Toward Practical Application-Aware Big Data Systems

by

Jie You

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2022

Doctoral Committee:

Associate Professor Mosharaf Chowdhury, Chair
Professor Samir Khuller, Northwestern University
Associate Professor Harsha V. Madhyastha
Professor Lei Ying

Jie You

jieyou@umich.edu

ORCID iD: 0000-0002-2037-5366

To my family

Acknowledgments

First and foremost, I am deeply indebted to my advisor, Professor Mosharaf Chowdhury, for his continuous support, sharp insight, and tremendous patience during my PhD journey. Professor Mosharaf Chowdhury took me under his wing when I was inexperienced in system research and provided the most needed guidance. He taught me the most important lesson in doing research, which is to appreciate simplicity in solutions instead of complexity, even when solving complicated and intricate problems. I am grateful beyond words for having the opportunity to work with him in these years.

I am truly thankful to my committee members, Professor Samir Khuller, Professor Harsha V. Madhyastha, and Professor Lei Ying. They are my role models and the best shepherds for this dissertation, providing me with constructive feedbacks and valuable comments.

This dissertation is a culmination of many collaborative efforts. I would like to thank Professor Samir Khuller for the insights and feedbacks on the TERRA project. I am thankful to Professor Xin Jin for the advice on the KAYAK project. I cannot express my thanks enough to Jae-Won Chung for his contributions to the ZEUS project. The work would not have been possible without his help in designing the algorithm, implementing the system, running the experiments and polishing the draft. Thanks also go to Sheng Yang for providing the mathematical proof for TERRA, and Jingfeng Wu for the theoretical analysis for KAYAK.

I would also like to show my appreciation to everyone in the Symbiotic Lab: Dr. Juncheng Gu, Dr. Peifeng Yu, Hasan Al Maruf, Fan Lai, Yiwen Zhang, Sanjay S. Singapuram, Jiachen Liu, Jae-Won Chung, Insu Jang. With their company, the lab felt like home. I am specially grateful to Dr. Peifeng Yu, for patiently answering my non-stop engineering and technical questions.

I also want to thank my friends Qi Chen, Yunhan Jia, Yuru Shao, Huan Feng, Xiao Zhu, Kaiyu Yang, Rui Liu, Haizhong Zheng, Ze Zhang, Boyu Tian, Haojun Ma, and Varun Aggarwal. I cherish every moment we spend together in this beautiful small town of Ann Arbor.

Last but not least, this dissertation is dedicated to my father Dr. Yuxiang You, mother Lihua Wei and grandfather Yunxian You, for their love and support. I am forever grateful for my caring and supportive family.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF APPENDICES

ABSTRACT

The increasing number of Internet of things (IoT) and other connected devices has led to a surge in the amount of data collected and analyzed. Data scientists collect useful insights from these data through data analysis or machine learning, all of which are performed on distributed big data infrastructures. Improving the performance and efficiency of such systems can, therefore, improve user experience and reduce their operating cost.

These big data infrastructures abstract away low-level details such as resource allocation and task placement decisions and expose simple APIs for the users. Despite simplifying the development process, the decoupling between applications and infrastructures complicates the optimization for end-to-end performance and efficiency metrics, as the contextual details and optimization objectives of applications cannot be expressed using the APIs. We observe that system designers often optimize only for common system-level metrics such as throughput and latency, inadvertently ignoring application-level semantics. As a result, these best-effort local optimizations do not always improve in application-level performance or efficiency. Moreover, the situation is exacerbated as new emerging applications become more intricate and their interactions with the infrastructures become more complex.

To alleviate the mismatch, we focus on bringing application-awareness into the infrastructures. We explore the opportunities of co-optimizing applications and infrastructures for three popular big data use cases: data analytics, transaction processing, and deep learning, and highlight two ways of implementing application-awareness: white-box design and black-box design. For data analytics, we present TERRA which explicitly passes application-level context into the infrastructures, adopting the white-box design of application-awareness. For transaction processing, it is impossible to maintain an accurate model of the performance curve capturing the relationship between request rate and latency, rendering the white-box design infeasible. Therefore, we present KAYAK with a black-box application-aware design to adaptively arbitrate incoming requests, improving overall throughput and CPU utilization. For deep learning, the white-box solution is intractable due to the increasing depth of the neural network. To this end, we also present ZEUS with a black-box design, adopting a Multi-Armed Bandit algorithm to optimize for faster and more energy-efficient DL training.

In this dissertation, we demonstrate the adoption of application-awareness as a means of optimizing the end-to-end performance and efficiency of the underlying infrastructures with additional

information from the application. We discuss two design patterns for implementing application-awareness, along with their tradeoffs, in the context of three popular big data use cases. The insights from this dissertation promote the adoption of application-awareness and can help future system researchers build more performant and efficient big data infrastructures.

CHAPTER 1

Introduction

In recent years, the growing number of Internet-of-Things (IoT) and other connected devices has created a surge in the amount of data being generated each day [21, 41]. To cope with that, large organizations leverage tens to hundreds of datacenters and edge sites [24, 2, 15, 67, 88, 105, 140] to collect, store and process data such as end-user sessions, monitoring logs, and performance counters.

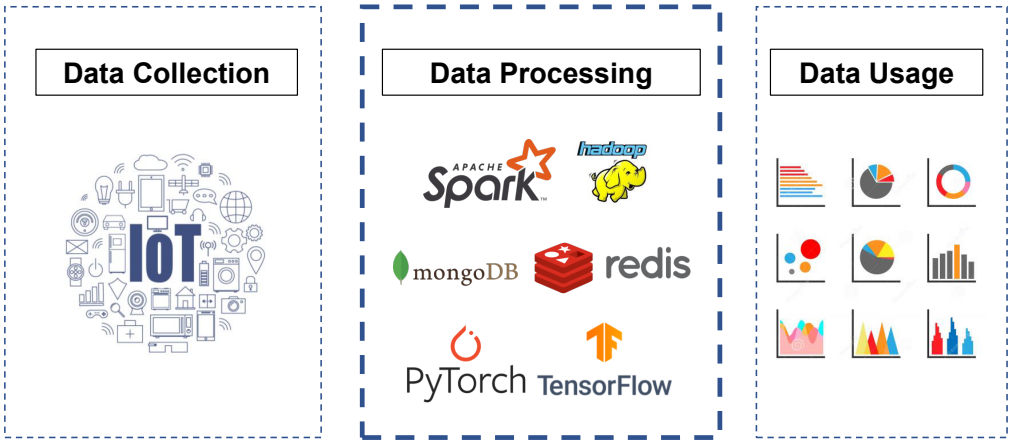
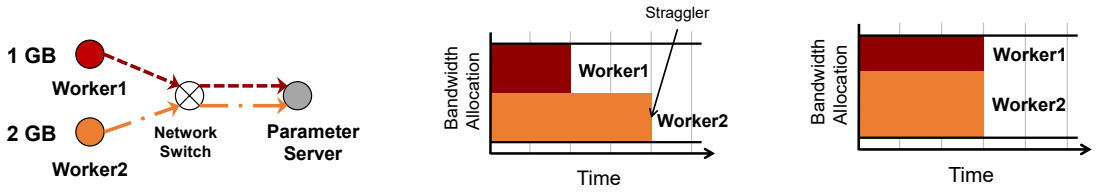


Figure 1.1: Data processing has been the focus of the big data pipeline.

Data scientists gather useful insights from this data typically through a big data pipeline, as shown in Figure 1.1. In this dissertation, we focus on the most resource-demanding and performance-sensitive stage of data processing. Specifically, we focus on three popular use cases: data analytics, transaction processing, and deep learning (DL). Improving the performance and efficiency of this stage has been a central focus for both industry and academia, with many software systems built to support the use cases of this stage efficiently [23, 34, 3, 218, 171].



(a) Gradient accumulation of distributed DL training

(b) Application-agnostic fair bandwidth allocation

(c) Application-aware bandwidth allocation

Figure 1.2: An example of the mismatch between infrastructure-level optimizations and the application-level objective.

These big data infrastructures sit between the applications and the hardware. They handle and abstract away low-level details such as resource allocation and task placements while providing simple APIs for application developers. Although this decoupling between the infrastructures and applications makes the software development process easier, it creates a gap between the system and applications. Application developers using the provided API is unable to express their application-specific performance requirements. Consequently, system designers can only optimize these infrastructures for generic system-level performance and efficiency metric, which does not always align with application-level objectives.

For example, for the gradient accumulation process in Distributed DL training (Figure 1.2a), an application-agnostic fair bandwidth allocation results in longer time spent in this process because `Worker1` would become the straggler (Figure 1.2b). Instead, an application-aware allocation scheme gives `Worker2` more bandwidth to eliminate the straggler, achieving the application-level objective of reduced completion time. This simple example illustrates that the mismatch between the application and the infrastructure can hurt application performance. This mismatch is worsened as the interactions between emerging applications and hardware become more and more complicated [61, 146, 59, 162].

To build a bridge between applications and infrastructures, this dissertation explores the opportunities and proposes different means of co-optimizing applications and infrastructures, by bringing application-awareness into the infrastructures and optimizing for the end-to-end performance and efficiency metrics.

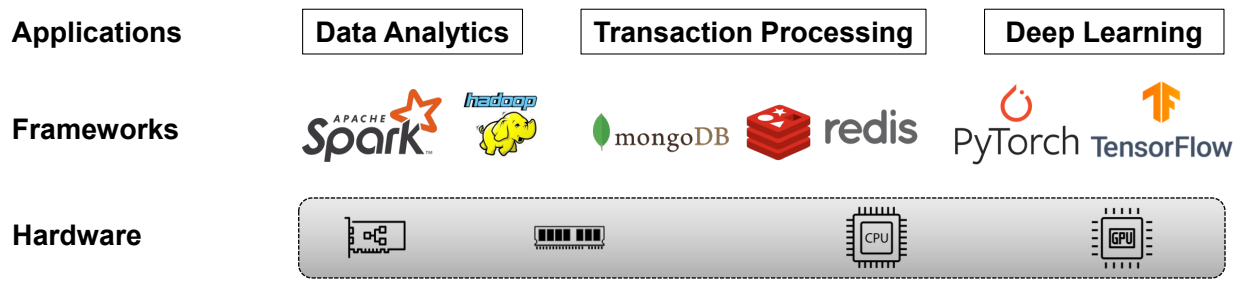


Figure 1.3: Big data infrastructures in the scope of this dissertation.

1.1 Background

This section introduces the background knowledge for the three popular big data use cases of our focus: data analytics, transaction processing, and deep learning.

1.1.1 Data Analytics

By far, analyzing large datasets in a timely and resource-efficient manner has been a primary focus for more than a decade, with frameworks such as Map-Reduce [3, 69] and Spark [218] leading the way. Recently, hybrid multi-cloud computing has emerged as a popular deployment scheme for enterprise customers, with a two-fold increase in deployments over the last few years [39, 36, 28, 33, 27, 22, 38] In this case, data resides in multiple local and remote *sites* among which applications run across – a site can be an on-premise (on-prem) cluster or an off-premise (off-prem) datacenter – connected by wide-area network (WAN) links. Analyzing the data scattered across multiple sites brings new challenges such as expensive WAN bandwidth, and regulatory constraints (e.g., European Union's General Data Protection Regulation) [175, 130, 14].

1.1.2 Transaction Processing

To keep up with uprising capacity and throughput demand, transaction processing systems have undergone drastic evolutions in both hardware and software. On the hardware side, it has evolved from using spinning disks and flash disks to in-memory storage [34, 23], in order to meet the high data throughput and low access latency requirements from applications such as serverless computing and web applications. On the software side, non-relational (NoSQL) database system optimized

for horizontal scaling onto a large cluster of nodes stands out [90] against traditional relational database (RDBMS). Various types of NoSQL databases are now increasingly being employed for large-scale data analytics applications, of which key-value stores become the most popular choice in production environments for their performance and simplicity. Consequently, in-memory key-value stores [23, 34] comes into the limelight in both academy and industry.

When deployed in a distributed setting, the network latency becomes the performance bottleneck for these in-memory key-value stores. A recent line of research has been focusing on utilizing kernel bypass techniques such as RDMA and DPDK to avoid the notoriously slow Linux kernel networking stack [155, 119].

1.1.3 Deep Learning

Complementary to traditional data analytics, deep learning (DL) has emerged as a new way for data scientists to comprehend and make sense of the data, with applications in speech processing, computer vision, natural language processing, recommendation systems, medical applications and so on [91, 70, 143, 188]. It uses artificial neural networks to mimic the neurons in a brain through a set of algorithms, in the same way as the traditional statistical linear regression. Running DL applications involve a large amount of complex matrix algebra operations, which is suitable for specialized hardware such as Graphics Processing Unit (GPU), Tensor Processing Unit (TPU) and Field-Programmable Gate Array (FPGA). Of all the specialized hardware, GPU is the most widely used one for deep learning [99], hence is what we focus on in this dissertation.

Deep learning tasks can be divided by its purpose into two stages: *model training* where a model consisting of neural networks is tuned to minimize its prediction error on training data, and *model serving* where the model is used to perform predictions. Model training works in an iterative fashion. In each iteration, the training process operates on a few samples of training data called mini-batch. It first performs a forward pass with one mini-batch of input data. The error between the outputs of the forward pass and the desired outputs will be propagated over the model through a backward pass, where the gradient for each model parameter is computed. Before the end of iteration, the model parameters will be adjusted using their gradients by the model updating function. This process repeats for many iterations, until the model reaches a target accuracy.

1.2 Thesis Statement

Although the decoupling between the infrastructures and applications makes developing applications easier, it also creates a gap between them. This gap manifests itself in misaligned objectives between application developers and system optimizers, as application-level semantics and objective is abstracted away and not expressed at the infrastructure level.

Thesis Statement – *This dissertation studies the gap between system-level optimizations and application-level improvement for big data systems, and introduces application-awareness into the infrastructure with the purpose of enhancing both performance and efficiency.*

In this dissertation, we investigate the potential of improving end-to-end performance by adopting application-awareness for three popular use cases in big data systems: data analytics, transaction processing and deep learning. Figure 1.4 illustrates the use cases and the corresponding chapters.

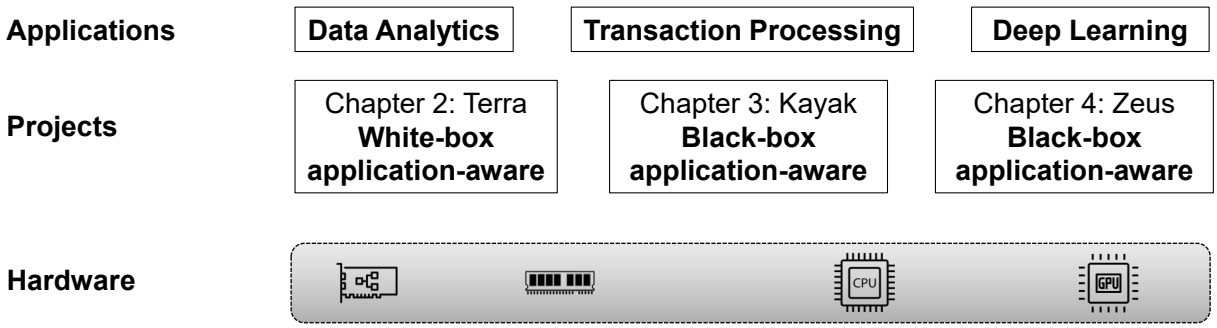


Figure 1.4: Application-awareness in different use cases of big data systems.

We highlight two means of implementing application-awareness:

- *White-box design*. We first modify both the application and the infrastructure so that the application-level context is encoded into the API and passed to the infrastructure. Then we build a precise model of how the infrastructure-level knobs such as routing and rate control decisions can affect our end-to-end application-level objective such as Job Completion Time (JCT). Finally, we apply optimization method to this model, which results in choices of infrastructure-level knobs improving application-level objectives.

The effectiveness of this approach is heavily dependent on the accuracy of the model as well as the availability of detailed application-level context, both of which varies among the use cases. In addition, it can be ad-hoc and hard to implement, as it essentially redefines the abstraction boundary between the application and the infrastructure.

- *Black-box design*. We only modify the infrastructure to optimize for application-level objective, without detailed application-level context. The infrastructure infers application-level context from real-time signals such as performance counters, and optimizes itself accordingly.

  This approach has the benefit of being *transparent*, i.e., the applications do not need to be modified. This is important for adoption of the design and its adaptation to new applications. Moreover, application-level contextual details are not always helpful when it is difficult to accurately model the performance, especially for complex and intricate applications such as DL training. For instance, knowing the structure and hyperparameters of a neural network does not help in predicting the training time until convergence [86].

Next, we discuss the tradeoffs between the two design patterns in the context of the characteristics and contraints of the use cases. In TERRA, we choose to apply the white-box application-awareness because the objective - Job Completion Time - can be accurately modeled with application-level information such as the query execution plan. However, in KAYAK, we choose to apply the black-box design of application-awareness, because we cannot capture the throughput-latency curve using an analytical model with a closed-form formula. For ZEUS, it is even more difficult to capture the relationship between the optimization target and application-level context, because the depth of the neural network and the stochastic nature of the training process makes modeling performance of DL applications accurately very challenging. Hence, we also apply application-awareness with the black-box design in ZEUS.

We highlight our contributions and summarize the results in the next section.

1.3 Summary of Contributions and Results

TERRA We present TERRA, a data transfer framework for hybrid multi-cloud analytics (HMCA) jobs that simultaneously performs scheduling, path selection, and bandwidth allocation for inter-site

shuffles in a scalable manner. We formulate it as an optimization problem that minimizes the time of transferring a WAN traffic matrix, and formally prove its optimality in the offline case. Using this optimization formulation as a building block, we provide support for online scheduling of multiple HMCA jobs. We implement our algorithms in a scalable system (TERRA) that integrates with and complements existing frameworks so that unmodified user-written jobs can transparently run in hybrid multi-cloud environments. TERRA enforces our algorithm-determined data transfer plan, i.e. routes, schedules, and rates, efficiently and can quickly detect and react to runtime dynamics. Evaluations of TERRA with WAN topologies from three major companies and three industry-standard benchmarks (TPCx-BB, TPC-DS, and TPC-H) show that it outperforms the state-of-the-art by up to $3.43\times$ for the smallest and up to $32.04\times$ for the largest deployment.

KAYAK Our project KAYAK shows that transaction processing systems can achieve higher throughput and lower latency with the help of application-level information. In this project, we show that by proactively and adaptively combining RPC *and* KV together, overall throughput and CPU utilization can be improved. We propose an algorithm that dynamically adjusts the rate of requests and the RPC fraction to improve overall request throughput while meeting latency SLO requirements. We then design and implement a system called KAYAK. Our system implementation ensures work conservation and fairness across multiple tenants. Our evaluations show that KAYAK achieves sub-second convergence and improves overall throughput by 32.5%-63.4% for compute-intensive workloads and up to 12.2% for non-compute-intensive and transactional workloads.

ZEUS Training deep neural networks (DNNs) is becoming increasingly more resource- and energy-intensive every year. Unfortunately, existing works primarily focus on optimizing DNN training for faster completion, without considering the impact on energy efficiency.

In this project, we observe that common practices to improve training performance can often lead to inefficient energy usage. More importantly, we demonstrate that there is a tradeoff between energy consumption and performance optimization. To this end, we propose an optimization framework, ZEUS, to navigate this tradeoff by automatically configuring job- and GPU-level configurations of recurring DNN training jobs. ZEUS uses an online exploration-exploitation approach in conjunction with just-in-time energy profiling, averting the need for expensive offline measurements, while

adapting to workload changes and data drifts over time. Our evaluation shows that ZEUS can improve the energy efficiency of DNN training by 15.3%–75.8% for diverse workloads.

1.4 Thesis Structure

The remainder of this dissertation is organized as follows. Chapter 2 presents TERRA, an application-aware data transfer optimizer for hybrid multi-cloud analytics (HMCA) jobs that simultaneously performs scheduling, path selection, and bandwidth allocation for inter-site shuffles in a scalable manner. Chapter 3 presents KAYAK, an application-aware optimizer which proactively and adaptively combines RPC *and* KV together to improve throughput and CPU utilization. Chapter 4 presents ZEUS, an application-aware framework for improving performance and efficiency of deep learning training jobs running on GPU. We conclude the dissertation and discuss future directions in chapter 5.

CHAPTER 2

TERRA: Application-Aware Hybrid Multi-Cloud Analytics

2.1 Introduction

The emergence of *hybrid multi-cloud computing* – with a two-fold increase in deployments over last few years [39, 36, 28, 33, 27, 22, 38] – can potentially democratize the same for enterprise customers [8, 16, 10, 1, 19]. In this case, data resides in multiple local and remote *sites* among which applications run across – a site can be an on-premise (on-prem) cluster or an off-premise (off-prem) datacenter – connected by wide-area network (WAN) links to serve a globally distributed audience (Figure 2.1). Collecting data related to end-user sessions, monitoring logs, and performance counters, and thereafter analyzing and personalizing this data can significantly improve overall user experience.

Single-Site Big Data Analytics Analyzing large datasets in a timely and resource-efficient manner has been a primary focus for more than a decade now, with frameworks such as MapReduce [3, 69] and Spark [218] leading the way. They have contributed to democratizing big data analytics by hiding most of the complexity related to machine coordination and scheduling, data transfer, scalability, and fault tolerance. However, they all assume single-site deployments (i.e., machines are always in the same on-prem cluster or the same off-prem datacenter), where the bandwidth between machine pairs is high and often uniformly distributed.

Wide-Area Data Analytics When data is distributed across many sites, applying traditional big data analytics solutions require organizations to copy remote data to a central location and analyze it locally [202, 203, 204]. This consumes expensive WAN bandwidth proportional to the size of the input data, and sometimes may not even be possible due to regulatory constraints (e.g., European Union's General Data Protection Regulation) [175, 130, 14]. Recent works have

focused on developing specialized wide-area/geo-distributed data analytics solutions that take WAN bandwidth into consideration during query planning, job scheduling, and job execution across many datacenters [138, 202, 203, 204, 130, 175, 201, 105, 107]. While existing solutions lay the foundation for analytics over the WAN, they all assume simple network topologies where datacenters are directly connected and focus primarily on reducing the amount of data going in and out of each datacenter.

Challenges In case of *hybrid multi-cloud analytics (HMCA)*, not all sites have the same connectivity, nor are they all directly connected. Instead, they use a patchwork of inter-site links with diverse network characteristics (§2.2).

