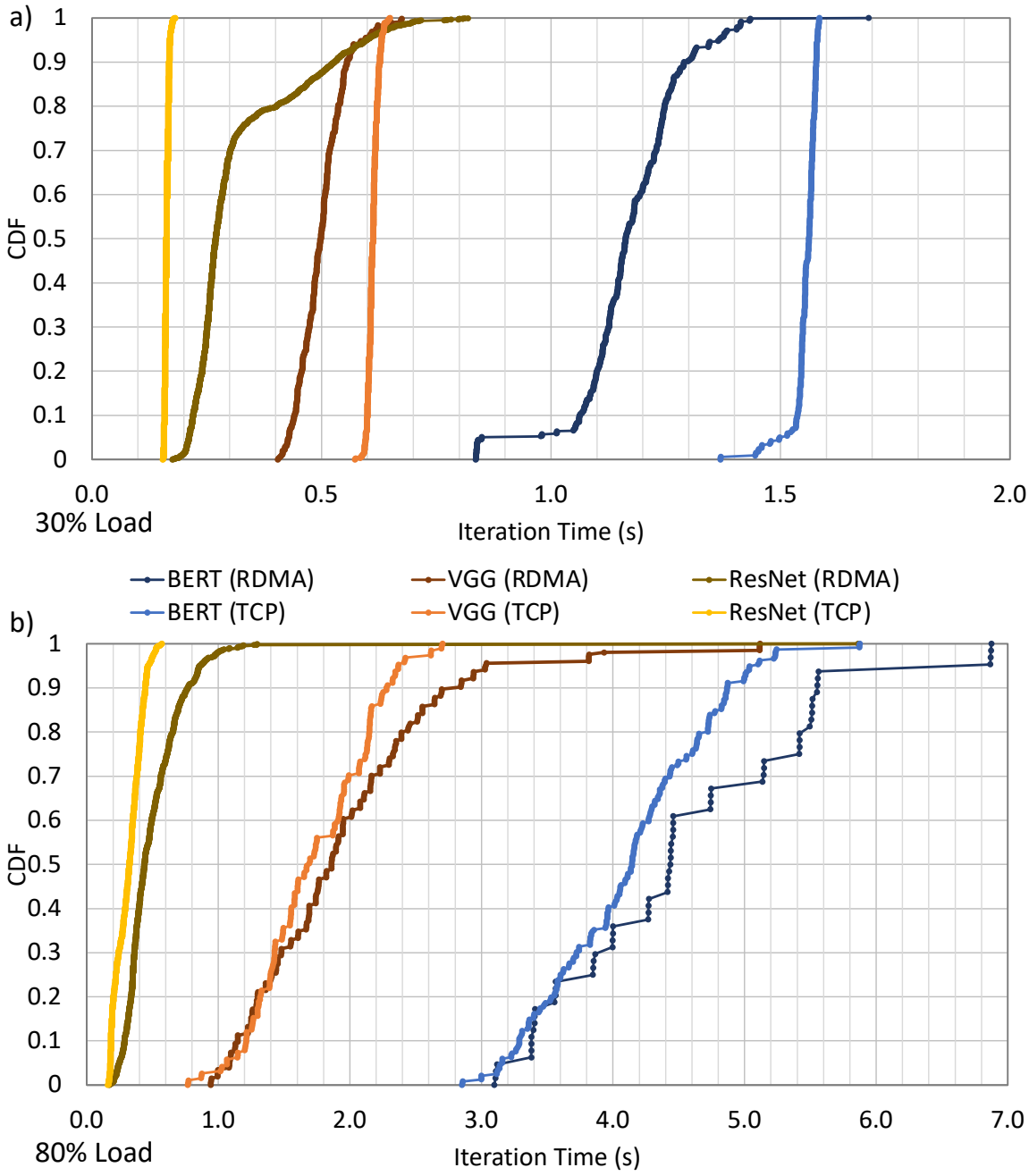

Figure B-1: CDF of DL iteration times for three different size models, BERT, VGG and ResNet, under (a) light (30%) and (b) heavy (80%) cross traffic. The two series show the performance of DL training utilizing either the RDMA protocol or TCP for its communication phase. Note the time axis in graph (a) is considerably shorter than in graph (b).

B.2 Empirical Models from Traffic Graphs

I utilize re-scaled hyperbola versus iteration time graphs to estimate an appropriate empirical mathematical model to apply to each traffic type, as discussed in Section 3.2, Section 3.3.1, and Section 3.3.2. I produce a new series of plots, shown in Figure B-2, and fit least-square error trends to each data series and obtain coefficients of determination (R^2) values for each model. After examination and testing, I found that the large models in Figure B-2(a) follow the re-scaled hyperbola characteristic, the large models in Figure B-2(b) follow the square root of the re-scaled hyperbola, and all models in Figure B-2(c) follow the square of the scaled reciprocal below 17 Gbps. The fits for the high ranges of burst web-search traffic (c) are improved by using the square root model, as discussed in Section 3.4.2, where considerably higher R^2 values are obtained.