The heterogeneity in how remote sites are inter-connected raises several challenges (§2.3). First, for a single job, existing solutions attempt to minimize the total amount of data transfers from inter-site data shuffles, but they only use a single path out of multiple available ones. This leads to network underutilization, which, in turn, increases job completion times. Furthermore, existing solutions cannot effectively schedule inter-site shuffles from multiple concurrent jobs either. Finally, runtime dynamics such as bandwidth fluctuations and link failures require dynamic readjustments, but existing solutions optimize jobs only at the beginning of execution.

Our Approach In this project, we make the case for a principled approach toward hybrid multi-cloud analytics by presenting TERRA, which provides the same familiar interface of existing big data analytics and wide-area data analytics solutions. Unlike existing solutions, however, TERRA judiciously manages network bandwidth in a topology-aware manner by leveraging multiple paths between any two sites, improving the performance of data-parallel jobs corresponding to individual HMCA queries. As such, it complements and builds on top of existing wide-area data analytics solutions. TERRA also schedules large data transfers across multiple queries. In both cases, it dynamically adapts to runtime dynamics.

Implementing and enforcing these, especially for complex SQL analytics queries and large HMCA deployments, pose scalability challenges in two fronts:

1. *Efficient algorithm design:* Existing data analytics solutions only leverage a single path between two remote sites, leading to bandwidth underutilization. Recent multipath approaches from the networking community (e.g., MPTCP [178]) can utilize multiple paths, but they cannot efficiently schedule data transfers of HMCA jobs. Simultaneously determining which paths to

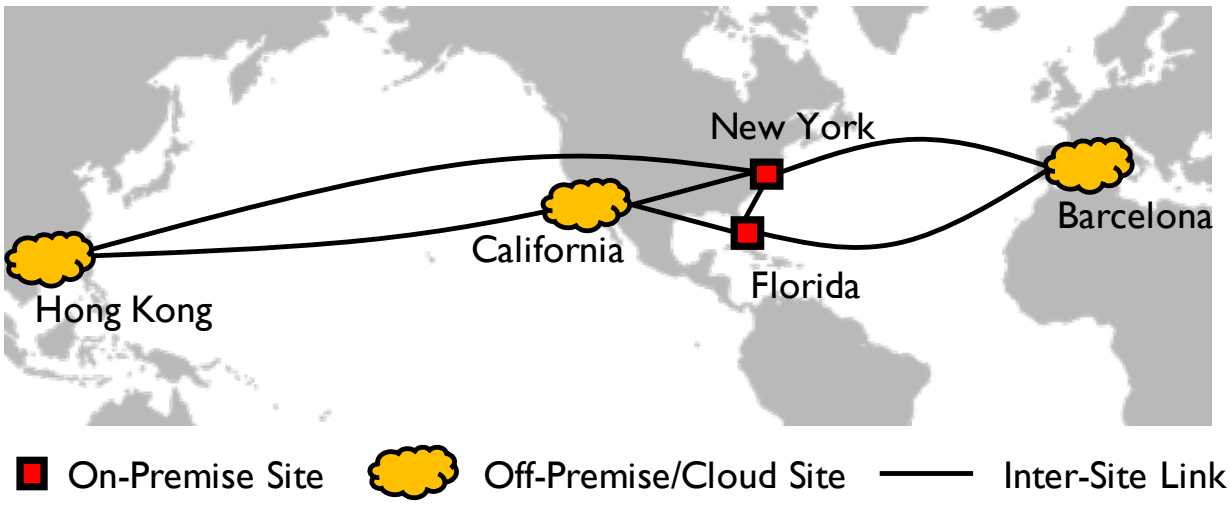


Figure 2.1: Data-parallel jobs in hybrid multi-cloud analytics run over data stored in on-prem and off-prem sites connected by heterogeneous network links.

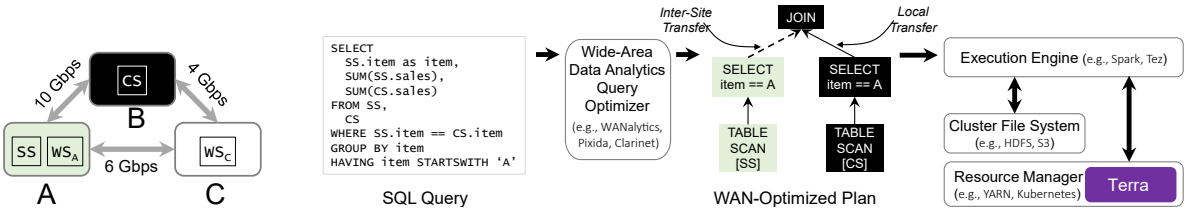
take (*routing*), which job's data to send (*scheduling*), and what rate to send data at (*rate limiting*) in an efficient and timely manner is an NP-hard problem.

2. *Dynamic system design and implementation:* Even with an efficient algorithmic solution, designing and implementing it in a system requires global knowledge and the ability to quickly enforce updates across many inter-site links to dynamically adjust according to online job arrivals/departures and runtime dynamics.

TERRA overcomes these challenges by taking an algorithm-system co-design approach that can scale to large HMCA jobs and WAN topologies.

Contributions We make the following contributions:

1. We propose a scalable algorithm that optimizes inter-site shuffles from complex HMCA jobs on large WAN topologies connecting many on-prem and off-prem sites. We formulate it as an optimization problem that minimizes the time of transferring a WAN traffic matrix, and formally prove its optimality in the offline case. Using it as a building block, we provide support for online scheduling of multiple HMCA jobs (§2.4).
2. We implement our algorithms in a scalable system (TERRA) that integrates with and complements existing frameworks (anything that can run on Apache YARN) so that unmodified user-written jobs can transparently run on hybrid multi-cloud environments. TERRA enforces our algorithm-determined data transfer plan, i.e. routes, schedules, and rates, in an efficient



(a) Example topology (b) Life cycle of an HMCA job

Figure 2.2: Overview of the HMCA stack with the proposed addition of TERRA.

manner and can quickly detect and react to runtime dynamics. It does not require any changes to the network either (§2.5).

3. We evaluate TERRA with WAN topologies from three major companies and three industry-standard benchmarks (TPCx-BB, TPC-DS, and TPC-H) and show that it outperforms the state-of-the-art by up to $3.43\times$ for the smallest and up to $32.04\times$ for the largest deployment. We further break down the performance benefits of its components. We also show that TERRA reacts quickly to WAN events; it scales well, and its benefits hold across a wide range of parameter and environment settings (§2.6).

   We compare TERRA with related work in Section 2.7.

2.2 Background

2.2.1 Hybrid Multi-Cloud Environment

Local, Cloud, and Edge Sites The core idea of hybrid multi-cloud computing is to pool together resources across different premises. This is required for addressing a variety of challenges:

1. enabling a global footprint and avoiding vendor lock-in [40];
2. moving data across sites is prohibitive because network bandwidth is limited (e.g., cannot support large-scale HD video streaming for video analytics [122, 104]) and expensive (up to $0.13/GB [13]); and
3. policy constraints such as General Data Protection Regulation (GDPR) [14] limiting the transfer of raw data across regions.

Each site consists of a number of compute machines that are often connected by a high bandwidth

intra-site local-area network (LAN). The amount of compute resources and their capability can vary between sites; e.g., edge sites are likely to be smaller whereas a cloud site will likely be larger. Different sites are connected by WAN links to form a hybrid multi-cloud computing environment.

Inter-Site Network Characteristics On-prem and off-prem sites in a hybrid multi-cloud environment are connected by a variety of inter-site links:

1. on-prem and off-prem sites are connected by off-prem provider solutions (e.g., AWS Direct Connect [7] for Amazon EC2 and Azure ExpressRoute [9] for Azure);
2. off-prem and off-prem sites often use the cloud provider's private software-defined WAN (SD-WAN) [101, 113] when they belong to the same cloud provider and over the public Internet for different providers; and
3. on-prem to on-prem connections go through the public Internet or privately leased WAN tunnels.

Bandwidth and latency of individual links vary according to the link type, geographical locations of sites, and the number of cores in individual (virtual) machines, with bandwidth varying from 468Mbps to 4.9Gbps and latency from 1ms to 400ms [138, 139]. For a given link, WAN bandwidth typically varies every few minutes [101, 113]. The bandwidth, latency, and availability may also change due to failures and re-configurations. In short, the inter-site network is *heterogeneous* and *dynamic*.

2.2.2 Hybrid Multi-Cloud Analytics

Executing applications across multiple on-prem and off-prem sites naturally produces data that are globally distributed, making sense of which can provide great value. We refer to analytics running in such an environment as hybrid multi-cloud analytics (HMCA).

Life Cycle of an HMCA Job Figure 2.2 provides a quick overview of a simple SQL query's life cycle in HMCA. Users submit analytics queries written in familiar higher-level interfaces (e.g., SparkSQL [49] and Hive-QL [199]) typically to one central *job master* that resides in one of the many distributed sites [203, 204, 202, 175, 130] – this example has three sites. A WAN-aware query optimizer – Clarinet [201] in this example – picks an ideal query plan to minimize the amount of WAN transfer – for example, rows from table `SS` are to be transferred to site B, while rows from `CS` will be shuffled over site B's LAN. Typically, shuffles and joins lead to expensive inter-site transfers,

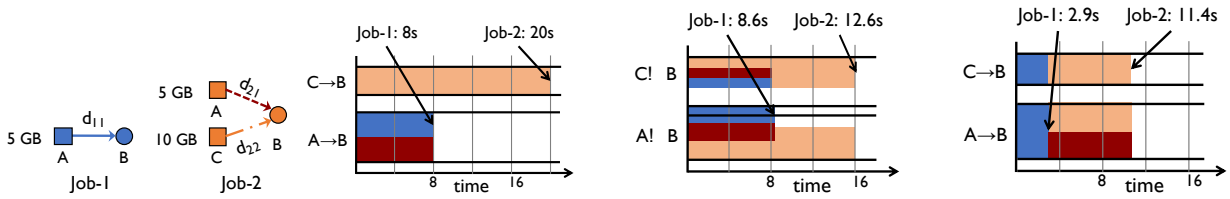
while tasks with one-to-one communication between them are executed on the same site to avoid costly WAN transfers [130]. This WAN-aware query plan is then converted into a physical execution plan and represented by a directed acyclic graph (DAG). Nodes of this DAG represent computation stages with many parallel tasks and edges represent stage dependencies and data transfers between tasks in different sites. The master then launches the corresponding data-parallel job and its tasks in machines across different sites based on data locality, resource availability, and their dependencies [108, 203, 204, 202, 201].

WAN Transfers of an HMCA Job As stages complete, the join must take place in site B as decided by the query optimizer, and an inter-site shuffle is performed by the data transfer layer of the framework. Indeed, a large amount of data must be transferred across sites even after significant reductions in the total data volumes by high-quality query planning. This is because the total amount of data we collect and analyze is increasing rapidly – for example, Alibaba reports processing exabytes of data in 2019 [105]. Even a $10\times$ reduction as claimed by WANalytics [203, 204], Pixida [130], or Clarinet [201] would leave tens to hundreds of terabytes of data to be transferred across many analytics queries.

Existing wide-area data analytics stacks still rely on communication layers of traditional big data analytics stacks with the high-bandwidth assumption. Because of the high bandwidth, the inefficiencies of decentralized communication is not a big issue inside a datacenter. However, it leads to significant inefficiency in an HMCA setting (Section 2.3 has illustrative examples). These solutions do not provide any scheduling mechanisms either. Finally, data reductions in wide-area data analytics query optimizers is a one-shot operation, which cannot react to runtime dynamics.

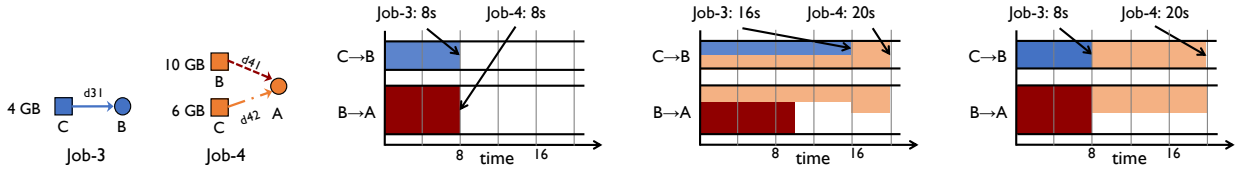
2.2.3 TERRA in the HMCA Stack

We propose TERRA to be a drop-in component of existing wide-area data analytics stacks to optimize hybrid multi-cloud SQL analytics. It is topology-aware, and it can determine schedules, routes, and rates of inter-site shuffles of many jobs simultaneously. It is transparent to the users too – unmodified jobs from Hive- and Spark-based wide-area data analytics frameworks such as WANalytics and Pixida can take advantage of TERRA by using TERRA's ShuffleHandler integration with Apache Hadoop YARN (Figure 2.2(b)). In short, TERRA is an application-aware WAN optimizer/adapter for wide-area data analytics applications.



(a) WAN transfers of two jobs (b) Default (c) w/ MPTCP (d) TERRA

Figure 2.3: Opportunities for HMCA optimization for two jobs running on the scenario shown in Figure 2.2(a). (a) Job-1 (dark/blue) has one transfer and Job-2 (light/orange) has two (dark/red and light/orange). (b)–(d) Bandwidth allocations of the two bottleneck links ($A \rightarrow B$ and $C \rightarrow B$) w.r.t. time for three policies. Average completion times for (b) default settings in existing frameworks is 14 seconds; (c) using MPTCP instead of TCP is 10.6 seconds; and (d) TERRA finds the optimal solution in this case, which is 7.15 seconds.



(a) WAN transfers of two jobs (b) Optimal (c) Suboptimal upon failure (d) Optimal after failure

Figure 2.4: Need for re-optimization in HMCA. (a) On Figure 2.2(a)'s topology, Job-3 (dark/blue) has 1 transfer and Job-4 (light/orange) has 2 (dark/red and light/orange). Average completion times (b) for the optimal solution is 8 seconds; (c) after rerouting d_{42} due to the failure of the A–C link is 18 seconds; (d) for the new optimal solution after the failure is 14 seconds.

Scope and Assumptions TERRA is applicable to any SQL analytics query that existing Hive- and Spark-based wide-area data analytics frameworks can execute. It is invoked only when there is an inter-site WAN transfer – primarily due to shuffle/join; existing wide-area data analytics query optimizers will mitigate the rest – in an HMCA query; otherwise, it remains transparent. As such, TERRA complements existing frameworks and assumes their presence for query optimization and job execution.

2.3 Design Challenges

Despite reducing the amount of WAN transfers, existing wide-area data analytics frameworks face three key challenges when applied to hybrid multi-cloud environments:

1. Bandwidth underutilization from not leveraging multiple paths;

2. Performance loss due to lack of global knowledge and scheduling; and
3. Suboptimal decisions under runtime dynamics.

We highlight them using simple illustrative examples below. Note that the benefits are much larger for complex SQL queries in TPC-DS, TPC-H, and TPCx-BB benchmarks (§2.6).

Example Scenario Consider the hybrid multi-cloud environment spanning three sites $\{A, B, C\}$ shown in Figure 2.2(a) with the three tables: SS and CS are located in sites A and B, respectively, and WS is partitioned across two sites A and C. Now consider two queries: Query-1 involves a join between SS and CS (as shown in Figure 2.2(b)), while Query-2 involves a join between WS and CS. After applying query optimization techniques such as predicate push down, partition pruning, and minimizing silos [130, 201], two data-parallel HMCA jobs (Job-1 and Job-2) are produced corresponding to the two queries. For Job-1, the selected rows of SS are transferred to site B ($|\sigma(SS)| \ll |\sigma(CS)|$), where the reducers are located and the join will take place; similarly, for Job-2, both shards of WS are transferred to B ($|\sigma(WS_i)| \ll |\sigma(CS)|$; $i = \{A, C\}$). Sizes of corresponding intermediate outputs that must be transferred between sites are shown in Figure 2.3a.

Existing Solutions Let us now consider what would happen in frameworks that use the default single-path TCP for inter-site shuffles. First, because TCP can use only a single path – likely the shortest one given typical routing protocols – the link between A and C will remain unused. Second, because it equally divides a link, the bandwidth of the $A \rightarrow B$ link will be evenly divided between transfers for d_{11} and d_{21} (Figure 2.3b). Thus, both transfers complete by 8 seconds, whereas d_{22} completes by 20 seconds with no contention on $C \rightarrow B$. Hence, the joins for Job-1 and Job-2 can start after 8 and 20 seconds, respectively.

A reasonable improvement would be using multiple paths (e.g., MPTCP [76]) to increase network utilization (Figure 2.3c). In this case, all data transfers are split across the available paths. Assuming equal split and fair sharing in each link, even though Job-2's data transfer finishes faster (12.6 seconds), Job-1's becomes slightly slower (8.6 seconds). While this provides small improvement for this example, we observe that not all links are in use all the time; moreover, giving equal priority to all transfers can hurt performance. Worse, we found existing MPTCP implementations to perform even poorer than TCP for inter-site shuffles in experiments (§2.6.2).

Finally, because both the alternatives make local decisions, they cannot optimize global, job-level objectives.

Co-Optimization Improves Performance So far we have shown two sub-optimal solutions, where they optimize only the networking aspect of HMCA. However, by using global knowledge, if we judiciously decide scheduling of jobs, routing of transfers, and their rates, we can achieve the optimal average completion time of only 7.15 seconds (Figure 2.3d) and both jobs improve.

Re-Optimization is Necessary Under Dynamics Now consider the same topology as in Figure 2.2(a) but with two different jobs (Figure 2.4a). Existing frameworks (e.g., Clarinet) will schedule Job-3 and Job-4 together to finish each in 8 seconds. However, if the link between A and C fails (or experiences low bandwidth) right after the decision, the network will reroute d_{42} and the completion times of Job-3 and Job-4 would become 16 and 20 seconds, respectively.

The optimal solution is dynamically rescheduling Job-3 before Job-4 so that they complete in 8 and 20 seconds.

Note that changes in the data transfer requirements of the jobs themselves can also lead to runtime dynamics and need subsequent re-optimization.

2.4 TERRA Algorithm Design

TERRA schedules, routes, and rate limits data transfers for HMCA jobs over the WAN topology connecting on- and off-prem sites. This requires designing both a *scalable algorithmic solution* that can quickly make joint scheduling and multipath routing decisions, and a *scalable system* that can quickly enforce those decisions across the WAN and dynamically react to runtime variations. In this section, we focus on the former. Section 2.5 discusses the latter.

2.4.1 Preliminaries

The key building block of TERRA is an optimal algorithm for inter-site shuffle, which is similar to a traditional data shuffle in data analytics frameworks [69, 218] except that the source and destination machines are located in far-apart sites.

The WAN itself is a directed graph $G = (S, L)$, where S is the set of remote sites and L is the set of WAN links connecting those sites – e.g., Figure 2.1 has five sites and seven links. We represent multiple physical links between sites u and v ($u, v \in S$) with one logical link, which corresponds to edges $(u, v), (v, u) \in L$ with the cumulative bandwidth for each direction. At time

instant T, the available bandwidth for (u, v) is $c_T(u, v)$.

For each inter-site shuffle i, TERRA algorithm takes a data matrix $\mathbf{D}_i = [d_i(u, v)]_{|M|\times|M|}$ corresponding to an inter-site shuffle from the job framework, where M is the set of all machines across all sites. Each entry $d_i(u, v)$ refers to the amount of data of inter-site shuffle i to be transferred from machines u to v in two sites. Because each site can involve tens to thousands of tasks/machines [105], $|M|$ is large and $\mathbf{D}_i$ can have huge cardinality. Given the input, TERRA determines which WAN paths to use to transfer each piece of data in $\mathbf{D}_i$ and at what rate. Note that a data-parallel job can submit multiple traffic matrices.

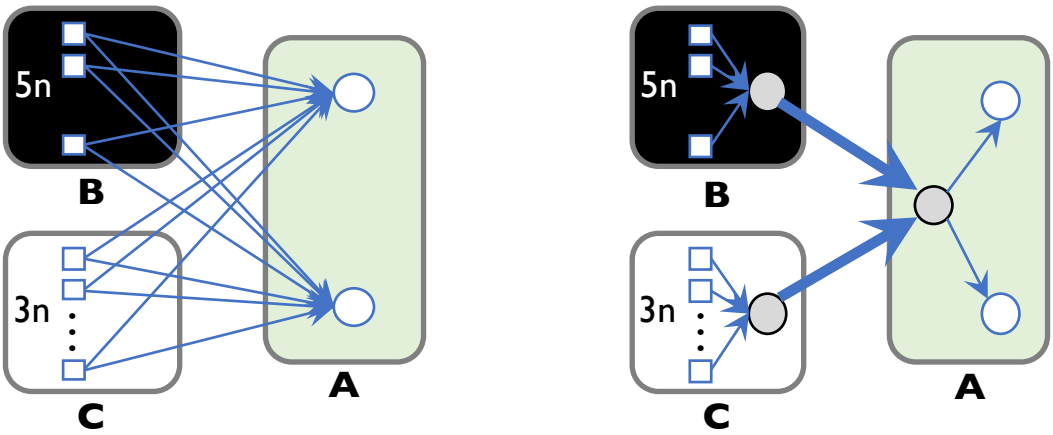
2.4.2 Single-Query Optimization

We start by focusing on a single inter-site shuffle and decide *which path(s) to take* for each of its transfers and *at what rate* to send the traffic. To this end, we propose the DataBundle abstraction to scale down the problem (§2.4.2.1) and use it to design an optimal algorithm (§2.4.2.2). Using this as a building block, we design a single-query optimization mechanism (§2.4.2.3) as well as a multi-query scheduling algorithm that schedules traffic matrices from concurrently running HMCA jobs, considering both offline and online scenarios (§2.4.3).

Scalability Limitation Theoretical works, e.g., a recent 2-appro-ximation algorithm [64], calculate the rate and determine the paths for *every* single machine pairs involved in an inter-site shuffle. This can be impractical due to the huge cardinality of the traffic matrix. In fact, these works spend tens of seconds on calculation, even in a relatively small HMCA deployment with only five sites (Figure 2.1). This overhead, which further increases for larger topologies, renders the approach infeasible for an online system, as scheduling will be triggered frequently by events such as submission of new jobs and the fluctuation of link capacities.

Collective Optimization Our primary insight is that *individual rate allocation of data transfers between machine pairs is unnecessary*. This is because we do not care about when individual transfer of a shuffle finishes, but when the very last one does.

Consider a simple shuffle on the same topology shown in Figure 2.2(a). Assume there are $5n$ map tasks placed in B, $3n$ map tasks placed in C and 2 reduce tasks placed in A (Figure 2.5a). So there are a total of $16n$ machine-to-machine data transfers (also known as network flows). Suppose for each flow we need to send 1 GB data. Enforcing *all flows to finish together* – as prescribed



(a) Old: $16n$ inter-site transfers (b) New: Two DataBundles

Figure 2.5: Scaling down the inter-site shuffle problem instance using the DataBundle abstraction.

in datacenter shuffle transfer optimization works [65] – gives all flows $\frac{1}{n}$ Gbps bandwidth. This allocation gives a minimal shuffle completion time of $8n$ seconds, and both link $B \rightarrow A$ and $C \rightarrow A$ are fully utilized all the time. Instead, if we take all the flows traversing through link $B \rightarrow A$ and, for example, schedule them one-by-one in any FIFO order without any gap in between, we can still achieve the same completion time of $8n$ seconds.

2.4.2.1 The DataBundle Abstraction

Given the observation, we introduce the *DataBundle* abstraction that groups individual machine-to-machine transfers between the same pair of sites as a single, collective site-to-site transfer. *The rates of individual flows within such a group do not directly affect the shuffle completion time, because the difference comes from when the last one finishes. All these flows have the same source and destination, and we only need to care about the total rate of the group.* This grouping is complementary to the Silo abstraction for grouping tasks introduced in Pixida [130] and similar to that of FlowGroup [134].[1]

Introducing DataBundles brings performance improvements in both calculating and enforcing bandwidth allocation. Because we only need per-DataBundle allocation, the scale of our problem formulation is reduced by orders-of-magnitude from $O(|M| \times |M|)$ to $O(|S| \times |S|)$. For example, in Figure 2.5b, $16n$ inter-site transfers are replaced by only two inter-site transfers and $8n + 2$ local

[1]A Silo is a collection of computation tasks in a site, and a FlowGroup is a collection of flows between two sites; DataBundle bridges the gap in that a Silo can produce a DataBundle that can lead to a FlowGroup.

transfers. This significantly lowers our scheduling overhead (§2.6.5).

2.4.2.2 Optimal Inter-Site Shuffle Algorithm

We can now formulate an optimization problem to minimize the completion time of a *single* inter-site shuffle on a general network topology. We assume that a DataBundle can be split across many paths, which (i) enables multipath routing and (ii) leads to a linear program (LP) formulation instead of a more computationally intensive integer linear program (ILP) formulation.

The algorithm comes in two steps:

1. Scale down traffic matrices using DataBundle; and
2. Solve a linear program that minimizes overall completion time to obtain a set of paths with fractional rates for each DataBundle.

Step 1: Scaling Down Using DataBundles In this step, we collapse all individual transfers from the *same* shuffle with the same (src_site, dst_site) pair to one DataBundle. We can now represent an inter-site shuffle i as a collection of DataBundles $\mathbf{B}_i = [b_i(u,v)]_{|S|\times|S|}$, where $|b_i(u,v)|$ represents the total amount of WAN transfers of this shuffle from machines in site u to machines in site v.

Step 2: Finding Paths and Rates We now determine the *paths* and *rates* of individual inter-site DataBundles of the i-th inter-site shuffle to minimize its completion time (Γ_i) in the offline case. Recall that we are dealing with a *single* query. Here Γ_i, the completion time of query i, is defined as:

$$\Gamma_i = \max_k T(b_i^k), \quad b_i^k \in \mathbf{B}_i,$$

where b_i^k is the k-th DataBundle, and $T(\cdot)$ is the completion time of a DataBundle (0 when the DataBundle is empty). Hence, the slowest DataBundle $b_i^k \in \mathbf{B}_i$ determines Γ_i. Our objective is given as:

$$\text{Minimize } \Gamma_i. \tag{2.1}$$

Let $f_i^k(u,v)$ be the load on edge (u,v) for DataBundle b_i^k. To minimize Γ_i, we enforce *equal rate of progress* for all DataBundles. In other words, for each DataBundle, we ensure that it makes $\frac{1}{\Gamma_i}$ progress every time unit. Flow conservation and capacity constraints are enforced with the following linear constraints.

$$\sum_{u \in S} f_i^k(u,v) - \sum_{w \in S} f_i^k(v,w) = \begin{cases} |b_i^k|, \ v = \text{dst}(b_i^k) \\ -|b_i^k|, \ v = \text{src}(b_i^k) \\ 0, \ \text{otherwise} \end{cases} \quad \forall v \in V \tag{2.2}$$

$$\sum_k f_i^k(u,v) \leq \Gamma_i \cdot c_T(u,v) \quad \forall u, v \in S \tag{2.3}$$

$$f_i^k(u,v) \geq 0 \quad \forall u, v \in S, \forall k$$

Constraint (2.2) ensures flow conservation, while Constraint (2.3) captures the total bandwidth available for the total time horizon of Γ_i from time instant T.

For a certain LP solution, we schedule DataBundle k at a rate of $\frac{f_i^k(u,v)}{\Gamma}$ on link (u,v) – this ensures that the rate of a DataBundle from its source and to its destination is proportional to its volume. More importantly, this reserves the maximum amount of bandwidth possible for other shuffles that are considered later without sacrificing its own completion time, which is beneficial when dealing with multiple inter-site shuffles (§2.4.3).

If Optimization (2.1) has a feasible solution (it will always have a solution unless certain DataBundle cannot be routed at all), it creates a matrix $\mathbf{f}_i^k = [f_i^k(u,v)]_{|S| \times |S|}$ for each DataBundle corresponding to its allocations on individual links of the WAN. Because $f_i^k(u,v)$ can be fractional, a DataBundle can be subdivided across many paths from u to v. We enforce this in our system design (§2.5).

Optimality Formally, we prove the following.

Theorem 1. Optimization 2.1 optimizes the minimum Γ_i.

Proof. First, we prove that Optimization 2.1 can give a valid solution that finishes every DataBundle by time Γ_i. Constraint (2.3) ensures that the total bandwidth usage on link (u,v) is $\sum_k \frac{f_i^k(u,v)}{\Gamma_i} \leq c_T(u,v)$. Flow conservation is achieved similarly. The total transmission for DataBundle k is the following:

$$\Gamma_i \cdot \left(\sum_{u \in S} \frac{f_i^k(u, \text{src}(b_i^k))}{\Gamma_i} - \sum_{w \in S} \frac{f_i^k(\text{src}(b_i^k), w)}{\Gamma_i} \right) = |b_i^k|.$$

Now, consider any schedule $\hat{S}$ (even if we do not use DataBundles); we prove that it needs Γ_i

Input: Inter-Site Shuffles $\mathbb{S}$, WAN G
Output: Bandwidth Allocation

Procedure allocBandwidth ()
  Scale down G by $(1-\alpha)$ ▷ Scale down network bandwidth to leave headroom.
  $\mathbb{S}_{\text{Failed}} = \emptyset$ ▷ Shuffles that cannot be scheduled.
  forall $\mathbf{B}_i \in \mathbb{S}$ **do**
    $\Gamma_i, \mathbf{f}_i^k$ = Solve Optimization (2.1) for $\mathbf{B}_i$ on G
    if $\Gamma_i = -1$ **then** ▷ C_i cannot be scheduled completely
      $\mathbb{S}_{\text{Failed}} = \mathbb{S}_{\text{Failed}} \bigcup \mathbf{B}_i$
      continue
    $\mathbb{P}_i^k$ = {End-to-end paths from $\mathbf{f}_i^k$ allocations}
    G = Updated G by subtracting $\mathbf{f}_i^k$ allocations
  $\mathbf{B}^* = \bigcup \mathbf{B}$ for all $\mathbf{B} \in \mathbb{S}_{\text{Failed}}$
  Allocate $\mathbf{B}^*$ on G using MCF ▷ Work conservation
  Allocate $\mathbb{S} \setminus \mathbf{B}^*$ on G using MCF
Procedure minimizeCCTOffline (*Inter-Site Shuffles* $\mathbb{S}$)
  $\mathbb{S}'$ = Sort $\mathbb{S}$ by increasing Γ_i
  allocBandwidth($\mathbb{S}'$, G)

Algorithm 1: Offline Multi-Query Optimization

amount of time to transfer all data in that shuffle. In fact, let $\hat{f}_i^k(u,v)$ denote the total amount of transmission through link (u,v) for demands that originate from src(b_i^k) and go to dst(b_i^k); then clearly $\hat{f}_i^k(u,v)$ satisfies Constraint (2.2). Suppose this schedule finishes in time $\hat{\Gamma}_i$, then the total data transmission on link (u,v) is bounded by $\hat{\Gamma}_i \cdot c_T(u,v)$, which satisfies Constraint (2.3). Therefore, $\hat{S}$ corresponds to a solution to the LP above. According to the optimality of the LP solution, Γ_i is as good as $\hat{\Gamma}_i$. □

2.4.2.3 Support of Pipelined Workloads and Dynamic Traffix Matrices

Many data analytics jobs form DAGs of computation stages with communication stages in between [5, 110, 218, 4, 49]. Job masters can submit requests for each shuffle in a DAG to TERRA as dependencies are met.

Job masters can also submit a partial traffic matrix and update it over time as more data becomes available. This is useful when the preceding computation tasks do not finish at the same time. In

this case, TERRA tries to finish all the submitted transfers together, eventually finishing the entire traffic matrix. Our evaluation shows that this simple strategy performs well (§2.6).

2.4.3 Multi-Query Optimization

In this section, we focus on scheduling inter-site shuffles ($\mathbb{S} = \{\mathbf{B}_i\}$ after converting to DataBundles) from multiple concurrent queries. Unlike a single shuffle, scheduling multiple shuffles in an optimal manner is NP-hard even when all of them start together, their traffic matrices are known a priori, and the WAN has $|S|^2$ links directly connecting all S sites [64]. A naive approach that treats all shuffles as one would not work, as it gives the same completion time for all shuffles, though some of them could be finished earlier. Given the intractability of the problem, we focus on designing an efficient offline heuristic; we then extend it to online scenarios.

2.4.3.1 Offline Scheduling

Given multiple ongoing inter-site shuffles over the same shared WAN topology, scheduling one can impact the completion times of all others that are scheduled afterwards. Consequently, a natural extension of the shortest-remaining-time-first (SRTF) policy is sorting them by their Γ values and scheduling them in that order (MINIMIZECCTOFFLINE in Pseudocode 1). This requires solving $O(N)$ instances of Optimization (2.1) during each scheduling round, which is activated by a job arrival, completion, and runtime WAN events. We schedule a shuffle if all of its DataBundles can be scheduled simultaneously.

Ensuring High Utilization In some cases, the WAN may still have under-utilized links after scheduling all shuffles that can be scheduled in their entirety. We then try to allocate the remaining bandwidth to $\mathbb{S}_{\text{Failed}}$ by running a max-min multi-commodity flow (MCF) formulation (line 12,13 in Pseudocode 1). This ensures work conservation (i.e., all available bandwidth has been allocated) and maximizes WAN utilization.

2.4.3.2 From Offline to Online

So far, we have assumed that all the shuffles arrive together and are known a priori. However, in practice, they arrive over time as DAG dependencies are met in each of the HMCA DAGs. Additionally, WAN links can fail and its bandwidth can fluctuate. Scheduling them in the FIFO

order is a simple solution, but it can result in head-of-line blocking – i.e., a large transfer can slow down many smaller ones behind it.

Preemption is a well-known technique to minimize head-of-line blocking. However, it can lead to starvation – a steady stream of small shuffles can keep preempting a larger inter-site shuffle and hurt a large job's completion time.

Starvation-Free Preemption We allow inter-site shuffles with smaller expected remaining completion times to preempt larger ones to avoid head-of-line blocking. Whenever a new shuffle arrives, we sort all unfinished shuffles by their expected remaining completion time, and prioritize scheduling the smaller ones. To avoid starvation, we guarantee each one to receive some share of the network. Specifically, α fraction of the WAN capacity is left as headroom and shared between preempted shuffles (line 2 in Pseudocode 1). By default, $\alpha = 0.1$.

Scalable Online Scheduling We use the offline algorithm as a building block for our online algorithm. For each round of online scheduling, we run the offline optimization based on the most up-to-date information assuming no knowledge of the future. Therefore, re-optimization is needed whenever this information is outdated due to the events such as the following:

1. New shuffles being submitted as dependencies are met;
2. DataBundle finishes;
3. Shuffle finishes because all its DataBundles finish;
4. Runtime WAN topology changes due to bandwidth fluctuations/failures.

Running the offline algorithm upon every event has high complexity. TERRA avoids this by aggregating multiple events and then rerunning for all of them together. For WAN bandwidth fluctuations, we consider a change of 25% of the initial bandwidth to be the threshold for significant bandwidth change that causes a rescheduling, filtering out short-term perturbations [101].

2.5 TERRA System Design

So far we have focused on designing a scalable algorithm for optimizing data transfers from inter-site shuffles of HMCA jobs (§2.4). In this section, we discuss how to implement the solution in a scalable manner at the individual machine level. Furthermore, we consider how to make it robust

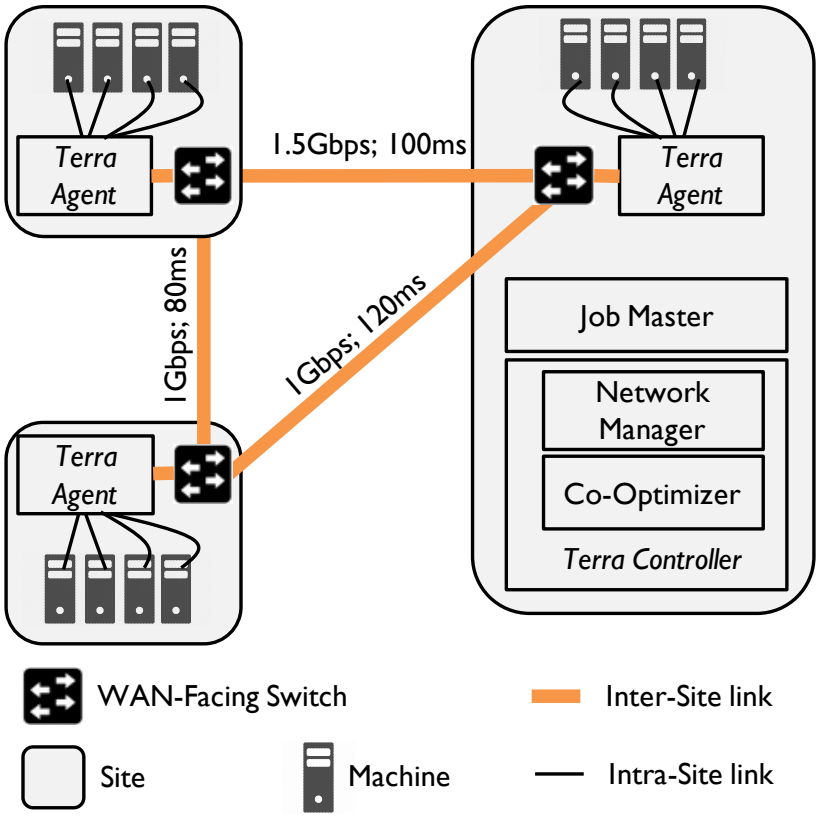


Figure 2.6: TERRA architecture. HMCA jobs interact with TERRA using a client library. TERRA controller performs joint scheduling, path selection, and rate determination. Its decisions are enforces by TERRA agents that also decouple inter-site data transfers from intra-site transfers.

to runtime WAN variabilities. We start with an architectural overview of the whole system and then provide insights into designing individual components. We conclude with details on TERRA's implementation and integration with Apache YARN.

2.5.1 Architectural Overview

TERRA has two primary components (Figure 2.6). A logically centralized TERRA *controller* orchestrates all data transfers. In each site, a set of TERRA *agents* coordinate with the controller and transfer data on behalf of the jobs. Interactions between them can be summarized as follows:

1. Job masters submit traffic matrices corresponding to inter-site shuffles to the TERRA controller using the TERRA API. An HMCA job can submit multiple traffic matrices as their dependencies are met.

2. The controller maintains an up-to-date, global view of the WAN[2]and ongoing transfers. Given these, it computes which traffic matrices to schedule, which paths to use, and at what rates data should be sent via specific paths.
3. Finally, the controller sends path and rate information to corresponding TERRA agents, which enforce rate limiting across multiple paths between sites. Inside each site, the agent acts as a relay to the reducer machines and transfer data while taking into machine-specific bandwidth into account. As such, the agent decouples inter-site transfers from intra-site ones.

The entire process takes place in an online manner.

2.5.2 Need for Global Knowledge

In TERRA, all scheduling and routing decisions are made by its centralized master. This is because making such decisions without global knowledge can lead to *arbitrarily worse* performance [163, Theorem 5.1 on page 3].

Naturally, TERRA's centralized design brings scalability concerns to its design forefront both during its normal operations, where every arrival or departure of a job can cause rescheduling, and in the presence of dynamics such as WAN bandwidth fluctuations and failures that also require rescheduling. Each rescheduling round requires a central computation, followed by the dissemination and enforcement of decisions. We discussed the scalability improvements at the algorithmic level using DataBundles in Section 2.4, and we discuss the latter below.

2.5.3 Scalability

Minimizing Scheduling Overhead The total number of entries in each traffic matrix adds significant time complexity to an integer linear program-based solution. Consequently, TERRA compresses the problem and removes the integral constraints, leading to a practical solution (§2.4). Each scheduling round takes few 100s of milliseconds for smaller topologies with less than 10 sites, and a second or so for larger topologies with 10s of sites (§2.6.5). Given that many HMCA jobs will deal with massive datasets [105], TERRA is not time-constrained in decision-making and dissemination. We expect seconds-duration jobs to not use it at all because the overhead of managed

[2]The Network Manager interacts with the underlying network substrate, which is either an SD-WAN, when available, or an application-layer overlay.

transfers will not be worthwhile.[3] This is similar to how modern WANs do not throttle interactive services [101, 113].

Restricting the Number of Paths (k) Running an unconstrained MCF instance may result in path/route assignments that require many updates throughout the network. As such, TERRA provides operators the opportunity to constrain the maximum number of paths to consider between any pair of sites. Operators can set any (sets of) $f^k(u, v) = 0$ to enforce such constraints in TERRA (§2.6.6). The value of k also dictates how many network connections TERRA must maintain between agent pairs.

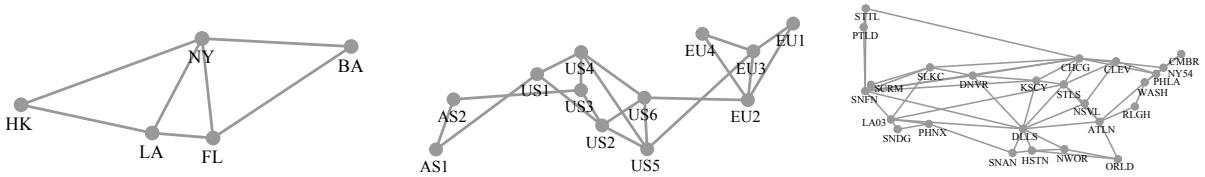
Minimizing Rule Updates in the WAN In the context of TERRA, an additional challenge is minimizing expensive rule updates throughout the entire WAN caused by route changes [101]. Instead of setting up new rules for each flow, TERRA maintains a set of single-path persistent flows between each pair of sites and sets up only one set of rules for each persistent flow. The controller sets up forwarding rules in the SD-WAN to enforce their paths. Directing a data transfer through a specific path is then simply reusing corresponding pre-established flows. To completely avoid expensive rule updates, TERRA reuses persistent connections for each of the k paths between two remote sites and performs communication on behalf of the applications (§2.5.5.2). WAN states change only when these flows are initialized or reestablished after failures. A collateral benefit of this approach is that TERRA uses only a small number of rules in each switch – e.g., up to 168 in a switch for the SWAN [101] topology in our evaluation.

2.5.4 Robustness to Dynamics

Failures of TERRA agents do not permanently affect job executions because frameworks can fall back to default transfer mechanisms. States in TERRA agents can be rebuilt upon restart when they contact the controller. Failure of the TERRA controller is tied to that of the framework controller. Its states can be rebuilt after all the TERRA agents and job masters reconnect.

TERRA is robust to WAN events such as link or switch failures and large bandwidth fluctuations from background traffic, because it monitors and reschedules according to the latest WAN state. Upon such events, TERRA updates its internal topology representation, recomputes the set of viable

[3] Big data jobs typically follow heavy-tailed distributions [201]; there are many small jobs, but footprint is not large enough to affect the rest.



(a) Microsoft (5 sites; 7 links) (b) Google (12 sites; 19 links) (c) AT&T (25 sites; 56 links)

Figure 2.7: Microsoft, Google and AT&T topologies used in evaluation.

paths between pairs of sites, and updates corresponding schedules. The entire process takes a small fraction of a typical HMCA job's duration (§2.6.4).

Because data transfers are decoupled from the jobs, we do not have to consider sender (e.g., Map) task failures. If a receiver is restarted in another machine of the same site upon failure, TERRA API supports updating destination(s) of a submitted traffic matrix.

2.5.5 Implementation Details

2.5.5.1 Framework Integration

We have implemented TERRA in about 10,000 lines of Java code including integrations with Apache Hadoop YARN [200]. It is open-source.

Essentially, TERRA substitutes the ShuffleHandler of Apache Hadoop. TERRA master runs in the same site as the Application Master of HMCA jobs. When Map tasks finish, instead of a shuffle request, the Application Master submits a traffic matrix request to the TERRA controller. TERRA informs the Application Master after the traffic matrix has completely been transferred.

2.5.5.2 Network Integration

WAN topologies have many redundant paths, and TERRA adopts multipath routing to minimize transfer times and maximize WAN utilization. Enforcing multipath rate allocations of TERRA's optimizer poses several practical challenges.

1. **Multipath Transfers Using Many Single-Path TCP Connections** The first challenge is implementing a multipath transport across the WAN. We note that existing MPTCP implementations [178] perform poorly when deployed in an HMCA setting (§2.6.2). So we build a multipath data transfer layer on top of single-path TCP. TERRA creates an application-layer overlay network

over the WAN using persistent connections between agents. For each path between each agent pair, one or more persistent TCP connections are maintained. All data transfers happen over these pre-established connections. We implement TERRA on top of an SD-WAN. The routing of these pre-established connections are enforced by the SD-WAN controller. During an offline initialization phase, the TERRA controller generates the corresponding forwarding rules for each TCP connection identified by the TCP 4-tuple, and installs these rules to the SDN switches via the SD-WAN controller.

When an SD-WAN is not available, the routing can still be enforced via static routing rules in the application layer overlay, such as using Linux `iptables`.