To compare the overall behavior of RDMA and TCP-backed training, I fit the empirical models found in Figure B-2 to a dataset captured from each type of DL task with the same consistent web-search cross traffic. I utilized the square root hyperbola model across all data types in this figure, with the exception of the RDMA ResNet-50 series, as this is a smaller model with low figures only, so the squared hyperbola model is more appropriate. The fits from this process and their coefficients of determination are provided in Figure B-3, and show that the difference between these protocols is minimal in practice, aligning with the other results.

B.3 Web-Search Traffic Samples

In order to verify that the generated web-search traffic has the expected profile, I capture `tcpdump` traces for burst and consistent web-search samples to compare against my expectations and the generated configuration file. A sample load (configuration) file is shown in Figure B-4, demonstrating the burst and trough pattern expected in a configuration file. Sample traces for (a) burst and (b) consistent web-search cross traffic are shown in Figure B-5 for 10 and 6 Gbps loads. As these points are re-

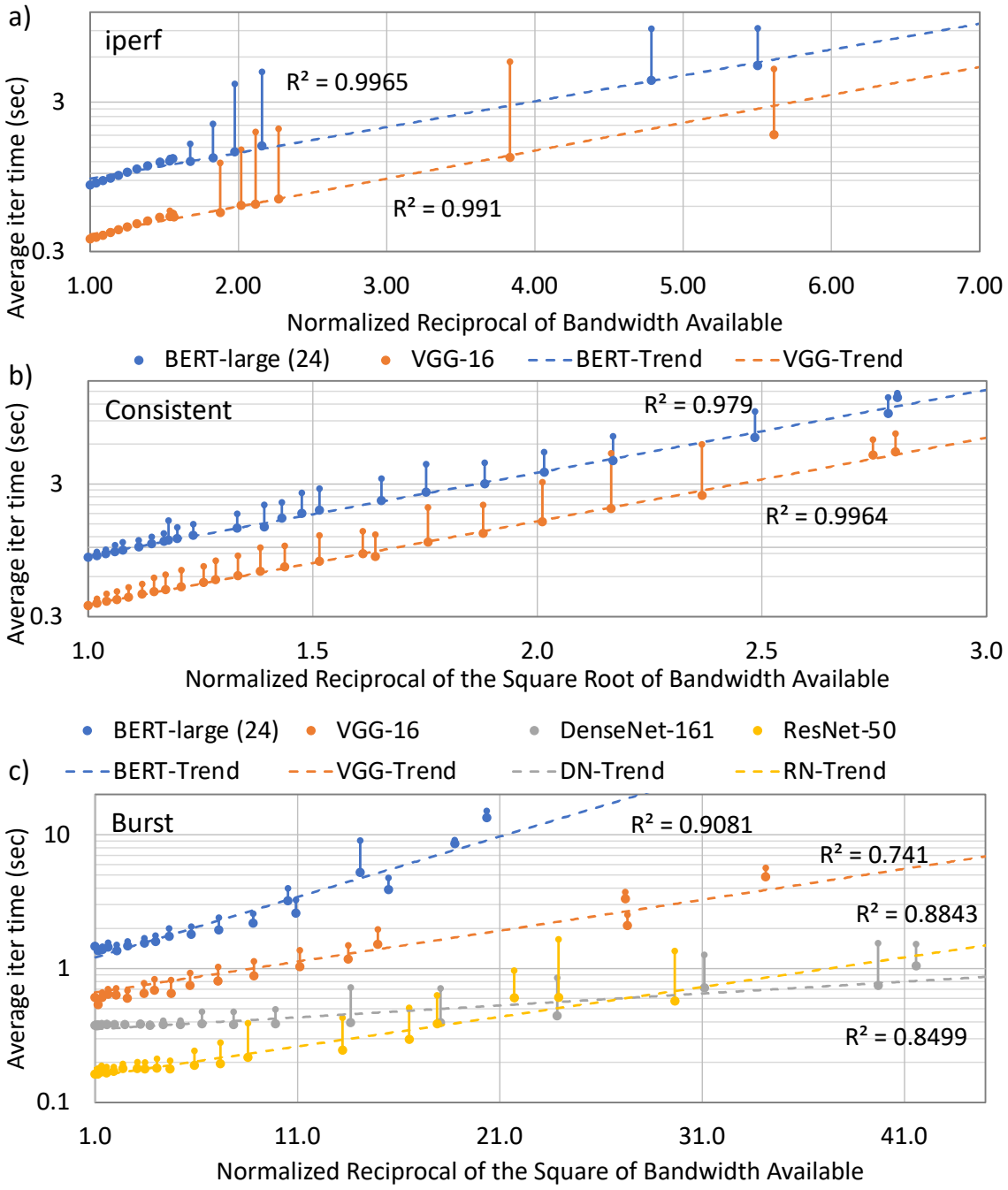

Figure B-2: Exponential trend-lines for the similar, consistent web-search and burst web-search cross traffic scenarios, listing coefficients of determination for each fit case. These R^2 values are reported in the main body of this thesis.