2. **Handling WAN Latency Heterogeneity** Emulating multipath transmission over individual TCP flows brings a new challenge. Because of heterogeneous latencies between pairs of sites, such multipath transmissions can incur many out-of-order data chunks on the receiver side. TERRA mitigates this by efficiently using the shuffle destination files in Hadoop – that are stored on disk – as large reordering buffers. As TERRA substitutes the ShuffleHandler of Apache Hadoop, it writes out-of-order data chunks directly into the destination file – each chunk is sent reliably over single-path TCP but there is no ordering requirement across two chunks taking different paths. It informs the application only after all the chunks in the entire shuffle destination file has been received.
3. **Rate Enforcement in TERRA Agent** An additional challenge is enforcing rate allocation for each DataBundle in the TERRA agent. TERRA maps a DataBundle across multiple end-to-end TCP flows, each with a unique combination of a sending agent, a receiving agent, its sending rate, and the path. When a DataBundle is scheduled to start, be preempted, or changes rate, the controller informs its sending agents, which transmit data through pre-established connections for each path at designated rates. TERRA agents perform rate limiting via the RateLimiter provided by the Guava library [17].

2.6 Evaluation

We evaluate TERRA on three WAN topologies and three benchmarks/industrial workloads in testbed experiments and large-scale simulations. Our key findings are as follows:

- TERRA outperforms the state-of-the-art by up to $3.43\times$ for the smallest and up to $32.04\times$ for the largest deployment. Median (95th) percentile improvements are up to $6.49\times$ ($2.86\times$) and $28.02\times$ ($13.40\times$) (§2.6.2).
- It improves WAN utilization by up to $1.76\times$ (§2.6.3).
- It is robust against real-time network failures (§2.6.4), its controller can scale to large topologies (§2.6.5), and it performs well under different parameter settings (§2.6.6).

2.6.1 Methodology

WAN Topologies We consider three HMCA deployment scenarios with real-world topologies (Table 2.1 and Figure 2.7):

Topology	Latency (ms)		Bandwidth (Gbps)	
	Min	Max	Min	Max
Microsoft [101]	40	200	5	10
Google [113]	30	200	10	100
AT&T [6]	10	100	10	100

Table 2.1: HMCA configurations used in our evaluation.

We use geographic distances to calculate link latencies and estimate link bandwidth according to the gravity model [182], resulting in heterogeneous bandwidths and latencies.

Workloads We use three industry standard benchmarks for data analytics – *TPC-DS*, *TPC-H*, and *TPCx-BB* [37]. In each run of the benchmarks, we choose jobs randomly from one of the corresponding benchmarks and apply the arrival rates from that in production traces [35], because the benchmarks do not have arrival distributions.

We vary the database scale factor of the queries from 40 to 100 – so that each job lasts from few minutes to tens of minutes – and create workloads with 400 jobs from the corresponding benchmark. The input tables are randomly spread across the different geographical regions, similar to prior studies [175, 201, 202]. Table 2.2 shows the average shuffle count of individual jobs in these benchmarks.

Baselines We compare TERRA against the following:

1. *Default*: Default TCP-based shuffle service in Apache YARN, which is used by most wide-area data analytics frameworks.

Workload	TPCx-BB	TPC-DS	TPC-H
Avg. shuffle count per job	5.49	4.53	5.01

Table 2.2: Average shuffle count of jobs in our TPC workloads.

2. *MPTCP*: We replace TCP in Default with MPTCP v0.94, and keep the default setting shipped with Linux 4.14.70.mptcp, with `net.mptcp.mptcp_path_manager` set to `fullmesh`.
3. *Siphon* [149]: A recent WAN traffic manager for wide-area data analytics frameworks.
4. [64]: A 2-approximation offline algorithm.

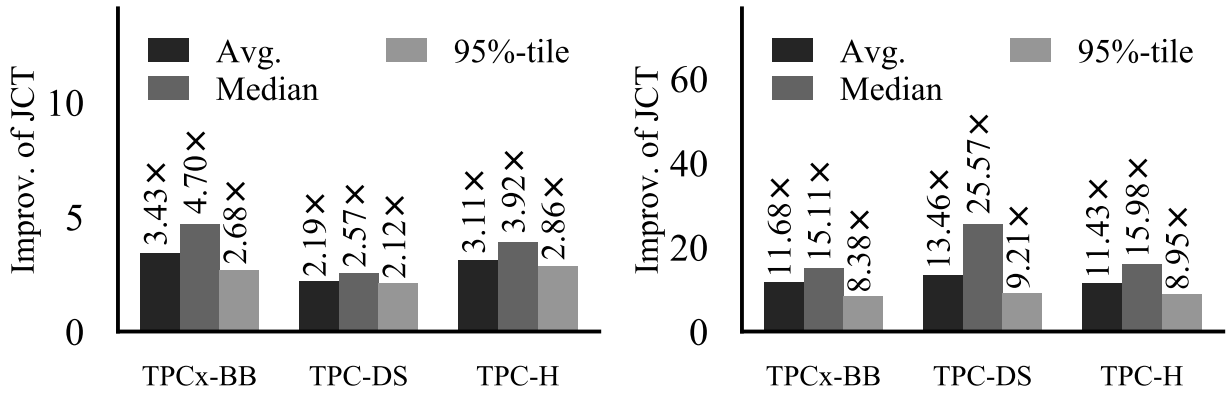
Jobs are executed by Apache Tez [5] after applying site selection/task placement-related single-query optimizations proposed in WANalytics, Pixida, and Clarinet to minimize WAN transfers.

Metrics Our primary metric to quantify performance improvement is *job completion time (JCT)*, and we use *average WAN utilization* to quantify the efficiency of the compared solutions.

Testbed Setup We build a testbed according to the Microsoft topology, with 10 machines in each site. TERRA controller runs on a host inside the site that minimizes the control message latency to all other sites. Each site has one switch, represented by a machine running one Open vSwitch instance. The switches are connected by VLANs, on top of a physical 40Gbps switch. The bandwidth and latency constraints are enforced by Linux Traffic Control (`tc`).

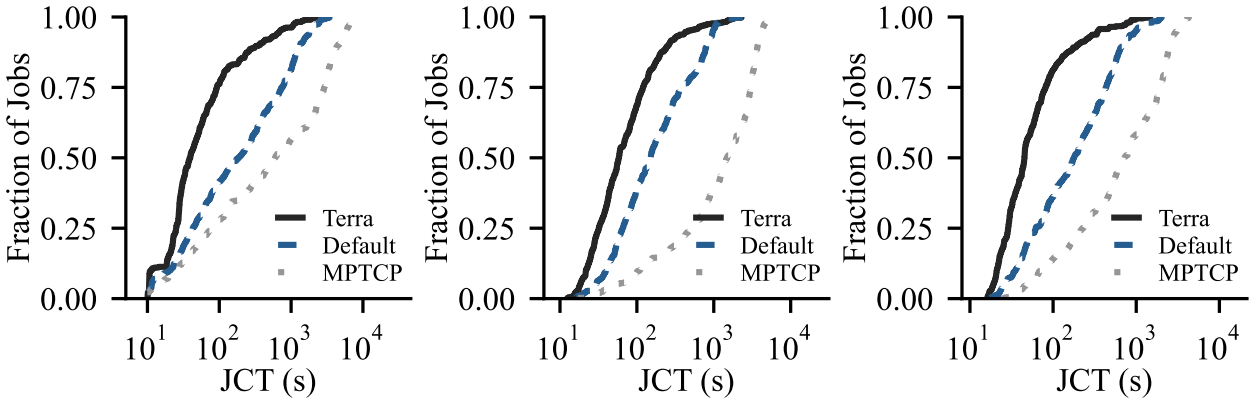
Simulator We conduct large-scale evaluations using a detailed flow-level simulator that implements the same logic as in the actual TERRA controller. We use the simulator to evaluate TERRA in larger deployment scenarios where cost of testbed experiments is prohibitive; more importantly, we use it to compare against solutions whose system implementations are not open-source. The simulator simulates an ideal network without considering packet-level behavior such as TCP slow-start and packet reordering, as packet-level simulation for such a large-scale network is infeasible. In the simulator we model the I/O and computation time in direct proportion to the size of shuffle data. In our simulation, we use a default of 200MB/s disk throughput per host.

We verify the fidelity of our simulator by confirming the job completion time (JCT) of TERRA with the testbed results. However, the simulator systematically overestimates the performance of the baselines such as Default and MPTCP, because it does not expose the issues of exacerbated packet reordering for these baselines. Hence the simulation shows a "lower bound" of performance gain



(a) w.r.t. Default (b) w.r.t. MPTCP

Figure 2.8: [Testbed] Improvement of job completion time (JCT) using TERRA w.r.t. Default and MPTCP on the Microsoft topology.



(a) TPCx-BB (b) TPC-DS (c) TPC-H

Figure 2.9: [Testbed] JCTs of individual jobs on the Microsoft topology.

using TERRA when compared to testbed experiments.

2.6.2 HMCA Performance Improvements

We evaluate TERRA on our testbed for Microsoft topology to examine its impact on JCT and WAN utilization, and we evaluate Google and AT&T topologies using simulation due to cost concerns.

	Topology	Microsoft			Google			AT&T		
	Workload	TPCx-BB	TPC-DS	TPC-H	TPCx-BB	TPC-DS	TPC-H	TPCx-BB	TPC-DS	TPC-H
Default	Avg.	1.78×	2.16×	2.53×	9.79×	12.45×	16.63×	8.64×	8.98×	9.57×
	Median	3.87×	3.36×	6.49×	9.23×	11.83×	28.02×	10.46×	12.65×	17.19×
	95%-tile	1.57×	1.63×	1.65×	8.82×	13.40×	11.93×	10.61×	7.41×	7.15×
MPTCP	Avg.	1.30×	1.25×	1.28×	2.77×	2.88×	3.24×	3.75×	3.03×	2.82×
	Median	2.88×	2.15×	3.02×	3.05×	3.03×	4.44×	4.44×	4.97×	5.07×
	95%-tile	1.15×	0.90×	0.86×	2.49×	3.51×	2.96×	5.24×	2.32×	2.26×
Siphon	Avg.	1.19×	1.34×	2.12×	7.30×	11.57×	23.38×	10.08×	16.64×	32.04×
	Median	1.45×	1.13×	1.47×	3.26×	3.45×	6.09×	9.87×	9.99×	12.59×
	95%-tile	1.24×	1.14×	2.25×	7.26×	21.03×	43.29×	11.88×	23.05×	35.07×

Table 2.3: [Simulation] Improvements in the average, median, and 95th percentile JCTs using TERRA w.r.t. baselines.

2.6.2.1 Improvements in the Microsoft Deployment

Figure 2.8 shows that TERRA improves the average JCT by at least 2.19× in comparison to Default (i.e., single-path, fixed-route TCP). At the 95th percentile, the improvement is at least 2.12×. Note that these numbers include the overheads of schedule computation, preemption, and rescheduling messages from the controller.

Figure 2.9 presents the CDFs of JCTs for all jobs across workloads. Except for the very short and very long jobs, TERRA's improvements are even larger as demonstrated by the median improvement of at least 2.57×. The former are dominated by other overheads outlined above, while the latter use all available bandwidth with little room for optimization.

Next, we evaluate the same scenario in Microsoft topology in the simulator to understand the fidelity of our simulator. We observe that w.r.t. Default, TERRA improves the average JCT by 1.78×–2.53× (Table 2.3), which is not far from the improvements observed in our testbed.

MPTCP We run MPTCP v0.94 using the default configuration and the same configuration for TCP sockets as in Default. Surprisingly, MPTCP performs even worse (Figure 2.9). The reason for this anomaly is high variations in end-to-end latencies between different paths in the Microsoft topology. This results in more out-of-order packets, which may require significantly larger buffers to be reordered before sending up the stack [68]. In contrast, TERRA writes out-of-order data chunks to the destination file and informs the application after the entire file has been transferred.

Overall, although TERRA's improvements over MPTCP is 1.25×–1.30× in simulation, its real-world improvement is much larger in experiments (11.43×–13.46×). This is because our conservative simulator does not consider the out-of-order packet problem.

Siphon Finally, we compare against a recent WAN optimization solution for wide-area data

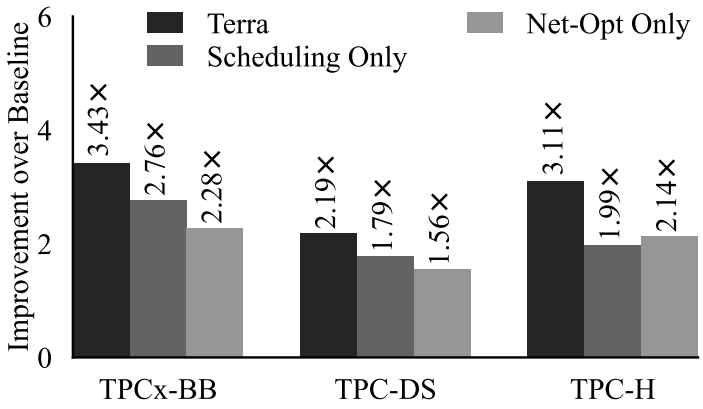


Figure 2.10: [Testbed] Improvement of average JCT w.r.t Default using TERRA, Scheduling Only, and Net-Opt Only.

analytics called Siphon [149]. We find that TERRA outperforms Siphon by $1.19\times$–$2.12\times$ on the Microsoft topology and up to $32.04\times$ for larger topologies. This is because (i) Siphon suffers from starvation and head-of-line blocking which is reflected in the high 95%-tile JCT, and (ii) it has high scheduling overhead due to its Monte Carlo simulation-based design (§2.6.5).

2.6.2.2 Performance Improvements Breakdown

We break down the performance improvements of TERRA by incrementally enabling the optimizations applied. Essentially, the optimizations in TERRA are two-fold: (i) application-aware scheduling and (ii) networking optimizations based on application layer multi-path routing. In Figure 2.10, we run experiments while independently enabling only the scheduling optimization (Scheduling Only) or only networking optimizations (Net-Opt Only), and calculate the improvement of average JCT. As can be seen, the performance improvements of TERRA comes from both the scheduling and networking optimizations, with each contributing almost equally in significance for TPC-H and scheduling contributing more in TPCx-BB and TPC-DS.

2.6.2.3 Improvements in Larger Topologies

TERRA's improvements increase as topologies become larger with more paths between any pair of sites. Across the baselines, TERRA improves the average JCT by $2.77\times$–$23.38\times$ in Google and $2.82\times$–$32.04\times$ in AT&T with similar median and 95th percentile improvements (Table 2.3).

Workload	TPCx-BB	TPC-DS	TPC-H
Improv. over Default	1.76×	1.49×	1.32×

Table 2.4: [Testbed] WAN utilization improvement using TERRA.

Workload	TPCx-BB	TPC-DS	TPC-H
Avg. Breakeven Duration (s)	11.75	15.32	14.84

Table 2.5: [Testbed] Average duration of jobs that run faster with Default than TERRA

2.6.2.4 Impact of Workloads

TERRA's improvements are evident across all workloads. To dig deeper, we calculate the Pearson's correlation coefficients (r) between TERRA's improvements w.r.t. the Default baseline and total WAN utilization for all ¡topology, workload¿ combinations. The result shows consistent negative correlations (-0.05 to -0.39) across the board, suggesting that smaller jobs see more benefits than the bigger ones due to the shortest-first nature of TERRA's scheduler.

2.6.2.5 Which Jobs Should Bypass TERRA?

Even though smaller jobs are favored by the scheduler, the smallest jobs should bypass TERRA to avoid the overhead of scheduling and orchestration. To find the watershed point, we filter out jobs that run faster with Default than TERRA and calculate their average duration, as shown in Table 2.5. This result suggests that jobs shorter than tens of seconds should not be submitted to TERRA.

2.6.2.6 How Far are We from the Optimal?

Because calculating the optimal solution for this problem is infeasible, we compare it against the recent 2-approximation offline algorithm [64] instead. We found that TERRA performs $1.14\times$–$1.16\times$ better than the 2-approximation algorithm.

2.6.3 WAN Utilization Improvements

As is shown in Table 2.4, TERRA improves the WAN utilization by at least $1.32\times$ w.r.t. Default. This improvement comes from the multi-path routing of TERRA. Note that the job-level JCT

Baseline	Topology	TPCx-BB	TPC-DS	TPC-H
MPTCP	Microsoft	1.12×	1.14×	1.06×
	Google	1.22×	1.14×	1.09×
	AT&T	1.42×	1.34×	1.76×
Default	Microsoft	1.14×	1.19×	1.06×
	Google	1.37×	1.16×	1.11×
	AT&T	1.53×	1.34×	1.77×
Siphon	Microsoft	1.16×	1.22×	1.21×
	Google	1.39×	1.20×	1.38×
	AT&T	1.56×	1.67×	2.53×

Table 2.6: [Simulation] WAN utilization improvement.

improvement (§2.6.2) is higher than the WAN utilization improvement, because the JCT has an additional boost from scheduling.

Table 2.6 shows the improvements in WAN utilization using TERRA w.r.t. all three baselines for all ¡topology, workload¿ combinations. TERRA achieved $1.06\times$ to $2.53\times$ higher utilization across all combinations. Note that similar to performance improvements, these simulated improvements are also a "lower bound", as the simulator does not capture packet-level behaviors.

2.6.4 Reactive Re-Optimization Upon Failure

We evaluate TERRA's robustness upon failure by a case study in our testbed. Figure 2.11 shows a scenario where one link (LA-NY) failed when there are two jobs $\{$Job 1, Job 2$\}$ running, and Figure 2.12 shows the bandwidth allocation change throughout this process. Note that we set $\alpha = 0$ for ease of exposition.

In this case, Job 1 has a smaller volume and thus higher priority than Job 2. TERRA reacts to the link failure within 10 seconds and preempts Job 2 (Figure 2.11b), minimizing the impact of link failure to Job 1's throughput. After the completion of Job 1, TERRA re-schedules Job 2 (Figure 2.11c). Finally, when the failed link is reinstated (Figure 2.11d), TERRA adds a new path for Job 2.

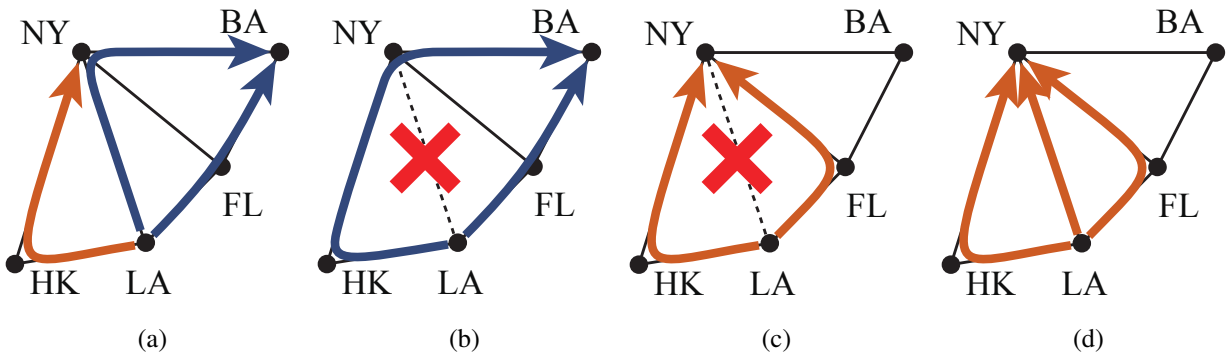


Figure 2.11: [Testbed] An example of failure handling. DataBundle of Job 1 is denoted by the blue (dark) arrows, DataBundle of Job 2 is denoted by orange (light) arrows. The dashed line with a cross denotes the failed link. (b): Link failed, Job 2 got preempted. (c): Job 1 finished, Job 2 got rescheduled. (d): Link recovered, Job 2 received a new path.

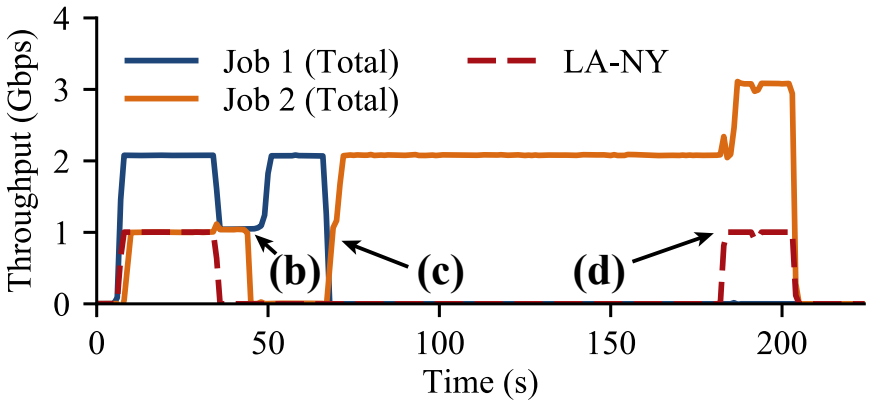
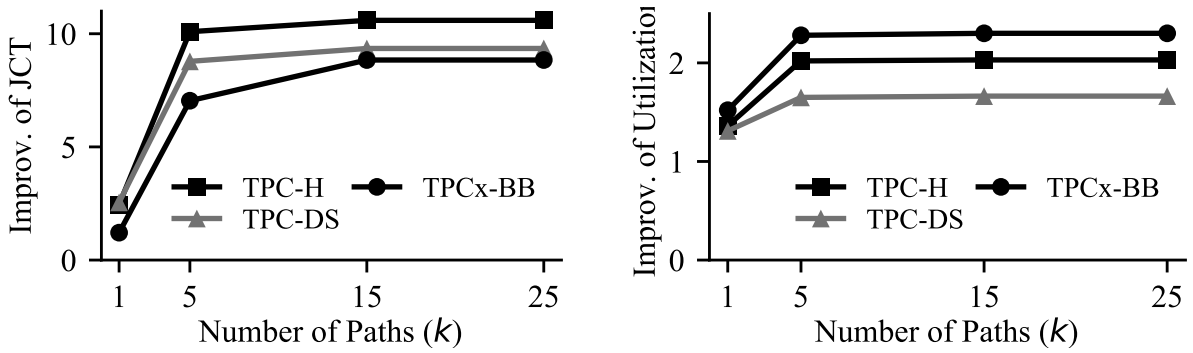


Figure 2.12: [Testbed] WAN throughput of the two jobs in Figure 2.11 on the LA–NY link as it cycled through failure and recovery states.

2.6.5 Complexity and Scalability

In our experiments, we run TERRA controller on a machine with two 2.6GHz Intel Xeon Gold 6142 CPU and measured the number of LP computations and total time spent for each scheduling decision. For TPCx-BB on Microsoft topology, for each schedule, TERRA needs to solve 28.4 LPs on average, taking 74.43 ms. For the same workload on AT&T topology, each schedules takes 589.1 ms to solve 31.46 LPs on average (2.991 s to solve 52 LPs at the 95th percentile). Given that many jobs last for a few minutes, this overhead is acceptable.

We also compare TERRA against Siphon [149]. Siphon is highly computationally intensive, with a complexity of $O(n^d)$ for the Monte Carlo Simulation-based scheduling [149]. For the Siphon



(a) Improvements in Avg. JCT

(b) Improvements in Utilization

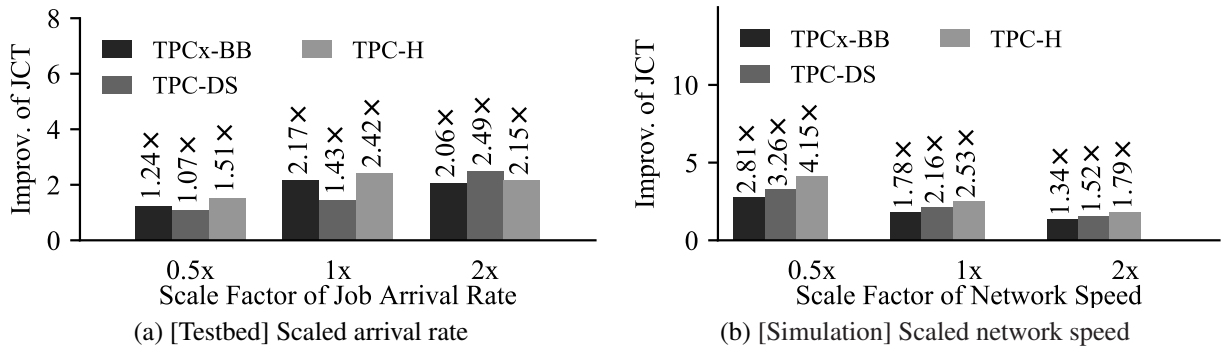
Figure 2.13: [Simulation] Improvements in the average JCT and utilization using TERRA w.r.t. Default on AT&T topology as the number of paths between two sites (k) is varied.

simulation, we choose $n = 100$ and $d = 2$ which is too small to guarantee good searching result. However, even with this minimal parameter setting, Siphon is $7.15\times$–$17.61\times$ slower across all topologies. For $n = 100$ and $d = 3$, Siphon is $763\times$–$1649\times$ slower, making it infeasible to deploy to online systems.

Finally, we compare our online algorithm with an offline-only 2-approximation algorithm [64] which is also based on solving an LP. Running the offline algorithm with TPCx-BB on the smallest Microsoft topology takes more than 2 days to complete, whereas our online algorithm takes only 11.2 min. This huge difference in overhead comes from their different LP formulations. In the offline algorithm, the number of variables in the LP is approximately $|Jobs| \cdot |Links| \cdot |Intervals|$, compared to $|Jobs| \cdot |Links|$ in our proposed online algorithm. Therefore, for any LP solver that takes $\Omega(n)$ time, the time complexity of the offline algorithm is higher by a factor of $\Omega(|Intervals|)$. This difference can be very large depending on the granularity and accuracy requirements for the offline algorithm.

2.6.6 Sensitivity Analysis

Impact of Restricting Paths If the number of possible paths between them (k) increases, TERRA has to establish and maintain more TCP connections. Figure 2.13 shows the impact of k for the all workloads on the AT&T topology. We observe that: (i) With larger k, TERRA's improvements become higher. This is because larger k allows for more multipath benefits. (ii) After k reached a threshold (it is between $k = 5$ and $k = 10$ for AT&T), increasing k does not significantly affect



(a) [Testbed] Scaled arrival rate

(b) [Simulation] Scaled network speed

Figure 2.14: Improvement in the average JCT for workloads with scaled (a) arrival rates and (b) network speed in the Microsoft topology w.r.t. Default.

JCT or utilization.

Impact of Job Arrival Rate and Network Speed We scale the arrival rate of the queries to increase load and evaluated TERRA's performance in our testbed. Figure 2.14a shows that TERRA performs better with increasing arrival rate – i.e., with increasing load. When the network is more heavily loaded, the room for performance improvement is higher. Increasing load by scaling down the network speed (instead of making jobs arrive in shorter intervals) also resulted in increasing benefits. (Figure 2.14b)

Computation vs. Communication By keeping the time spent in communication constant, we vary the number of machines in each site and estimate the average JCT. Due to resource constraints, we use the testbed for less than 40 machines per site and simulation for more than 40 machines per site. Figure 2.15 shows improvements of average JCTs w.r.t. the number of machines used for all jobs on Microsoft topology. Because JCT $= (T_{\text{Comm}} + T_{\text{Comp}})$ for each stage, the improvements increase with the number of machines used.

Choice of α We also vary α and measure the average JCT improvements across all workloads. Figure 2.16 shows that highest average JCT improvements are achieved with α between 0.1 and 0.2. This is because when α is too low starvation will occur, and when α is too high the performance will degrade to be the same as MPTCP without any scheduling.

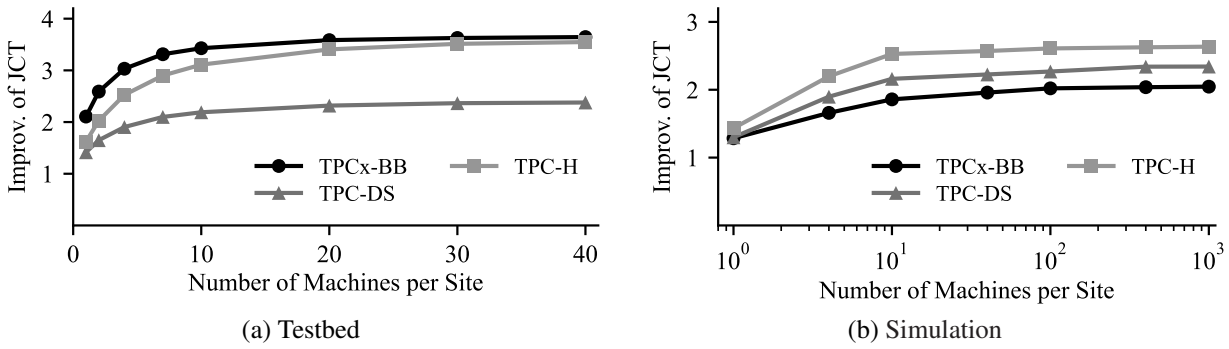


Figure 2.15: Improvements in the average JCT using TERRA w.r.t. Default on the Microsoft topology with increasing number of machines in each site.

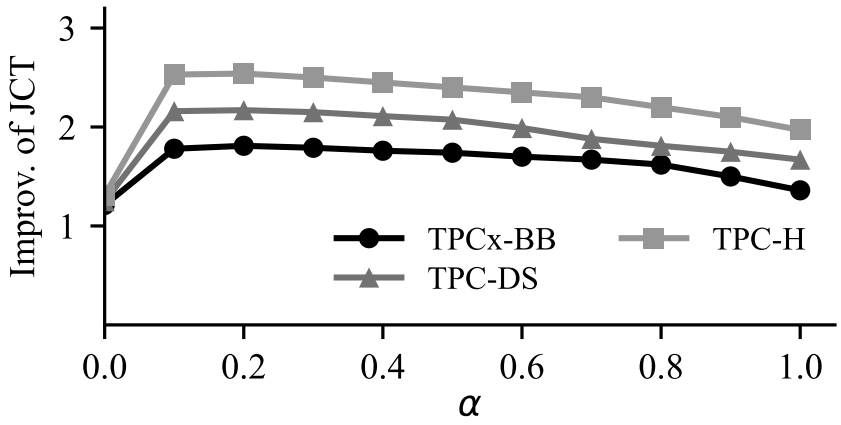


Figure 2.16: [Simulation] Improvements in the average JCT using TERRA w.r.t. Default on the Microsoft topology with increasing α.

2.7 Related Work

Wide Area and Geo-Distributed Analytics Systems Existing wide-area data analytics solutions primarily focus on two goals: (i) minimizing WAN usage and (ii) minimizing the average JCT via query planning and/or data placement [175, 202, 203, 204, 177, 103, 201, 108, 107, 130, 46, 45]. However, *they all ignore the WAN topology and cannot dynamically react to WAN variabilities*. In contrast, TERRA focuses on simultaneously scheduling and routing inter-site shuffles from HMCA queries and can complement existing solutions.

SmartCoflow [145] is a recently proposed framework for jointly optimizing endpoint placement and coflow scheduling to minimize the average completion time of coflows across geo-distributed datacenters. Similar to previous studies [175, 202, 203, 204, 177, 103, 201, 108, 107, 130, 46, 45],

it models the WAN as a fully connected mesh network, ignoring the underlying topology.

Siphon [149] is another solution that focuses on the WAN transfer aspect of HMCA jobs. However, it is also agnostic to the WAN topology assuming a full-mesh network. Furthermore, the multi-path routing is achieved by a heuristic that iteratively diverts traffic from the heavily loaded links to alternative paths, which does not scale in larger deployments as confirmed in our evaluation (§2.6.5). It does not address the implications of heterogeneous latency of the WAN and cannot directly react to WAN dynamics either.

Theoretical Results There have been several theoretical studies on analyzing the optimality of WAN transfers over general graphs with a focus on wide-area data analytics workloads. The earliest of them can be credited to Jahanjou et al. [111] with an approximation factor of 17.6.

A recent study improved it and proposed a 2-approximation algorithm [64]. However, these algorithms have many times higher time complexity than TERRA, and TERRA outperforms them in practice (§2.6.5). In addition, we present a full-stack implementation and integration of TERRA.

Wide-Area Network Traffic Engineering Traditional networking techniques for optimizing WAN transfers revolve around tuning weights and adapting allocations across pre-established tunnels, often via MPLS [216]. A recent work [136] combines selecting forwarding paths and rate adaptation together. Some others focus on WAN data transfers with deadlines [220, 114, 116, 136, 166]. TERRA extends these works by incorporating application-level semantics. With the advent of software-defined networking (SDN), Google [113, 134] and Microsoft [101] have shown that it is indeed possible to perform traffic engineering in a (logically) centralized manner. TERRA takes a similar approach in its implementation.

2.8 Conclusion

We present TERRA, a data transfer optimizer for hybrid multi-cloud analytics (HMCA) jobs that simultaneously performs scheduling, path selection, and bandwidth allocation for inter-site shuffles in a scalable manner. While existing wide-area data analytics (WADA) solutions focus only on minimizing the amount of inter-site data transfer, TERRA goes one step further to judiciously manage how those data must be transferred after those optimizations have been applied. It builds on a provably optimal offline algorithm and uses it as a building block to perform single- and

multi-query optimizations with large performance benefits. Evaluation using three benchmarks on three HMCA deployments show large performance benefits.

CHAPTER 3

KAYAK: Application-Aware Transaction Processing

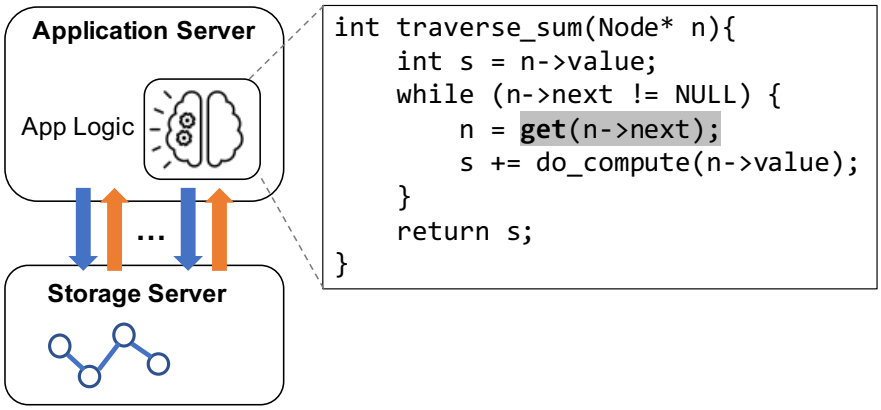
3.1 Introduction

Two trends stand out amid the rapid changes in the landscape of cloud infrastructure in recent years:

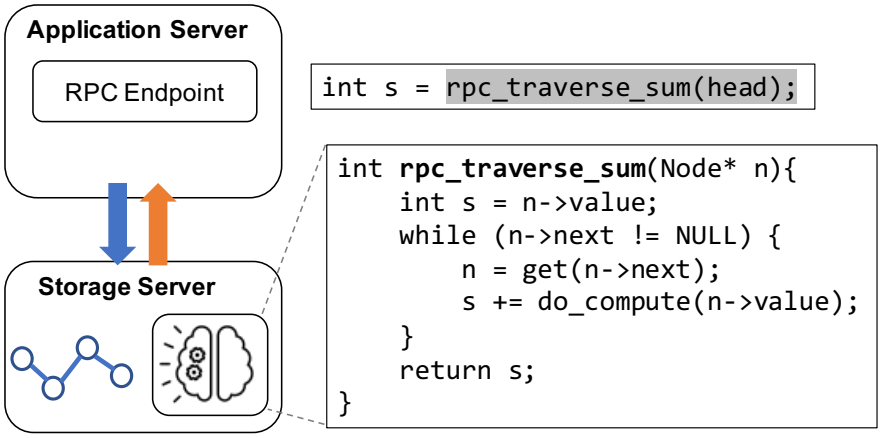
- First, with cloud networks moving from 1Gbps and a few hundred μs to 100Gbps and single-digit μs [119, 87, 52], disaggregated storage has become the norm [129, 20, 78, 190, 205, 109, 148]. It decouples compute from storage, enabling flexible provisioning, elastic scaling, and higher utilization. As a result, increasingly more applications now access their storage servers over the network using a key-value (KV) interface.
- Second, the steady increase in compute granularity, from virtual machines, to containers, to microservices and serverless functions, is popularizing storage-side computation using remote procedure calls (RPCs) [117]. Many databases allow stored procedures and user-defined functions [29, 120, 32, 193, 192, 25, 26], and some KV stores allow just-in-time or pre-compiled runtime extensions [133, 34, 187, 79].

The confluence of these two contradicting trends – the former moves data to compute, while the latter does the opposite – highlights a long-standing challenge in distributed systems: *should we ship compute to data, or ship data to compute?*

The answer, in broad strokes, boils down to the ratio of computation and communication. The benefits of storage disaggregation, i.e., shipping data to compute, typically holds when most of the time of a function invocation (hereafter referred to as a request) is spent in computation. However, when a single request triggers multiple dependent accesses to the disaggregated storage, time spent in network traversals and data (un)marshalling starts to dominate [42]. Figure 3.1a shows an



(a) Ship data to compute.



(b) Ship compute to data.

Figure 3.1: Graph traversal implemented with (a) disaggregated storage and (b) storage-side compute. The latter (3.1b) results in less network round-trips but exert more load on the storage server.

example of a simple graph traversal algorithm implemented on top of disaggregated storage, where "pointer chasing" can make network traversals the bottleneck.

In contrast, storage-side computation enables applications to offload part of their application logic to storage servers. The storage layer is customized to support application-specific RPC-style APIs [42, 133, 55], which shave off network round-trips. Figure 3.1b shows the previous example implemented with storage-side computing, where only one network round-trip is sufficient. However, this is not universally viable either; for compute-intensive workloads, the compute capacity of the storage servers can become the bottleneck when too much computation is offloaded to the storage.

In short, there is no one-size-fits-all solution. Existing works have explored different points in the design space (Figure 3.2). Arbitrarily forcing a disaggregated storage (i.e., ship data) or a

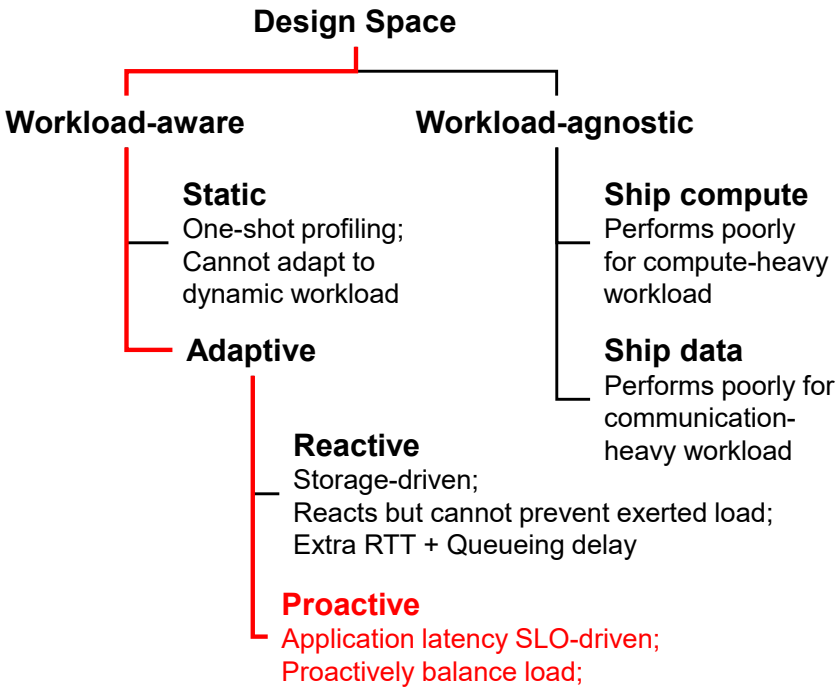


Figure 3.2: Design space of KAYAK.

storage-side computation architecture (i.e., ship compute) in a *workload-agnostic* manner leads to poor utilization and low throughput. Our measurements show that *workload-agnostic* solution leads to up to 58% lower throughput and 37% lower utilization when compared to the optimal. For *workload-aware* alternatives, one choice is taking a one-shot *static* approach, whereby a workload is profiled once at the beginning. However, statically choosing either KV- or RPC-based approach falls short even for a single tenant (§3.2.2).

The alternative, therefore, is taking an *adaptive* approach that can dynamically choose between shipping data and shipping compute. Existing adaptive solutions such as ASFP [55] take a *reactive* approach: all requests are forwarded using RPC to the storage server, which can then react by pushing some of them back. While this provides a centralized point of control, each request experiences non-zero server-side queueing delay, and more importantly, requests that are pushed back suffer from one *extra* round-trip time (RTT), which is detrimental to low-latency applications with strict latency SLOs. Moreover, throughput-driven designs cannot proactively throttle exerted load on the storage server w.r.t. tail latency service-level objectives (SLOs).

We observe that an ideal solution fundamentally calls for a *balanced* architecture that can

effectively utilize the available resources and increase overall throughput while satisfying tail latency SLOs. In this project, we present KAYAK that takes a *proactive* adaptive approach to achieve these goals (§3.3). In order to maximize throughput without SLO violations, KAYAK proactively decides between shipping compute and shipping data when executing incoming requests, and it throttles request rate in order to meet SLO requirements. Specifically, KAYAK takes a latency-driven approach to optimize two parameters simultaneously: (1) the *request rate*, and (2) the *RPC fraction*, which denotes the proportion of the incoming requests to be executed using RPC.

Unfortunately, the optimal RPC fraction varies for different workloads and their SLO requirements. There is no closed-form expression to precisely capture the relationship between RPC fraction, request rate, and tail latency either. Finally, we show that the order in which we optimize request rate and RPC fraction affects convergence of the optimization algorithm. We address these challenges by designing a dynamic optimization method using a dual loop control (§3.4). KAYAK employs a faster control loop to optimize request rate and a slower one to optimize RPC fraction. Combined together, KAYAK iteratively searches for the optimal parameters, with a provable convergence guarantee. In addition to increasing throughput in the single-tenant scenario, KAYAK must also ensure fairness and work conservation of shared server resources in multi-tenant settings. KAYAK pins tenants to CPU cores in a fair manner and employs work stealing to achieve work conservation.

Our evaluation on a prototype of KAYAK shows that: (1) KAYAK achieves sub-second convergence to optimal throughput and RPC fraction regardless of workloads; (2) KAYAK improves overall throughput by 32.5%-63.4% for compute-intensive workloads and up to 12.2% for non-compute-intensive and transactional workloads; and (3) in a multi-tenant setup, KAYAK approximates max-min fair sharing and scales without sacrificing fairness.

3.2 Motivation

3.2.1 Limitations of Existing Designs

Existing solutions either fail to efficiently utilize the available CPU cores for a large variety of workloads or introduce additional overhead to reactively adapt to workload variations, both of

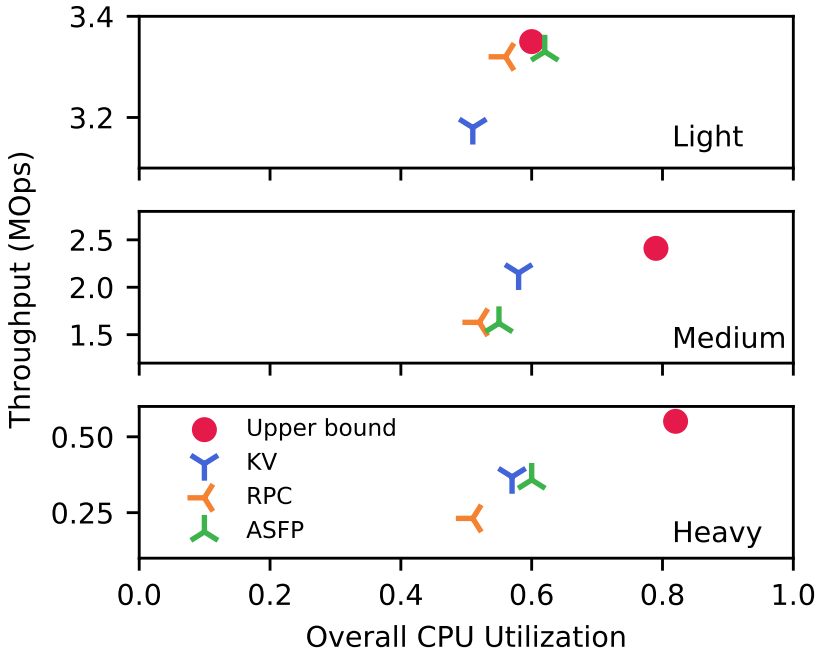
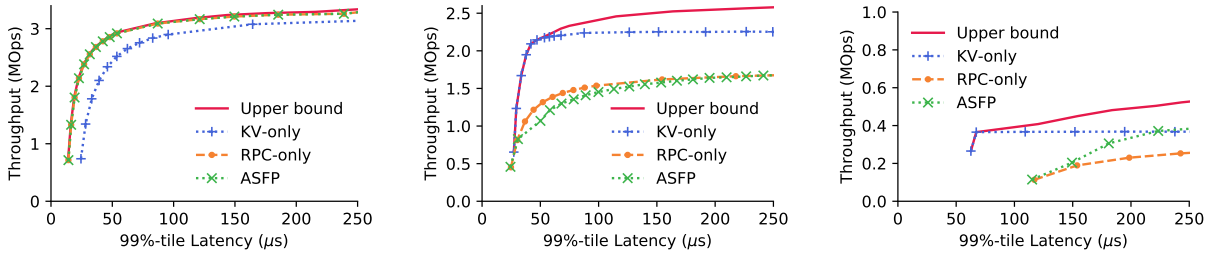


Figure 3.3: Throughput and overall request-level CPU utilization across both application and storage servers for three different workloads under different execution schemes, with 200μs SLO.