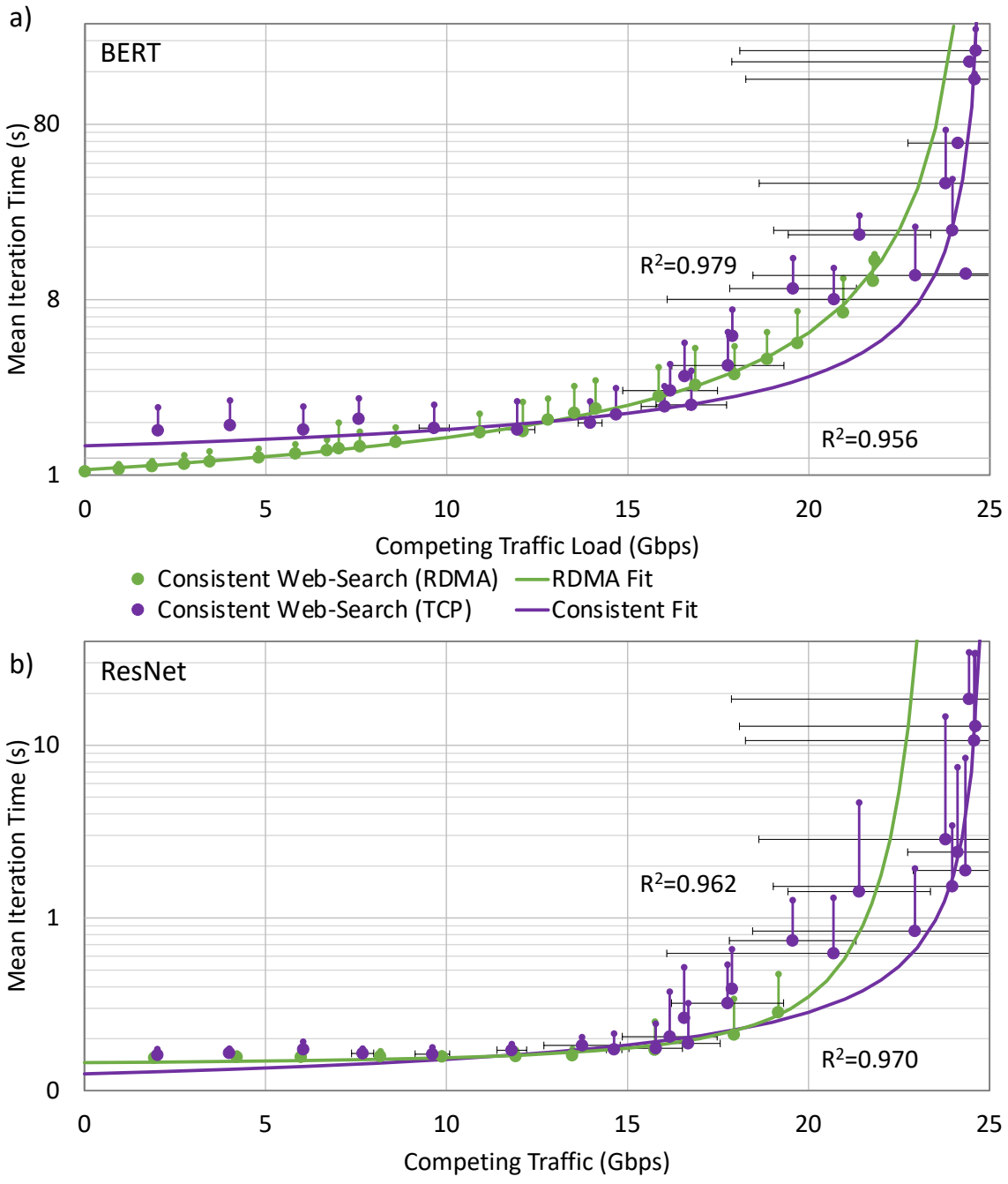

Figure B-3: Model fits for the iteration time behavior of (a) BERT-large, (b) ResNet DL training running over RDMA and TCP while competing against consistent web-search cross traffic. The maximum link capacity is 25 Gbps. The coefficients of determination for each case is labeled next to the fit, with the corresponding equation provided in Table 3.9.