(a) Light (100ns/RTT) (b) Medium (1μs/RTT) (c) Heavy (10μs/RTT)

Figure 3.4: Throughput w.r.t. SLO (99%-tile latency) for 3 different workloads under different execution schemes.

which lead to lower throughput.

Workload-agnostic approaches either use KV-only design or RPC-only design. The former results in excessive network round-trips of storage access during execution, while the latter overloads the storage server CPU and leaves the CPU on the application server underutilized. In either case, the overall CPU utilization is low which hinders the performance.

ASFP [55] presents an alternative to workload-agnostic solutions by taking a workload-aware

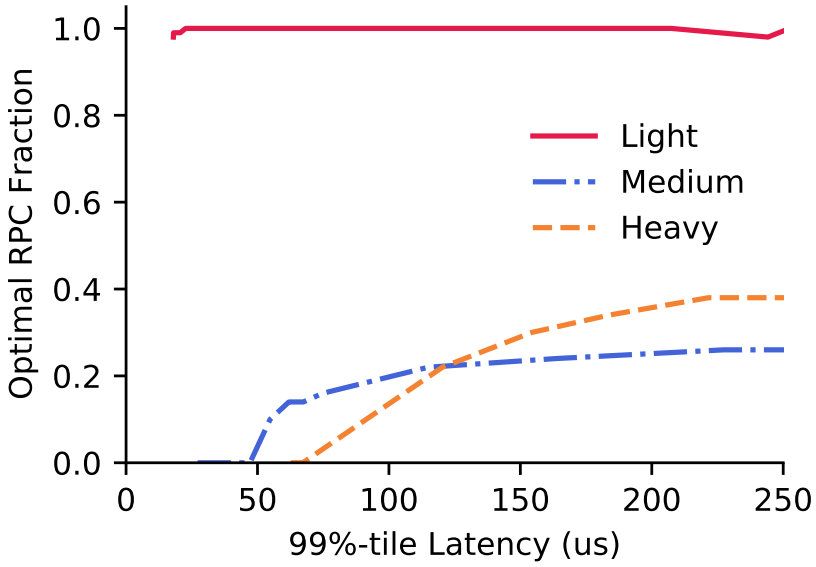


Figure 3.5: Optimal RPC fraction w.r.t. SLO (99%-tile latency) for 3 different workloads.

approach with runtime adaptation. In that design, the requests are executed using RPC by default, and if the storage server gets overloaded by the exerted computation, a pushback mechanism is triggered to push the exerted computation back to the application server side. However, the excessive queueing on the storage server still cannot be prevented, although it can be alleviated by the pushback mechanism. Furthermore, the execution of pushed-back requests needs to restart on the application server, wasting CPU cycles on both servers.

To illustrate these issues, we perform an experiment with the graph traversal application shown in Figure 3.1. We configure the workload so that the each request triggers two storage accesses (i.e., two network round-trips) when executed using the KV scheme. We vary the computation after each access and refer to them as Light (100ns computation time per access), Medium (1μs per access) and Heavy (10μs per access). For reference, adding one network round-trip incurs 9.2μs more latency for each request in our testing environment. We measure the maximum achievable throughput and the CPU utilization (defined as CPU cycles spent only in executing the requests). As shown in Figure 3.3, using only KV or RPC leads to lower overall CPU utilization and lower throughput. Although the reactive adaptive design utilizes a higher amount of CPU on both application and storage servers, its overheads add up quickly, and its CPU usage for request-level computation does not increase significantly.

To summarize, existing designs do not efficiently utilize the CPU resource on both application and storage servers, which limits their performance. There exists an optimal RPC fraction that maximizes the throughput (calculated by a comprehensive sweep as explained below).

3.2.2 Need for Dynamically Finding the Optimal Fraction

A key challenge here is that this optimal fraction varies for different workloads. To highlight this, we perform another experiment with the same graph traversal workload as before. We configure the application server to handle a fraction of the requests using RPC and the rest using the KV approach. We vary the RPC fraction from 0 to 1 and measure the overall throughput and end-to-end latency of all requests. In doing so, we obtain the throughput-latency measurements for all possible execution configurations, as shown in Figure 3.4 and Figure 3.5. The *upper bound* in Figure 3.4 is defined by selecting the best RPC fraction for each latency SLO to maximize the throughput, and the selected fraction is plotted in Figure 3.5. ASFP is configured as using RPC by default with pushback enabled.

As shown in Figure 3.4, workload with less computation favors RPC over KV, and vice versa. For Medium and Heavy workloads, the highest throughput is achieved by combining RPC and KV together. Moreover, we observe that, (1) for different workloads, the highest throughput is achieved with different RPC fraction; and (2) for different SLO constraints, the optimal fraction for highest throughput is also different. This calls for a proactive design to search for the optimal RPC fraction. Note that we repeated the same parameter sweep with the number of storage accesses per request set to 4 and 8, and observe similar trends (please refer to Appendix A.1.1).

3.3 KAYAK Overview

KAYAK proactively decides between *shipping compute* and *shipping data* in a workload-adaptive manner. It arbitrates incoming requests and proactively decides the optimal RPC fraction (i.e., what fraction of the compute to ship to storage servers) of executing the requests while meeting end-to-end tail latency SLO constraints.

3.3.1 Design Goals

KAYAK aims to meet the following design goals.

- *Maximize throughput without SLO violations:* Applications have stringent tail latency SLO constraints to ensure low user-perceived latency. KAYAK should maximize the throughput while not violating these SLO constraints.
- *High CPU utilization across all servers:* KAYAK should balance the computation imposed by application logic across the application and storage servers.
- *Fair sharing of storage server resources:* In a multi-tenant cloud, multiple applications may be sending requests to the same storage server, which should be fairly shared.
- *Ease of deployment:* Applications should be able to use KAYAK with minimal code modification.

3.3.2 Architectural Overview

At a high level, KAYAK adaptively runs application logic on both the application and storage servers (Figure 3.6). However, even though it ships some computation to the storage server, KAYAK's core control logic runs only on the application server and the storage server acts as an extended executor. Unlike existing reactive solutions, KAYAK proactively decides the amount of computation to ship to the storage side.

Key ideas. Essentially, KAYAK finds the optimal RPC fraction and maximizes throughput at runtime while meeting tail latency SLO constraints. The main design challenge is that the optimal fraction varies in accordance with different workloads and their SLO requirements (Figure 3.5), as well as the amount of load exerted on the storage server. In a multi-tenant cloud, all of these change dynamically. In order to keep up with the changing environment, KAYAK proactively adjust the RPC fraction and the request rate according to realtime tail latency measurements.

However, the relationship between the request rate, RPC fraction and latency cannot be easily captured with a closed-form expression. Thus KAYAK adopts a numerical optimization method based on a dual loop control algorithm (§3.4.3) to search for the optimal parameters iteratively. We notice that latency measurements in real systems exhibit large variance, which detrimentally impacts the performance of such iterative optimization algorithms. KAYAK's algorithm accounts for

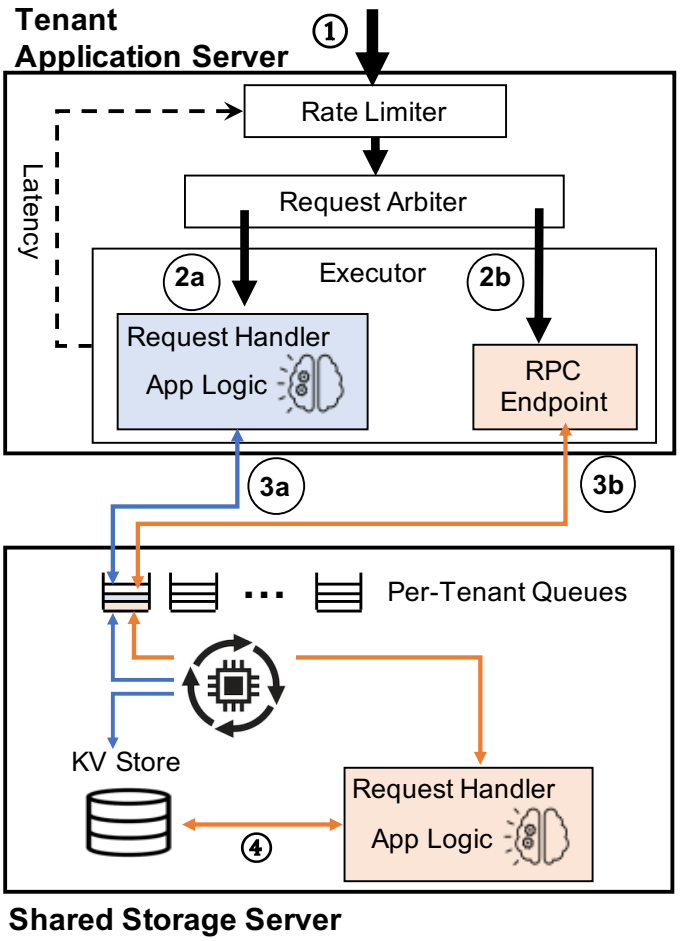


Figure 3.6: KAYAK architecture. Tenant application servers interact with shared storage servers using KAYAK, which proactively decides which requests to run on the application server using KV operations and which ones to ship to the storage server via RPC.

such variance and has a provable convergence guarantee.

From an implementation perspective, KAYAK faces another challenge as it has to make adjustments very quickly due to the high-throughput, low-latency nature of the environment. KAYAK cannot afford to gather global information and make centralized decisions. Hence, the algorithm for KAYAK is fully decentralized and runs only on the application servers.

Application server. The application server consists of three main components (Figure 3.6): (i) the Rate Limiter limits the rate of incoming requests from the tenant to satisfy SLO constraints; (ii) the Request Arbiter determines the optimal RPC fraction; and (iii) the Executor handles the requests according to the scheme decided by the Request Arbiter.

The Rate Limiter interacts with the tenants and ① receives incoming requests. It continuously monitors real-time tail latency of end-to-end request execution, and pushes back to signal the tenant to slow down. When the servers are overloaded and the SLO cannot be met, it drops overflowing requests. We assume that tenants implement mechanism to handle overflowing, such as resubmit dropped requests, and increase provisioning so that in the worst case scenario all requests can still be executed on all application servers within the SLO.

The Request Arbiter proactively determines the optimal RPC fraction. It selects the execution scheme for each request based on that fraction. For each request, the choice of execution scheme is determined by a Bernoulli distribution $B(1, X)$ where X is the proportion of requests processed using RPC.

The Executor handles the request in its entirety and reports the end-to-end completion time back to the Rate Limiter upon completion. It consists of two parts: (i) the Request Handler and (ii) the RPC Endpoint. If the request is to be executed using ②a the KV scheme, the Request Handler is triggered. The Request Handler executes the application logic locally on the application server and keeps track of the states of the request. Whenever it needs to access data stored in the storage server, ③a the KV API of the storage server is subsequently called. In contrast, if the request is to be executed on the storage side using the RPC scheme, then the request is simply forwarded to ②b the RPC Endpoint on the application server. The RPC Endpoint issues an RPC request ③b to the storage server for processing the request.

Storage server. The storage server includes an additional Request Handler to handle RPC requests in addition to a KV interface. Similar to allocating CPU cores in the application server to run application code, in KAYAK, computation resources in the storage server are also allocated to specific tenants at the CPU core granularity.

Each tenant has a dedicated request queue, from which its core(s) polls KV and RPC requests. Handling an incoming KV request in KAYAK is the same as what happens in a traditional KV store: the request is simply forwarded to the KV store. Upon receiving an RPC request, the Request Handler is triggered and executes the application logic on the storage server. The Request Handler calls ④ the local KV API whenever data access is needed, interacting with the stored data without crossing the network.

This static *pin-request-to-core* allocation scheme of KAYAK makes it easier to enforce fair computation resource sharing between tenants. However, static allocation of CPU cores cannot guarantee work conservation of the CPU cores on the storage server. KAYAK uses *work stealing* to mitigate this issue: whenever a tenant's dedicated queue is empty, the corresponding CPU core *steals* requests from other queues.

3.4 KAYAK Design

Our primary objective is to maximize the total throughput without violating the tail latency SLO. However, higher throughput inevitably leads to higher latency in a finite system [128], and there exists a fundamental tradeoff between throughput and latency. Unfortunately, the precise relationship between latency and throughput of a real system, however, is notoriously difficult to be captured by a closed-form expression. In this project, we use an analytical model to highlight our insights and take a tail latency measurement-driven approach to design a pragmatic solution.

At the same time, as illustrated in Section 3.2, a reactive approach to achieve this can lead to CPU wastage. Hence, KAYAK proactively decides what fraction of the requests to offload vs. which ones to run in the application server, while maximizing the total throughput within the SLO constraint. The need for optimizing both raises a natural question: *which one to optimize first?* In this section, we analyze both optimization orders and design a dual loop control algorithm with provable convergence guarantees. Detailed proofs can be found in the appendix.

3.4.1 Problem Formulation

We denote the proportion of requests to be executed using RPC by X, the total incoming request rate by R, and we define τ as the random variable of request latency, thus we have:

$$\tau \sim P(R, X),$$

where R and X are the parameters of distribution P. Table 3.1 includes the key notations used in this project.

We denote $T(X, R)$ as our SLO statistics metric, which takes a specific statistical interpretation

Sym.	Description
R	Total request rate
X	Proportion of requests processed using RPC
τ	Random variable of request latency
t_o	Latency SLO target
$T(X, R)$	Latency SLO as a function of X, R
$R(X)$	Function implicitly defined by $T(X, R(X)) = t_0$
k	Index of iterations

Table 3.1: Key notations in problem formulation.

for the particular SLO metric. For instance, if the SLO is defined as the 99%-tile latency then T is the 99%-tile for τ. We denote t_0 as the SLO target under the same statistic metric. Thus the problem can be formulated as:

$$\max_{X} \quad R \tag{3.1}$$

$$s.t. \quad T(X, R) \leq t_0 \tag{3.2}$$

$$R > 0, \tag{3.3}$$

$$X \in [0, 1]. \tag{3.4}$$

Here constraint (3.2) captures the latency SLO constraint, and constraints (3.3) and (3.4) represents the boundary of R and X, respectively.

We make the following observation when solving this optimization problem:

Observation 1. *Fixing X, $F_X(R) := T(X, R)$ is monotonic increasing.*

Observation 1 captures the relationship of throughput and latency from queueing theory [128] for finite systems like KAYAK.

3.4.2 Strawman: X-R Dual Loop Control

Optimization (3.1) cannot be directly solved with a closed-form solution of R and X due to the intractability of the function $T(X, R)$. Therefore, we use a numeric optimization method and try to optimize R and X independently and iteratively. To put it into our context, we need to design an iterative algorithm such that in each iteration, we first optimize either R or X, and then optimize

the other. We also have to prove that this algorithm would actually converge to ensure optimality and stability of the system.

Now we are facing a question: *which one to optimize first?* In our problem, there is an asymmetry for X and R: X is the parameter in Optimization (3.1) where as R is the objective. A straightforward solution is to optimize X first, which leads to the following algorithm.

Algorithm I (X-R Dual Loop Control) For $k = 1, \ldots, K$, we alternatively update X_k and R_k by

1. Fix R_k, and find the RPC fraction X_k that minimizes latency, i.e., $X_k = \arg\min_X T(X, R_k)$;
2. Update R_k according to *gradient descent* so that $T \approx t_0$, i.e.,

$$R_{k+1} = R_k + \eta \left(t_0 - T(X_k, R_k)\right),$$

where $\eta > 0$ is the stepsize.

In this algorithm, the first step is to solve a convex optimization problem. Because our assumptions guarantee that X_k is unique and finite, this iteration is well defined for at least the first step (and we will show it is also good for the second step). Moreover, because $T_{R_k}(X) := T(X, R_k)$ is μ-strongly convex and L-smooth, X_k can be solved very quickly (or mathematically, in a linear rate) by iterative algorithms such as gradient descent. We have the following theorem that characterizes the convergence of this algorithm. The rigorous statements and proofs are deferred to Appendix A.2.2.

Theorem 1. *Fixing R, $F_R(X) := T(X, R)$ is strongly convex and smooth. Suppose for all X, $0 < \alpha \leq \frac{\partial T(X,R)}{\partial R} \leq \beta$. Let $0 < \eta < \frac{1}{\beta}$, then under mild additional assumptions*[1]*,*

$$|R_K - R_*| \leq (1 - \eta\alpha)^K \cdot |R_0 - R_*|.$$

Here R_* denotes the optimal request throughput, and R_0 denotes the initialization. This result shows the iteration of X-R Dual Loop Control converges to the optimal requests exponentially fast, i.e., after at most $\mathcal{O}(\log \frac{1}{\epsilon})$ iterations, the algorithm outputs a solution that is ϵ-close to the optimal.

[1]For the sake of presentation, we omit the technical assumptions. For a complete description on the theorem, please refer to Appendix A.2.2

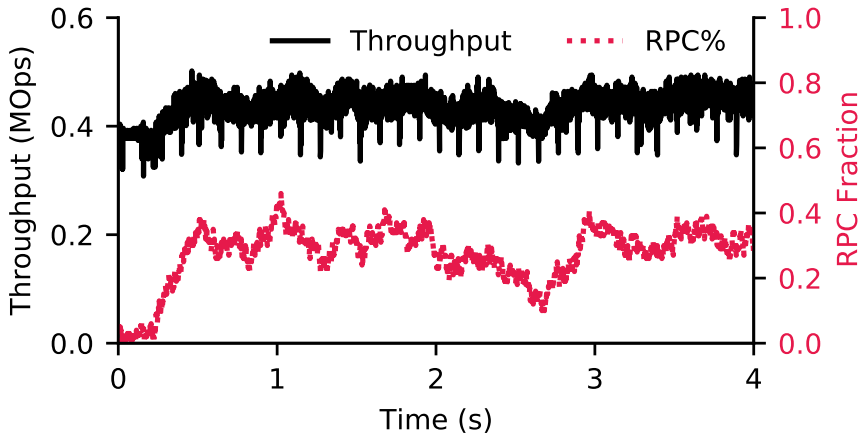


Figure 3.7: Instability of throughput and RPC fraction w.r.t time, with X-R Dual Loop Control.

Instability of X-R dual loop control. However, while Algorithm I is theoretically sound, it is not practical to be implemented in a real system. The key obstacle is that the latency SLO metric cannot be *directly* obtained in practice. Instead, we can only measure a set of samples of latency τ, and then gather statistics to derive the SLO metric. Hence the derived SLO metric – be it average or 99%-tile – is only an *estimate* $\widehat{T}$ based on *sampling*. While sampling might not be a problem for many systems by using a high sampling rate, it is indeed a problem for KAYAK. In particular, because of the microsecond-scale workload and the real-time requirement of KAYAK, the sample size for each estimate is limited. This leads to *large variance* in the estimated $\hat{T}$, which results in *degraded* convergence speed and quality.

To quantify the impact of this variance and show the gap between theory and practice, we conduct a verification experiment. We run Algorithm I with the Heavy workload from Section 3.2 under an SLO constraint of 200μs 99%-tile latency. Our experiment confirms the aforementioned issue of variance in SLO estimates. Figure 3.7 shows poor convergence quality of both the throughput and the RPC fraction.

3.4.3 Our Solution: R-X Dual Loop Control

A naive mitigation to counter the SLO variance is to simply use a metric that is more robust, such as average latency. But this limits the operators to only one viable SLO metric and compromises the generality of the system.

In order to solve the challenge of unstable SLO estimates, we must design an algorithm that

Input: Current throughput R, latency t, SLO target t_0
Output: Updated throughput R

/* Initialize global variables. */
$\mathbb{T} \leftarrow 0$; $\mathbb{R} \leftarrow 0$ ▷ Last involved latency and throughput.
/* Update R for each round. */
Procedure UpdateR ( )
  /* Calculate Δ_R according to Newton's method. */
  $\Delta_R \leftarrow \frac{(R-\mathbb{R})(t_0-T)}{T-\mathbb{T}}$
  /* Bounds checking, throughput should be positive. */
  if $R + \Delta_R$ *¡ 0* **then** ▷ Unlikely. Violates (3.3).
    $R \leftarrow \frac{R}{M}$ ▷ Discard Δ_R and divide R by half.
  else
    $R \leftarrow \Delta_R + R$

Pseudocode. 2: Dynamic search of optimal R.

is not sensitive to the variance of $\hat{T}$. Compared with the RPC fraction X, the request rate R has a more intuitive and better-studied interaction with latency T from extensive study in queueing theory. Specifically, we take inspiration from recent works [80, 135] showing that even with variance in latency measurements, one can still achieve rate control (i.e., optimization of the throughput) in a *stable* manner. From the starting point of latency-driven rate control, we design a dual loop control algorithm that first optimizes R and then optimizes X to numerically solve the optimization problem. The algorithm is shown as follows. The first part is latency-driven rate control, and the second is gradient ascent.

Algorithm II (R-X Dual Loop Control) For $k = 1, \ldots, K$, we respectively update X_k and R_k by

1. Apply rate control so that the latency approximates SLO, i.e., R_k *be such that* $T(X_k, R_k) \approx t_0$*;*
2. Use gradient ascent to search for the optimal X_k, i.e., $X_{k+1} = X_k + \eta \frac{\mathrm{d}R_k}{\mathrm{d}X}$, *where* $T(X, R_k) = t_0$*, and* η *is a positive stepsize.*

R loop: rate control. In order to satisfy the SLO requirement ($T \leq t_0$), we need to carefully control the request rate R. Intuitively, too high an R leads to excessive queueing on the server side,

causing SLO violations; at the same time, too low an R leads to low overall throughput and low resource utilization.