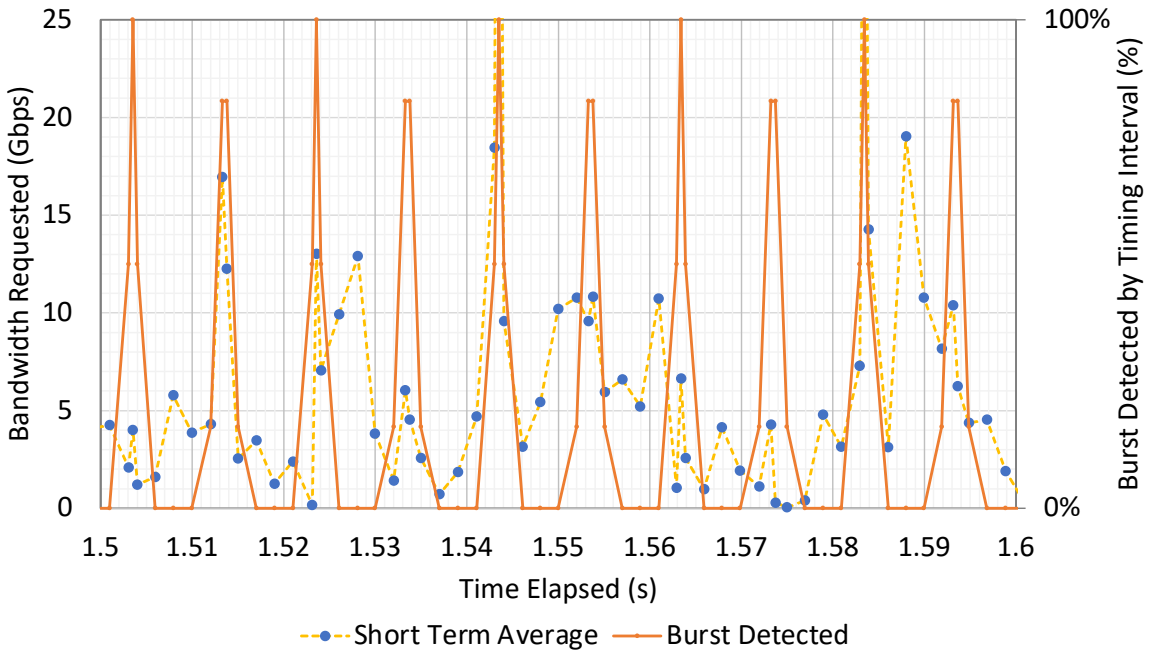

Figure B-4: Example of a load (configuration) file generated by Algorithm 2. Burst locations are automatically identified in orange, with the requested flows marked with blue dots, ordered by the yellow line. This case creates bursts of 1000 μs length with 10 ms spacing and an average load of 7.3 Gbps.

ceived from a `tcpdump` trace rather than a deterministic list of flows, the data points in this figure correspond to several averaged packets, rather than to discrete flows as in Figure B-4. A moving average filter is applied to visualize the traffic pattern at an appropriate level of depth, as looking at individual packet arrivals would be overwhelming and the estimated bandwidths observed would be less accurate, but an overly large filter window would blur out the random oscillation details I wish to verify, so a compromise width of 18 samples is selected.