Let $R(X)$ be a function implicitly determined by the boundary constraint Eq. (3.2), i.e.,

$$T(X, R(X)) = t_0.$$

The implicit function $R(X)$ is indeed well defined, since for any X, $T(X, R)$ is monotonically increasing,[2] implying there exists an unique request throughput $R(X)$ that satisfies the boundary constraint, i.e., the maximum throughput is achieved when the latency is equal to the SLO target.

Essentially, we have to design a dynamic algorithm that actuates $R(X)$ in real-time (via $R_k(X)$ in step 1). This problem can be solved with a root-finding algorithm such as the classic Newton's method. However, if we apply this method directly, we may encounter situations where the updated throughput R is negative, which violates constraint (3.3). This happens when the throughput is too high and needs to be significantly reduced. In this case, we divide R by M instead of updating it using Newton's method. This ensures that (i) the updated throughput is positive; and (ii) the updated throughput is still significantly lower than before. We note that this out-of-bound scenario does not happen frequently. For simplicity, we choose $M = 2$. Our algorithm of searching for the optimal R is shown in Pseudocode 2.

X loop: RPC fraction control. For any given RPC fraction, the rate control of KAYAK essentially maximizes throughput within the allowance of SLO requirement. With rate control, we effectively get the throughput as a function of the given RPC fraction ($R(X)$). In this part, we focus on the complementary and optimize the RPC fraction to maximize $R(X)$. We use a gradient ascent algorithm to achieve that. When the updated RPC fraction falls out of the range of $[0, 1]$, we apply rounding to ensure it is within the boundary. Our algorithm of searching the RPC fraction is shown in Pseudocode 3.

Putting them together. Combing the rate control and the RPC fraction control, our algorithm (Algorithm II) naturally forms a bi-level (nested) control loops [73], with two actuators X and R

[2]We assume that $T(X, R)$ is continuous, and for any X, there exist R_1 and R_2 such that $T(X, R_1) \leq t_0 \leq T(X, R_2)$. This assumption pluses monotonicity yields the existence and uniqueness of the implicit function $R(X)$.

Input: Current throughput R, RPC propotion X,
Output: Updated RPC propotion X

/* Initialize global variables. */
$\mathbb{R} \leftarrow 0;\ \mathbb{X} \leftarrow 0$ ▷ Last involved throughput and RPC fraction.

/* Update X for each round. */
Procedure UpdateX ( )
    /* Calculate Δ_X according to Gradient Ascent. */
    $\Delta_X \leftarrow -\eta \frac{R-\mathbb{R}}{X-\mathbb{X}}$
    /* Bounds checking, X should be within constraints. */
    if $X + \Delta_X \notin [0, 1]$ **then** ▷ Unlikely. Violates (3.4).
        $X \leftarrow \max\{\min\{X + \Delta_X, 1\}, 0\}$
    else
        $X \leftarrow \Delta_X + X$

Pseudocode. 3: Dynamic search of optimal X.

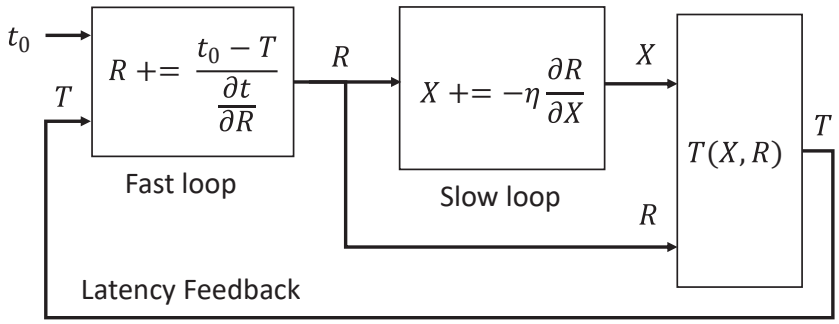


Figure 3.8: Nested control loops of KAYAK.

and only one feedback signal t. We adopt a single control loop (the inner/fast loop), called R loop, to implement the rate control, i.e., finding the maximum throughput R while not violating the SLO t_0. The input of this control loop is the measured latency SLO metric $\widehat{T}$ and the output is request rate R which is the input for our request arbiter. We then adopt another control loop (the outer/slow loop), called X loop, to implement our request arbiter, i.e., choosing the best X that maximizes R_0.

Although this dual loop control design decouples the two actuators X and R, the resulting two feedback loops may be coupled. The coupling between two feedback loops may cause oscillation, which can be mitigated by choosing different sampling frequencies [73]. The exact two values can be tuned by the operator according to different workloads and system configurations. However,

because the functioning of the second loop is dependent on the output of the first loop (R) to have converged to a stable point, it is best practice to choose a lower frequency for the second loop. Theoretically, we show that this dual loop control algorithm is guaranteed to converge in Section 3.4.4. Empirically, in our experiments, we let the sampling rates of the first and second loops to be 200Hz and 20Hz respectively, and we show that the system converges fast to near optimal throughput in Section 3.6. We evaluate the impact of frequency selection in detail in Section 3.6.5.

3.4.4 Perfomance Guarantee

From the R loop, we obtain an estimation $R_k(x)$ at each iteration k, which approximately satisfies $T(X, R_k(X)) \approx t_0$. In the X loop, we optimize X for our request arbiter such that R_0 is maximized. This is done by *stochastic gradient ascent* (SGA, or *online gradient ascent*) on X. There is a rich literature in online learning theory for SGA when $R_k(x)$ is concave, e.g., see [168]. Applying related theoretical results to our problem, we have the following performance guarantee for our system. The proof of Theorem 2 is deferred to Appendix A.2.3.

Theorem 2. *Suppose for all $k = 1, \ldots, K$, $R_k(X)$ is concave, and $\|\nabla R_k(X)\|_2 \leq L$. Consider the iterates of SGA, i.e.,*

$$X_{k+1} = X_k + \eta \nabla R_k(X_k).$$

Then we have the following regret bound

$$\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) \leq C \cdot \sqrt{K}, \tag{3.5}$$

where $C := L \|X_1 - X_*\|_2$ is a constant depends on initialization and gradient bound, and X_* can be any fixed number. Note that the regret bound holds even $R_k(X)$ is chosen adversarially based on the algorithm history.

Inteperation of Theorem 2. The sublinear regret bound implies SAG behaviors nearly optimal on average: we see this by setting $X_* = \arg\max_X \sum_{k=1}^{K} R_k(X)$, and noticing that

$$\frac{1}{K}\sum_{k=1}^{K} R_k(X_*) - \frac{1}{K}\sum_{k=1}^{K} R_k(X_k) \leq \mathcal{O}\left(\frac{1}{\sqrt{K}}\right) \to 0.$$

More concisely, in our algorithm, $\{R_k(X)\}_{k=1}^K$ corresponds to a sequence of inaccurate estimations to the true implicit function $R(X)$ — even so the theorem guarantees a sublinear regret bound, which implies that our algorithm behaviors nearly as good as one can ever expect under the estimations, no matter how inaccurate they could be.

Furthermore, if for each k, $R_k(X)$ is an *unbiased estimator* to the true concave function $R(X)$, i.e., $\mathbb{E}R_k(X) = R(X)$, then $\bar{X} = \frac{1}{K}\sum_{k=1}^K X_k$ converges to the maximal of $R(X)$ in expectation: we see this by choosing $X_* = \arg\min_X R(X)$ and noticing that

$$\begin{aligned}
\mathbb{E}\left[R(X_*) - R(\bar{X})\right] &\leq \frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[R(X_*) - R(X_k)\right] \\
&= \frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[R_k(X_*) - R_k(X_k)\right] \\
&\leq C \cdot \frac{1}{\sqrt{K}} \to 0.
\end{aligned}$$

The above convergence result does not require any assumptions on the randomness of R_k, as long as $R_k(X)$ is an unbiased estimator of $R(X)$. This means our algorithm can *tolerate* variance in the measured latency which causes variance in estimated R_k. The convergence is empirically validated by our experiments in Section 3.6.1.

3.4.5 Scalability and Fault Tolerance

Scalability. KAYAK is fully decentralized, and its control logic (e.g., rate and RPC fraction determination) is decoupled from the request execution in the dataplane. Throughput of a tenant is limited by its total available resources in application and storage servers; one can increase throughput by adding more application servers or by ensuring more resource share in the storage servers.

Fault tolerance. KAYAK does not introduce additional systems components beyond what traditional KV- or RPC-based or hybrid systems do. As such, it does not introduce novel fault tolerance challenges. The consistency and fault tolerance of the KV store is orthogonal to our problem and out of the scope of this project.

3.5 Implementation

We build a prototype of KAYAK with about 1500 lines of code and integrate it with the in-memory kernel-bypassing key-value store Splinter [133]. The code is available at: `https://github.com/SymbioticLab/Kayak`

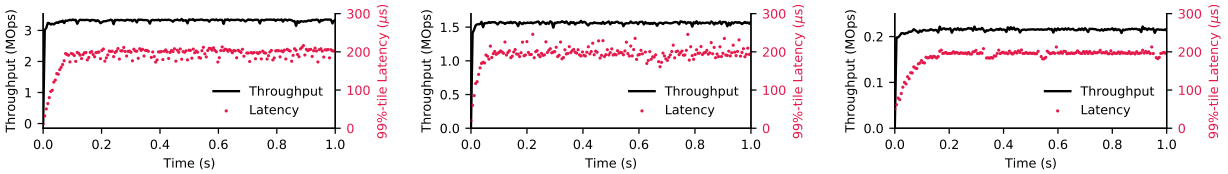
KAYAK interface. Users of KAYAK provide their custom defined storage functions (App Logic in Figure 3.6), which are compiled with KAYAK and deployed onto both the application server and storage server. At runtime, users connect to KAYAK and set the desired SLO target. Users then submit request in the format of storage function invocations to KAYAK.

Application server. The core control logic of KAYAK is implemented in the application server. One challenge we face during implementation is to optimize the code to reduce overhead, which is especially important because of the high throughput low latency requirement. For instance, the inner control loop constantly measures request latency and calculate the 99%-tile. One naive way is to measure the quantile is using selection algorithm to calculate the k-th order statistics of n samples, with has at least $O(n)$ complexity. Instead, we apply DDSketch [157] to estimate the quantile in real time with bounded error.

Storage server. The main challenge of implementing the storage server is supporting multi-tenancy and ensuring fairness and work conservation. We pin requests from different tenants to different CPU cores to ensure fairness. And we adopt work stealing to ensure work conservation: CPU cores with no requests to process steal requests from the queues of other cores. Specifically, similar to ZygOS [174], each CPU core of KAYAK steals from all other CPU cores, which is different from Splinter's work stealing from only neighboring cores. This further improves overall CPU utilization.

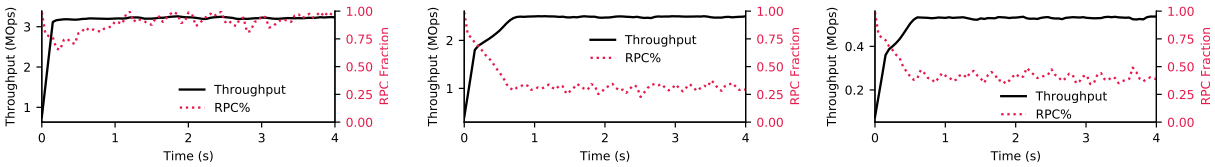
3.6 Evaluation

In this section we empirically evaluate KAYAK with a focus on: (i) verification of convergence; (ii) performance improvement against state of the art [55]; and (iii) fairness and scalability with



(a) Light (100ns/RTT) (b) Medium (1μs/RTT) (c) Heavy (10μs/RTT)

Figure 3.9: Throughput and 99%-tile latency w.r.t time, with only the fast loop of KAYAK and fixed RPC fraction of 100%.



(a) Light (100ns/RTT) (b) Medium (1μs/RTT) (c) Heavy (10μs/RTT)

Figure 3.10: Dynamics of throughput and RPC fraction w.r.t time, with nested control loops of KAYAK.

multiple tenants. Our key results are as follows.

- KAYAK achieves sub-second convergence to optimal throughput and RPC fraction regardless of workloads. It can proactively adjust to dynamic workload change as well (§3.6.1).
- KAYAK improves overall throughput by 32.5%-63.4% for compute-intensive workloads and up to 12.2% for non-compute-intensive and transactional workloads (§3.6.2).
- In a multi-tenant setup, KAYAK approximates max-min fair sharing, with a Jain's Fairness Index [112] of 0.9996 (§3.6.3) and scales without sacrificing fairness (§3.6.4).
- We also evaluate KAYAK's sensitivity to its different parameters (§3.6.5).

Methodology. We run our experiments on CloudLab [12] HPE ProLiant XL170r machines (Table 3.2). Unless specified otherwise, we configure KAYAK to use 8 CPU cores across all servers.

CPU	Intel E5-2640v4 2.4 GHz
RAM	64GB ECC Memory DDR4 2400MHz
NIC	Mellanox ConnectX-4 25 GB NIC
OS	Ubuntu 16.04, Linux 4.4.0-142

Table 3.2: Server configurations for our testbed in CloudLab.

SLO	YCSB-T		Light		Medium		Heavy		Bimodal	
	KAYAK	ASFP	KAYAK	ASFP	KAYAK	ASFP	KAYAK	ASFP	KAYAK	ASFP
50μs	2.63	2.58	3.05	3.05	1.71	1.06	N/A	N/A	2.13	1.43
100μs	3.12	2.78	3.36	3.36	2.37	1.45	0.37	N/A	2.74	2.16
200μs	3.35	3.01	3.59	3.52	2.54	1.64	0.48	0.33	2.98	2.37
400μs	3.35	3.02	3.70	3.61	2.61	1.68	0.57	0.40	3.03	2.48

Table 3.3: Throughput (MOps) of KAYAK and ASFP under different workloads and SLO targets. "N/A" means the SLO target is infeasible.

The fast control loop algorithm is configured to run every 5ms and the slow control loop runs every 50ms. The initial RPC fraction is set at 100%, and we define SLO as the 99%-tile latency.

Workloads. We use the workload described in Section 3.2. Unless otherwise specified, we configure the workload with a traversal depth of two so that each request issues two data accesses to the storage. We vary the amount of computation that takes place after each access and refer to them as Light (100ns computation time per access), Medium (1μs per access) and Heavy (10μs per access). This workload emulates a variety of workloads with different computational load in a non-transactional environment.

We extend this workload and create a Bimodal workload. We denote by `Bimodal(1us, 100ns, 50%, 5s)`, a workload that consists of 50% Medium (1μs/RTT) and 50% Light (100ns/RTT) with an interval of 5 seconds.

We also run YCSB-T [72] as a transactional workload. This workload is not computationally intensive.

Unless otherwise specified, for all workloads, we set our latency SLO target as: 99%-tile request latency lower than or equal to 200μs.

Baseline. Our primary baseline is ASFP [55], which is available at `https://github.com/utah-scs/splinter/releases/tag/ATC'20` and also built on top of Splinter [133].

3.6.1 Convergence

In this section, we validate that KAYAK's fast loop can converge to a stable throughput R while satisfying SLO constraint and when running together with fast loop, the slow loop can also converge to the optimal RPC fraction.

Fast loop only. We first disable the slow loop and run KAYAK with a fixed RPC fraction (100%), to show that the fast loop (rate control) can converge to optimal throughput with different workloads.

We run Light, Medium and Heavy workloads with one application server and one storage server, and measure how the throughput and 99%-tile latency changes with time. As shown in Figure 3.9, KAYAK ramps up the throughput quickly when the measured 99%-tile request latency is below the SLO threshold of 200μs. Along with the increase of throughput, the latency also increases, as observed from the rise of red line. The entire converging process happens within 0.2 seconds.

After approaching the SLO limit, both the throughput and latency remains stable with minor fluctuations, confirming the convergence of our fast loop. We note that the converged throughput are the same as the measurements of the RPC-only configuration in Figure 3.4. This means that our fast loop indeed converges to the optimal throughput.

Dual loop control. Now we move on to verifying the convergence of both loops combined. We repeat the previous experiments, but with both control loops enabled. Figure 3.10 shows the dynamics of throughput and RPC fraction and how they change with time. We highlight three observations.

- Similar to Figure 3.9, throughput increases rapidly within the first 0.2 seconds; this is due to the fast loop.
- With the Medium and Heavy workloads, the throughput increase slows down after 0.2 seconds. This increase comes from the slow loop, as we can see a change in RPC fraction. Note that the Light workload does not show this trend, because in this setup the initial RPC fraction (100%) is already the optimal for it.
- After 1 second since the start, the throughput converges to a stable value with only minor fluctuations.

Comparing the RPC fraction in Figure 3.10 against Figure 3.5, we observe that our algorithm converges to the optimal RPC fraction. Comparing the throughput in Figure 3.10 against the Optimal configuration in Figure 3.4, we observe that the converged throughput is the optimal throughput.

Convergence under dynamic workloads. One advantage of KAYAK is that it can proactively adjust to changing workload. To verify this, we run KAYAK with the `Bimodal(1us, 100ns,`

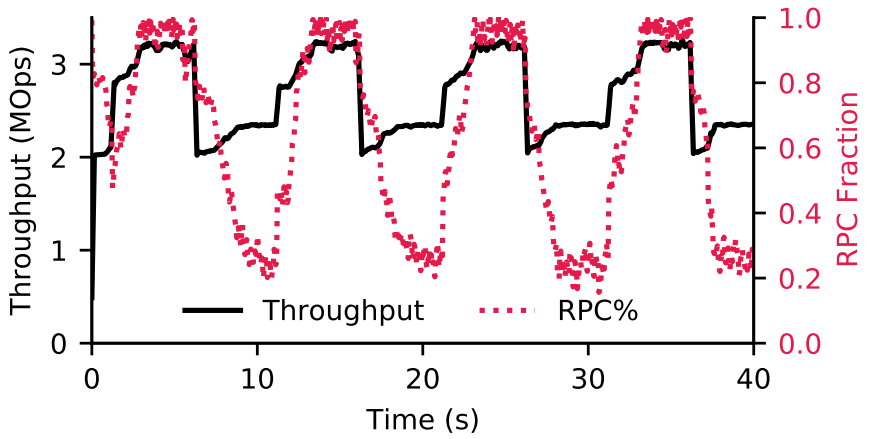


Figure 3.11: Dynamics of throughput and RPC fraction w.r.t time under Bimodal workload.

`50%, 5s)` workload. Figure 3.11 shows the dynamics of throughput and RPC fraction. As we can see, KAYAK adapts to the changing workload, and adjusts both the throughput and RPC fraction accordingly in a timely fashion.

3.6.2 Performance

Performance improvement. We compare the performance of KAYAK against the state of the art ASFP [55]. We use all the workloads: (i) Light/Medium/Heavy computational workload; (ii) Bimodal workload; and (iii) YCSB-T workload. We use one application server and one storage server, and vary the SLO target from 50μs to 400μs. The results are shown in Table 3.3, which we summarize as follows.

- For non-compute-intensive workloads (Light and YCSB-T), KAYAK achieves up to 12.2% throughput improvement. In this case, most of the requests are handled via RPC and KAYAK's opportunity for improvement is lower.
- For compute-intensive workloads, KAYAK achieves 32.5%-63.4% throughput improvement. In this case, the RPC fraction decreases, and ASFP's pushback mechanism kicks in; the overhead of pushing requests back in comparison to KAYAK's proactive placement increases the gap.

Overall, KAYAK outperforms ASFP because its proactive design is more efficient in using both application- and storage-side CPUs.

Workload	YCSB-T	Light	Medium	Heavy	Bimodal
Gap	5.4%	11.8%	6.7%	3.5%	10.6%

Table 3.4: KAYAK's performance gap from the upper bound for different workloads.

Gap between KAYAK and upper bound. We set the SLO target to be 200μs and compare the achieved throughput between using KAYAK and the upper bound obtained by the parameter sweep method (§3.2.2). We define the gap between KAYAK and the upper bound as follows:

$$Gap = \frac{Throughput_{max}}{Throughput_{max,Kayak}} - 1$$

Intuitively, the lower the gap is, the closer KAYAK is to the optimal. As shown in Table 3.4, KAYAK has a slowdown of 11.8% for computationally light workload and 3.5% for computationally heavy workload.

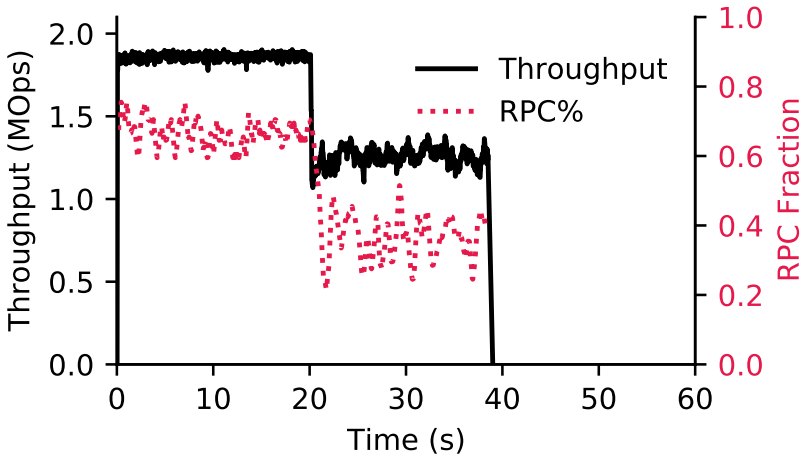
3.6.3 Fairness

In this section, we show that KAYAK can enforce max-min fairness when multiple tenants contend for server resources.

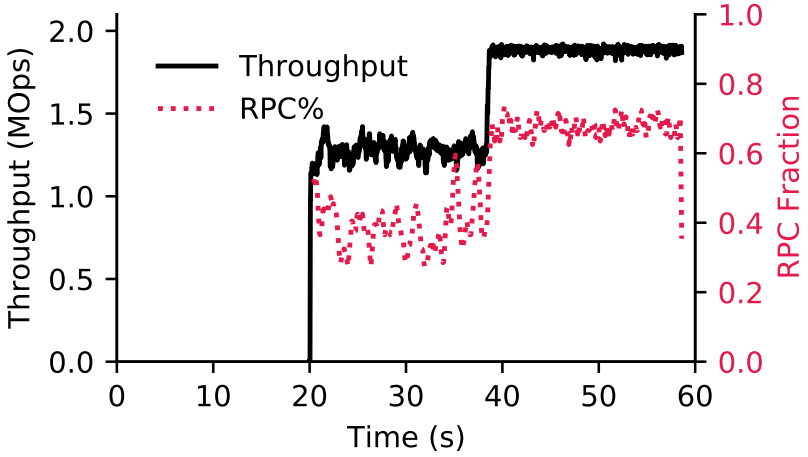
We run two tenants with the same setup and configure each to use four CPU cores instead of eight, while the server is still configured with eight CPU cores. Tenant A is started first, and after 20 seconds, tenant B is started. Figure 3.12a shows the dynamics of tenant A. During the first 20 seconds, tenant A quickly converges to the optimal throughput of around 1.82MOps. After 20 seconds when tenant B is started, the throughput achieved by tenant A drops to around 1.21MOps. Note that in this process, the optimal RPC fraction also shifts from 60% to 40%. This is because after tenant B joins, the server has less CPU resource to process tenant A's requests. After 40 seconds, tenant A stops sending requests.