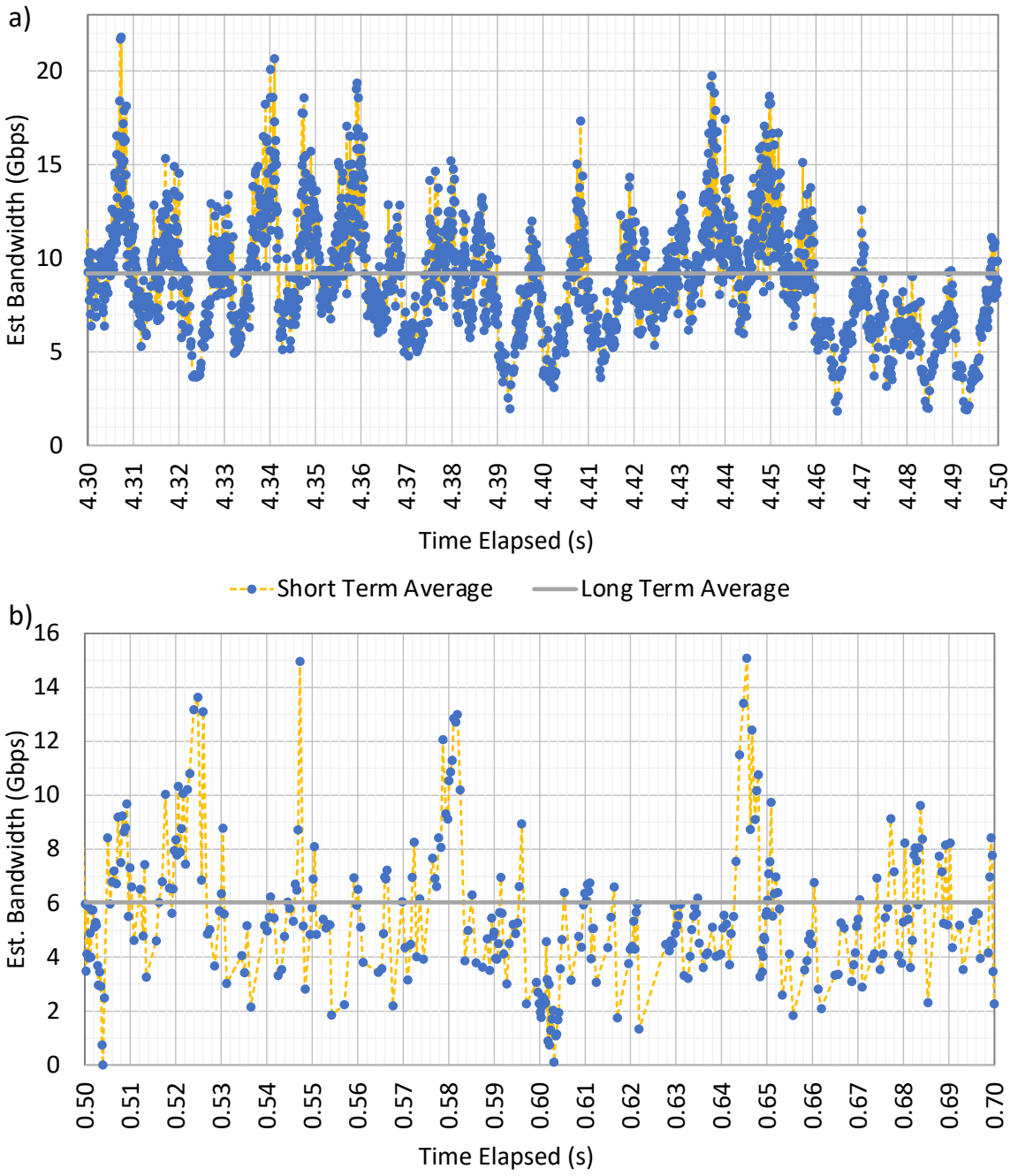

Figure B-5: Examples of (a) burst web-search traffic and (b) consistent web-search traces from `tcpdump`. The gray line corresponds to the long term average measured across the sample trace, the short term average is obtained from a moving average filter of width 18.

Bibliography

[1] Jaime Sevilla. Parameter counts in Machine Learning, 2021. [Online]. Available: `https://towardsdatascience.com/parameter-counts-in-machine-learning-a312dc4753d0` [Accessed: 2021-08-15].

[2] Karl Rupp. Data repository for my blog series on microprocessor trend data., 2021. [Online]. Available: `https://github.com/karlrupp/microprocessor-trend-data` [Accessed: 2021-08-15].

[3] NVIDIA Corporation. Nvidia Corporation Website, 2020. [Online]. Available: `https://www.nvidia.com` [Accessed: 2020-07-22].

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI '04)*, pages 137–150, San Francisco, CA, 12 2004. USENIX Association.

[5] Matei Zaharia, Andy Konwinski, Anthony D Joseph, and Randy Katz. Improving MapReduce Performance in Heterogeneous Environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI '08)*, pages 29–42, Berkeley, CA, 12 2008. USENIX Association.

[6] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint*, arXiv:1802.05799, 2 2018.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, MN, 6 2019. Association for Computational Linguistics.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle,

M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Proceedings of Advances in Neural Information Processing Systems 33 (NeurIPS '20)*, volume 33, pages 1877–1901, Virtual Event, 12 2020. Curran Associates, Inc.

[9] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter. BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems 1 (MLSys '19)*, pages 241–251, Palo Alto, CA, 4 2019.

[10] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pages 1–15, Huntsville, Ontario, Canada, 10 2019. Association for Computing Machinery.

[11] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint*, arXiv:1706.02677, 6 2017.

[12] NVIDIA Corporation. Nvidia Ampere Architecture in Depth, 2020. [Online]. Available: `https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/` [Accessed: 2021-08-16].

[13] Microsoft. Project Catapult - Microsoft Research, 2018. [Online]. Available: `https://www.microsoft.com/en-us/research/project/project-catapult` [Accessed: 2021-08-16].

[14] Google. Cloud TPU Tools, 2021. [Online]. Available: `https://cloud.google.com/tpu/docs/cloud-tpu-tools` [Accessed: 2021-08-15].

[15] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. SiP-ML: High-Bandwidth Optical Network Interconnects for Machine Learning Training. In *Proceedings of the 2021 ACM SIGCOMM Conference (SIGCOMM '21)*, pages 657–675, Virtual Event, 8 2021. Association for Computing Machinery.

[16] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting Distributed Synchronous SGD. *arXiv preprint*, arXiv:1604.00981, 4 2016.

[17] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Proceedings of Advances in Neural Information*

Processing Systems 32 (NeurIPS '19), Vancouver, BC, Canada, 12 2019. Curran Associates, Inc.

[18] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steve Reinhardt, Adrian Caulfield, Eric Chung, and Doug Burger. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18)*, pages 1–14, Los Angeles, CA, 6 2018. Association for Computing Machinery.

[19] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond Data and Model Parallelism for Deep Neural Networks. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems 1 (MLSys '19)*, pages 1–13, Palo Alto, CA, 4 2019.

[20] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. FireCaffe: Near-Linear Acceleration of Deep Neural Network Training on Compute Clusters. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '16)*, pages 2592–2600, Las Vegas, NV, 6 2016. IEEE Computer Society.

[21] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*, Atlanta, GA, 11 2020. IEEE Press.

[22] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil Devanur, and Ion Stoica. Blink: Fast and Generic Collectives for Distributed ML. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems 2 (MLSys '20)*, pages 172–186, Austin, TX, 3 2020.

[23] Dan Alistarh, Demjan Grubic, Jerry Z Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In I. Guyon, U. V. Luxburg, S. S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30 (NIPS '17)*, pages 1710–1721, Long Beach, CA, 12 2017. Curran Associates, Inc.

[24] Leslie Kaelbling, Tomás Lozano-Pérez, Isaac Chuang, and Duane Boning. *6.036 Introduction to Machine Learning Course Notes*. Massachusetts Institute of Technology, Cambridge, MA, 2019.

[25] Tanya Kolosova and Samuel Berestizhevsky. *Supervised Machine Learning*. CRC Press, Boca Raton, FL, 1st edition, 2021.

[26] Taeho Jo. *Machine Learning Foundations*. Springer International Publishing, Cham, Switzerland, 1st edition, 2021.

[27] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. cpSGD: Communication-efficient and differentially-private distributed SGD. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS '18)*, pages 7564–7575, Montréal, Québec, Canada, 12 2018. Curran Associates, Inc.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '16)*, pages 770–778, Las Vegas, NV, 6 2016. IEEE Computer Society.

[29] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '17)*, pages 2261–2269, Honolulu, HI, 7 2017. IEEE Computer Society.

[30] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv preprint*, ArXiv:1409.1556v6, 4 2015.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '09)*, pages 248–255, Miami, FL, 6 2009. IEEE Computer Society.

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *Open AI preprint*, 2019.

[33] Tal Ben-Nun and Torsten Hoefler. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *ACM Computing Surveys*, 52(4-65):1–43, 8 2019.

[34] Jin Kyu Kim, Abutalib Aghayev, Garth A Gibson, and Eric P Xing. STRADS-AP: Simplifying Distributed Machine Learning Programming without Introducing a New Programming Model. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '19)*, pages 207–222, Renton, WA, 7 2019. USENIX Association.

[35] Nadeen Gebara, Manya Ghobadi, and Paolo Costa. In-network Aggregation for Shared Machine Learning Clusters. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems 3 (MLSys '21)*, pages 829–844, Virtual Event, 3 2021.

[36] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. KungFu: Making Training in Distributed Machine Learning Adaptive. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 937–954, Virtual Event, 11 2020. USENIX Association.

[37] Siyuan Zhuang, Zhuohan Li, Danyang Zhuo, Stephanie Wang, Eric Liang, Robert Nishihara, Philipp Moritz, and Ion Stoica. Hoplite: Efficient and Fault-Tolerant Collective Communication for Task-Based Distributed Systems. In *Proceedings of the 2021 ACM SIGCOMM Conference (SIGCOMM '21)*, pages 641–656, Virtual Event, 8 2021. Association for Computing Machinery.

[38] Jiawei Fei, Chen-Yu Ho, Atal N. Sahu, Marco Canini, and Amedeo Sapio. Efficient Sparse Collective Communication and Its Application to Accelerate Distributed Deep Learning. In *Proceedings of the 2021 ACM SIGCOMM Conference (SIGCOMM '21)*, pages 676–691, Virtual Event, 8 2021. Association for Computing Machinery.

[39] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. Scaling Distributed Machine Learning with In-Network Aggregation. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*, pages 785–808, Virtual Event, 4 2021. USENIX Association.

[40] Andrew Gibiansky and Joel Hestness. Baidu all-reduce, 2017. [Online]. Available: `https://github.com/baidu-research/tensorflow-allreduce` [Accessed: 2021-09-02].

[41] NVIDIA Corporation. NVIDIA Collective Communications Library (NCCL), 2021. [Online]. Available: `https://docs.nvidia.com/deeplearning/nccl/index.html` [Accessed: 2021-08-26].

[42] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

[43] NVIDIA Corporation. System Specifications DGX-2, 2019. [Online]. Available: `www.nvidia.com/DGX-2` [Accessed: 2021-08-16].

[44] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the Effects of Data Parallelism on Neural Network Training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

[45] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard,

Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 265–283, Savannah, GA, 11 2016. USENIX Association.

[46] Constantinos Daskalakis, Nishanth Dikkala, and Siddhartha Jayanti. HOGWILD!-Gibbs can be PanAccurate. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS '18)*, Montréal, Québec, Canada, 12 2018. Curran Associates, Inc.

[47] Julaiti Alafate and Yoav Freund. Tell Me Something New: A New Framework for Asynchronous Parallel Learning. *arXiv preprint*, arXiv:1805.07483, 5 2018.

[48] Qirong Ho, James Cipar, Henggang Cui, Jin Kyu Kim, Seunghak Lee, Phillip B Gibbons, Garth A Gibson, Gregory R Ganger, and Eric P Xing. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems 26 (NIPS '13)*, pages 1225–1233, Lake Tahoe, NV, 12 2013. Curran Associates, Inc.

[49] Hang Shi, Yue Zhao, Bofeng Zhang, Kenji Yoshigoe, and Athanasios V Vasilakos. A Free Stale Synchronous Parallel Strategy for Distributed Machine Learning. In *Proceedings of the 2019 International Conference on Big Data Engineering (BDE '19)*, pages 23–29, Hong Kong, 6 2019. Association for Computing Machinery.

[50] Kaan Kara, Dan Alistarh, Gustavo Alonso, Onur Mutlu, and Ce Zhang. FPGA-accelerated dense linear machine learning: A precision-convergence trade-off. In *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM '17)*, pages 160–167, Napa, CA, 6 2017. IEEE.

[51] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. Band-limited Training and Inference for Convolutional Neural Networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML '19)*, pages 1745–1754, Long Beach, CA, 6 2019. PMLR.

[52] Lam M. Nguyen, Phuong Ha Nguyen, Peter Richtárik, Katya Scheinberg, Martin Takáč, and Marten van Dijk. New Convergence Aspects of Stochastic Gradient Algorithms. *Journal of Machine Learning Research*, 20(176):1–49, 11 2019.

[53] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi Wei Lin, and Varugis Kurien. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM Conference on*

Special Interest Group on Data Communication (SIGCOMM '15), pages 139–152, London, United Kingdom, 8 2015. Association for Computing Machinery.

[54] Siyu Wang, Yi Rong, Shiqing Fan, Zhen Zheng, Lansong Diao, Guoping Long, Jun Yang, Xiaoyong Liu, and Wei Lin. Auto-MAP: A DQN Framework for Exploring Distributed Execution Plans for DNN Workloads. *arXiv preprint*, arXiv:2007.04069, 7 2020.

[55] Peng Jiang and Gagan Agrawal. A Linear Speedup Analysis of Distributed Deep Learning with Sparse and Quantized Communication. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS '18)*, pages 2525–2536, Montréal, Québec, Canada, 12 2018. Curran Associates, Inc.

[56] Aurick Qiao, Abutalib Aghayev, Weiren Yu, Haoyang Chen, Qirong Ho, Garth A Gibson, and Eric P Xing. Litz: Elastic Framework for High-Performance Distributed Machine Learning. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '18)*, pages 631–644, Boston, MA, 7 2018. USENIX Association.

[57] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A Generic Communication Scheduler for Distributed DNN Training Acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pages 16–29, Huntsville, Ontario, Canada, 10 2019. Association for Computing Machinery.

[58] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 499–514, Virtual Event, 11 2020. USENIX Association.

[59] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H Campbell. TicTac: Accelerating Distributed Deep Learning with Communication Scheduling. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems 1 (MLSys '19)*, pages 418–430, Palo Alto, CA, 4 2019.

[60] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 863–879, Virtual Event, 11 2020. USENIX Association.

[61] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic Resource Sharing for Distributed Deep Learning. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*, pages 721–739, Virtual Event, 4 2021. USENIX Association.

[62] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, pages 583–598, Broomfield, CO, 10 2014. USENIX Association.

[63] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*, pages 1–18, Virtual Event, 7 2021. USENIX Association.

[64] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 481–498, Virtual Event, 11 2020. USENIX Association.

[65] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 533–548, Virtual Event, 11 2020. USENIX Association.

[66] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and Efficient GPU Cluster Scheduling. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*, pages 289–304, Santa Clara, CA, 2 2020. USENIX Association.

[67] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*, pages 485–500, Boston, MA, 2 2019. USENIX Association.

[68] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, pages 47–62, Huntsville, Ontario, Canada, 10 2019. Association for Computing Machinery.

[69] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken. Exploring Hidden Dimensions in Accelerating Convolutional Neural Networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML '18)*, pages 2274–2283, Stockholm, Sweden, 7 2018. PMLR.

[70] Haojie Wang, Jidong Zhai, Mingyu Gao, Zixuan Ma, Shizhi Tang, Liyan Zheng, Yuanzhi Li, Kaiyuan Rong, Yuanyong Chen, and Zhihao Jia. PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*, pages 37–54, Virtual Event, 7 2021. USENIX Association.

[71] Hongyu Zhu, Amar Phanishayee, and Gennady Pekhimenko. Daydream: Accurately Estimating the Efficacy of Optimizations for DNN Training. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '20)*, pages 337–352, Virtual Event, 6 2020. USENIX Association.

[72] Eli Cortez, Mark Russinovich, Anand Bonde, Marcus Fontoura, Alexandre Muzio, and Ricardo Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP '17)*, pages 153–167, Shanghai, China, 10 2017. Association for Computing Machinery.

[73] Brian Lebiednik, Aman Mangal, and Niharika Tiwari. A Survey and Evaluation of Data Center Network Topologies. *arXiv preprint*, arXiv:1605.01701, 5 2016.

[74] Muhammad Tirmazi, Adam Barker, Nan Deng Google, Md E Haque Google, Zhijing Gene Qin Google, Steven Hand Google, Mor Harchol-Balter CMU, John Wilkes Google, Nan Deng, Md E Haque, Zhi-jing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The Next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*, Heraklion, Greece, 4 2020. Association for Computing Machinery.

[75] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. ATP: In-network Aggregation for Multi-tenant Learning. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*, pages 741–761, Virtual Event, 4 2021. USENIX Association.

[76] Liang Luo, Peter West, Jacob Nelson, Arvind Krishnamurthy, and Luis Ceze. PLink: Discovering and Exploiting Locality for Accelerated Distributed Training on the Public Cloud. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems 2 (MLSys '20)*, pages 82–97, Austin, TX, 3 2020.

[77] NVIDIA Corporation. Developer Tools Overview | NVIDIA Developer, 2021. [Online]. Available: `https://developer.nvidia.com/tools-overview` [Accessed: 2021-08-15].

[78] NVIDIA Corporation. Profiler User's Guide, 2021. [Online]. Available: `https://docs.nvidia.com/cuda/profiler-users-guide` [Accessed: 2021-09-02].

[79] Intel Corporation. Intel VTune Profiler, 2021. [Online]. Available: `https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top.html` [Accessed: 2021-08-15].

[80] Mellanox. Understanding mlx5 ethtool Counters, 2021. [Online]. Available: `https://community.mellanox.com/s/article/understanding-mlx5-ethtool-counters` [Accessed: 2021-08-16].

[81] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv preprint*, arXiv:1512.01274, 12 2015.

[82] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla, Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Lu Fang Facebook, Junjie Bai Facebook, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 32 (NeurIPS '19)*, Vancouver, BC, Canada, 12 2019. Curran Associates, Inc.

[83] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*, pages 1–13, San Jose, CA, 10 2012. Association for Computing Machinery.

[84] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pages 123–137, London, United Kingdom, 8 2015. Association for Computing Machinery.

[85] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*, pages 63–74, New Delhi, India, 8 2010. Association for Computing Machinery.

[86] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*, pages 51–62, Barcelona, Spain, 8 2009. Association for Computing Machinery.

[87] Theophilus Benson, Aditya Akella, and David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th Annual Conference on Internet Measurement (IMC '10)*, pages 267–280, Melbourne, Australia, 11 2010. Association for Computing Machinery.

[88] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal near-Optimal Datacenter Transport. In *Proceedings of the ACM SIGCOMM 2013 Conference (SIGCOMM '13)*, pages 435–446, Hong Kong, 8 2013. Association for Computing Machinery.

[89] Advanced Micro Devices, Inc. AMD EPYC 7002 Series - 7502P, 2020. [Online]. Available: `https://www.amd.com/en/support/cpu/amd-epyc/amd-epyc-7002-series/amd-epyc-7502p` [Accessed: 2021-08-26].

[90] Juniper Networks, Inc. MX480 Universal Routing Platform, 2021. [Online]. Available: `https://www.juniper.net/us/en/products/routers/mx-series/mx480-universal-routing-platform.html` [Accessed: 2021-08-26].

[91] Linux Foundation AI & Data. Horovod Documentation, 2021. [Online]. Available: `https://horovod.readthedocs.io/en/stable` [Accessed: 2021-08-26].

[92] Pytorch Authors. Pytorch Documentation, 2021. [Online]. Available: `https://pytorch.org` [Accessed: 2021-08-26].

[93] Tensorflow Authors. Tensorflow Documentation, 2021. [Online]. Available: `https://www.tensorflow.org` [Accessed: 2021-08-26].

[94] NVIDIA Corporation. Developing a Linux Kernel Module using GPUDirect RDMA. Technical report, NVIDIA, Santa Clara, CA, 9 2021.

[95] Hejer Shaiek, Stanimire Tomov, Alan Ayala, Azzam Haidar, and Jack J. Dongarra. GPUDirect MPI Communications and Optimizations to Accelerate FFTs on Exascale Systems. In *Proceedings of the 26th European MPI Users' Group Meeting (EuroMPI '19) Posters*, Zurich, Switzerland, 9 2019.

[96] Frontline Systems, Inc. Excel Solver Help, 2021. [Online]. Available: `https://www.solver.com/excel-solver-online-help` [Accessed: 2021-10-10].