Figure 3.12b shows the dynamics of tenant B, and we can see that when the two tenants are running together the achieved throughput is 1.21MOps and 1.24MOps, respectively. The gap between the two tenants is only 2.4%, which indicates that the server resources are fairly shared. After 40 seconds, the throughput and RPC fraction of tenant B increases because the storage server has more available CPU resources after tenant A stops.

Then we increase to four tenants and start the tenants one after one, with ten seconds in between.



(a) Tenant A



(b) Tenant B

Figure 3.12: Dynamics of throughput and RPC fraction w.r.t time of tenant A and B in the multi-tenant experiment.

Each tenant is configured with one CPU core, and runs a different workload. Tenant $\{1, 2, 3, 4\}$ runs {Light,Medium,Heavy,Bimodal}, respectively. We plot the occupied CPU cycles on the storage server for each tenant in Figure 3.13. When the 4 tenants are running together, we measure the Jain's Fairness Index [112] to be 0.9996.

3.6.4 Scalability

In this section, we verify that the KAYAK control loops are decoupled from data plane and do not limit KAYAK's scalability. To do so, we run experiments with Heavy workload, and vary the number of application and storage servers. For simplicity, we make sure a single request does not

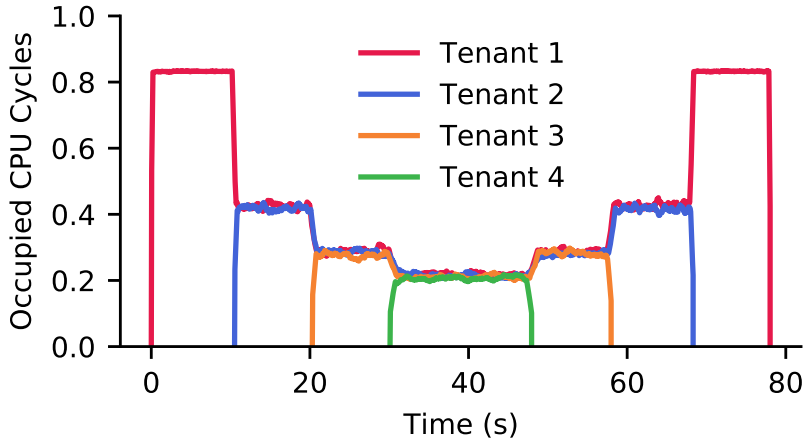
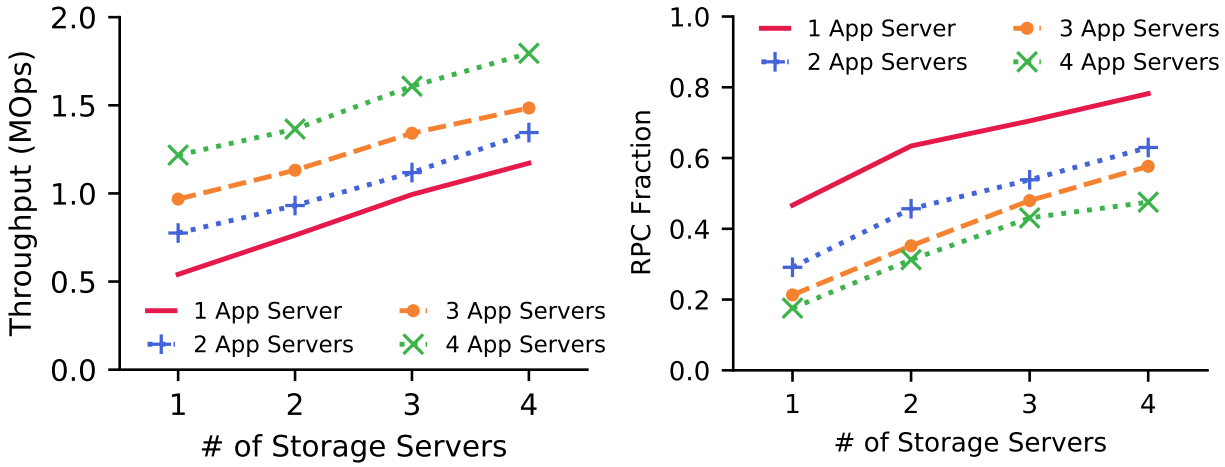


Figure 3.13: Fair-sharing of throughput for 4 applications sharing one storage server.



(a) Throughput

(b) RPC Fraction

Figure 3.14: Throughput and RPC Fraction of KAYAK with different server configurations.

access data from more than one storage servers. We measure the total throughput and the converged average RPC fraction. As shown in Figure 3.14a, throughput increases with both adding a storage and an application server. This is because adding either essentially adds more CPU cores to the system. Note that the RPC fraction increases with the increment of the storage servers but decreases with the increment of the application servers (Figure 3.14b). This is because KAYAK can judiciously arbitrate the requests and balance the load between the application and storage servers, by choosing the optimal RPC fraction.

3.6.5 Sensitivity Analysis

Initial state. First we evaluate whether KAYAK is sensitive to its initial state. We run four experiments using the Heavy workload, and vary the starting value for RPC fraction X from $\{0$, 0.25, 0.75, 0.100$\}$. As shown in Figure 3.15, KAYAK converges to the optimal request throughput and RPC fraction regardless of the initial RPC fraction in all four scenarios.

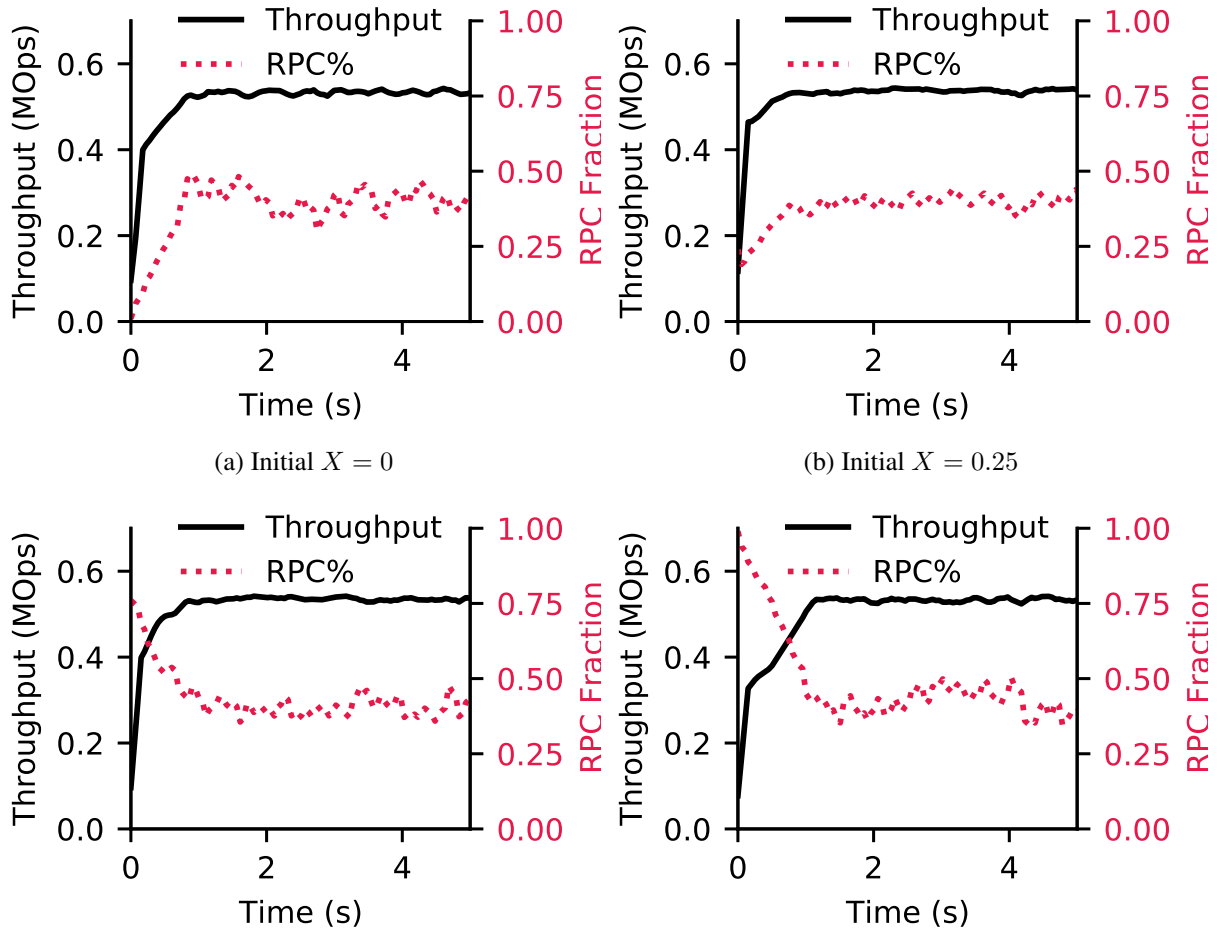
Choice of loop frequencies. In Section 3.4 we argue that in order for the dual loop control to work, we need to choose appropriate sampling frequencies for both loops. Here we analyze how different sampling frequencies affect the dynamics of our system.

We run two experiments using the Medium workload, and plot our results in Figure 3.16. In the first experiment we set the interval for both loop to be 200Hz; in the second experiment we invert the loop frequency so that the inner control loop runs slower than outer control loop. As shown in Figure 3.16a and 3.16b, the convergence quality degrades significantly in both cases. Hence, it is important to choose proper sampling frequencies to ensure that the inner control loop runs faster than the outer.

3.7 Related Work

Key-Value Stores. A recent line of research on key-value store has been focusing on utilizing RDMA to boost key-value store performance. Stuedi et al. [195] have achieved a 20% reduction in GET CPU load for Memcached using soft-iWARP without Infiniband hardware. Pilaf [160] uses only one-sided RDMA read to reduce CPU overhead of key-value store. In addition, it uses a verifiable data structure to detect read-write races. FaRM [74] proposes a new main memory key-value store built on top of RDMA. FaRM comes with transaction support but still support lock-free RDMA reads. HERD [118] further improves the performance of RDMA-based key-value store by focusing on reducing network round trips while using efficient RDMA primitives. FaSST [119] generalizes HERD and used two-sided RPC to reduce the number of QPs used in symmetric settings such as distributed transaction processing, improving scalability.

While there have been many research works focusing on improving raw performance of key-value stores, few investigates real performance implications on the application. TAO [57] is an



(a) Initial $X = 0$

(b) Initial $X = 0.25$

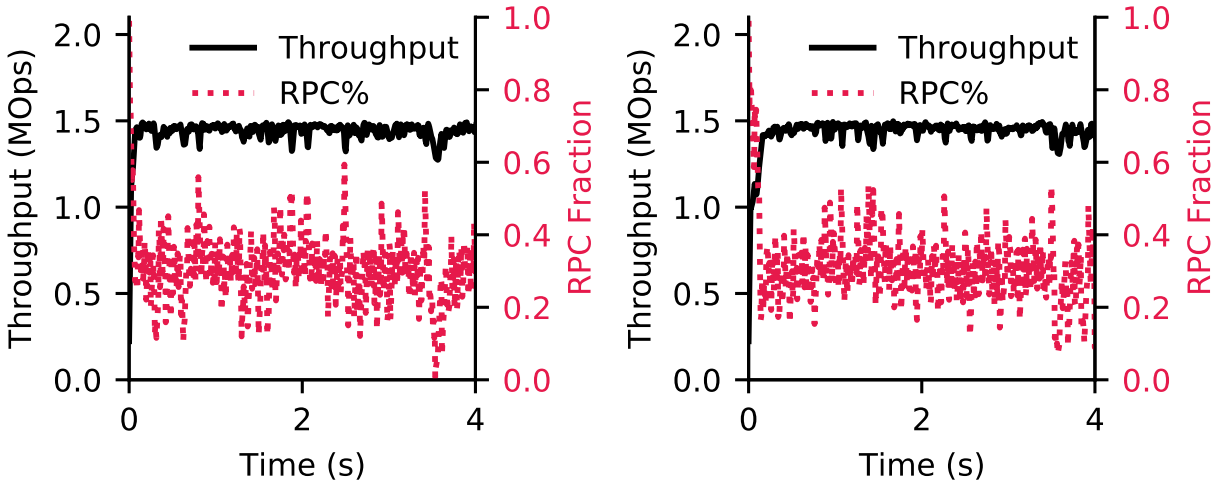
(c) Initial $X = 0.75$

(d) Initial $X = 1$

Figure 3.15: Throughput and RPC fraction during the converging process, with different starting RPC fraction.

application-aware key-value store by Facebook that optimizes for social graph processing. KAYAK builds on top of this concept and focuses on application-level objectives such as SLO constraint.

Storage-side computation. Storage-side computation (i.e. *shipping compute to data*) has made its way from latency-insensitive big data systems such as MapReduce [218, 3, 170] and SQL databases [29, 120, 32, 193, 192, 25, 26] into latency-critical KV stores [133, 34, 187, 79]. Comet [79] supports sandboxed Lua extensions to allow user-defined extensions to customize the storage by enabling application-specific operations. Malacology [187] utilizes Lua extensions contributed by users of the Ceph storage system [11], allowing installing and updating new object interfaces at runtime. Splinter [133] pushes bare-metal extension to storage server to allow RPC-like



(a) Equal loop frequency

(b) Inverse loop frequency

Figure 3.16: Dynamics of throughput and RPC fraction w.r.t time, with different loop frequencies: (a) both loops run at 200Hz; (b) inner loop at 20Hz, outer loop at 200Hz.

operations in addition to traditional key-value operations. These works breaks the assumption of dissagregated storage and necessitates the need for proactive arbitration provided by KAYAK.

Adaptive compute placement. An emerging line of research aims at adaptively balancing between client-side processing and server-side processing. ASFP [55] extends Splinter by reactively pushing back requests to the client side if the server gets overloaded, but at the cost of wasting CPU and network resource.Instead, KAYAK proactively balances the load exerted on both application and storage server. Cell [161] implements a B-tree store on RDMA supporting both client-side (RDMA-based) and server-side (RPC-based) search. Cell determines between these two schemes by tracking RDMA operation latency. This requires instrumentation into the application, which KAYAK avoids by measuring end-to-end request latency instead. A recent work called Storm [167] uses a reactive-adaptive approach similar to that of ASFP [55] but with a different policy, where for each request it will try the traditional KV API first, and switch to RPC API if it detects that the application is trying to chase the pointers.

3.8 Conclusion

In this project, we show that by proactively and adaptively combining RPC *and* KV together, overall throughput and CPU utilization can be improved. We propose an algorithm that dynamically adjusts the rate of requests and the RPC fraction to improve overall request throughput while meeting latency SLO requirements. We then prove that our algorithm can converge to the optimal parameters. We design and implement a system called KAYAK. Our system implementation ensures work conservation and fairness across multiple tenants. Our evaluations show that KAYAK achieves sub-second convergence and improves overall throughput by 32.5%-63.4% for compute-intensive workloads and up to 12.2% for non-compute-intensive and transactional workloads.

CHAPTER 4

ZEUS: Application-Aware Deep Learning Optimizations

4.1 Introduction

Deep neural networks (DNNs) have received ubiquitous adoption in recent years across many data-driven application domains such as computer vision [70, 96, 153], personalized recommendation [89, 97], and speech recognition [91]. As DNN models strive to become more accurate, their complexity, size, and overall demands for larger amounts of training data and computation resources are increasing too [189, 181]. As a result, training modern DNN models predominantly relies on highly parallel and increasingly more powerful GPUs [59, 162].

However, larger models and their growing computing demands ultimately translate to greater energy demands for training. For instance, training the GPT-3 model consumes 1,287 megawatt-hour (MWh) [172], which is equivalent to 120 years of electricity consumption for an average U.S. household [18]. This trend continues to grow as models become even larger: Meta reports an increasing electricity demand for AI, despite a 28.5% operational power footprint reduction [214]. Yet, existing literature on DNN training mostly ignores energy efficiency [186].

We observe that *common performance optimization practices for DNN training can lead to inefficient energy usage*. For example, many recent works propose using a large *batch size* to increase training throughput or goodput [84, 191, 176]. We show that increasing training throughput may come at the cost of lower energy efficiency. Similarly, modern GPUs allow a *GPU power limit* that dictates its maximum power draw, but existing solutions often ignore it. Our analysis of four generations of NVIDIA GPUs show that none of them are entirely power proportional, and drawing maximum power is not always ideal. Indeed, carefully choosing batch size and GPU power limit can reduce energy consumption by 23.8%–74.7% for diverse workloads (§4.2.2).

Unfortunately, reducing energy consumption is not entirely free. Indeed, we discover a tradeoff between energy consumption and training time for a given target accuracy – one must give when optimizing for the other (§4.2.3). Our characterization of the Pareto frontier highlights two notable phenomena. First, for a given training job, all Pareto-optimal configurations provide varying amounts of energy reductions in comparison to blindly using maximum batch size and GPU power limit. Second, the amount of reduction has a non-linear relationship with the loss of performance – while reducing one may increase the other, there exists an inflection point beyond which little savings cost disproportionately more. This raises a simple question: *how do we automatically identify and navigate the tradeoff between energy consumption and training time for DNN training?*

In this project, we present ZEUS to address this question. ZEUS is a plug-in optimization framework that automatically configures the batch size and GPU power limit to minimize the overall energy consumption and training time for DNN training jobs (§4.3). Developers can specify their preference for energy optimization, performance optimization, or their combination through a single knob. Unlike some recent works that only consider GPU-specific configurations [197, 54], ZEUS simultaneously considers job- and GPU-related configurations. It does not require per-job offline profiling or prediction model training either[207, 222], both of which can be prohibitive in large clusters with heterogeneous hardware and time-varying workloads [212]. Instead, ZEUS provides an online solution that is tailored to the characteristics of DNN training workflows. That is, as new data flows into the pipeline, the model needs to be periodically re-trained [95], manifesting as *recurring jobs* in production clusters [212]. Leveraging this fact, ZEUS automatically explores various configurations, measures corresponding gains or losses, and continuously adjusts its actions based on its measurement.

ZEUS takes an online exploration-exploitation approach that leverages such recurrence to minimize the overall energy-time cost of recurrent DNN training tasks (§4.4). It addresses two major sources of uncertainty to find an effective solution. First, the energy consumption of a training job varies even when the same job is run on the same GPU with the same configuration. This is due to randomness introduced into model initialization and data loading [185, 66]. Second, both DNN models and GPUs have diverse architectures and unique energy characteristics [211]. As a result, information acquired from profiling the energy consumption of specific models and GPUs offline do not generalize. To this end, we design a just-in-time (JIT) energy profiler which is triggered

by the training job online and efficiently captures the energy characteristics of it within just a few epochs. Moreover, ZEUS adopts a multi-armed-bandit formulation with Thompson Sampling [198], which allows us to capture the stochastic nature of DNN training and optimize under uncertainty with bounded regret [44, 43].

We have implemented ZEUS and integrated it with PyTorch [171] (§4.5). Our evaluation (§4.6) on a diverse workload consisting of speech recognition, image classification, NLP, and recommendation tasks shows that ZEUS reduces energy consumption by 15.3%–75.8% and training time by 60.6% w.r.t. simply selecting the maximum batch size and maximum GPU power limit. ZEUS converges to optimal configurations quickly and can adapt to data drift effectively. ZEUS's benefits expand to multi-GPU settings as well.

In summary, we make the following contributions:

- To the best of our knowledge, we are the first to characterize the energy consumption vs. performance tradeoff for DNN training in terms of job- and GPU-specific configuration parameters.
- We present an online optimization framework that can learn from and adapt to workload dynamics over time.
- We implement and evaluate the optimizer in ZEUS that integrates with existing DNN training workflows with little code change and negligible overhead, while enabling large benefits.

4.2 Motivation

In this section, we present an overview of energy consumption characteristics of DNN training on GPUs, opportunities for reducing energy consumption, and conclude with characterizing the tradeoff between reducing energy consumption and improving training performance.

4.2.1 DNN Training

Modern DNNs are trained by going over a large dataset multiple times, where each pass over the dataset is termed an *epoch* [83]. One epoch of training consists of thousands of *iterations* of gradient descent over equally sized mini-batches, with the *batch size* affecting model accuracy,[1] training

[1] In this project, we specifically target the *validation accuracy* of the model, which captures how well the model performs on unseen data.

throughput, and energy consumption. The performance of DNN training is often measured in terms of time-to-accuracy (TTA) for a given target accuracy [66], and increasing training throughput (or more precisely goodput [176]) leads to lower TTA.

As modern DNN models grow in size/complexity and are trained on increasingly larger datasets [189, 181], they are predominantly trained on increasingly more powerful GPUs, consuming more energy in the process [172, 214]. Recent benchmarks show that GPUs are responsible for around 70% of the total energy consumption during DNN training [99].

In production GPU clusters, as new data flow into the deep learning pipeline, DNNs need to be periodically re-trained [95]. This need manifests itself in the form of *recurring jobs*. Indeed, as reported in a recent cluster trace [212], more than 50% of the jobs recur more than 35 times over a two-month period (see Appendix B.3).

4.2.2 Opportunities for Improving Energy Efficiency

We focus on two job and hardware configurations that can cause sizable energy inefficiency in DNN training: (1) choice of batch size for each mini-batch; and (2) leaving the GPU at the maximum power limit (default configuration).

Impact of batch size on energy efficiency The size of each mini-batch during DNN training (batch size) determines how many samples are processed in one iteration. The higher it is, the faster we can go over the entire input dataset. It is typically set to the maximum value that fits in the GPU VRAM.

We observe across diverse DNN training workloads that choosing a batch size simply to increase throughput can lead to more energy consumption for the same target accuracy. Specifically, we performed a sweep over a large range of valid batch sizes (from 8 to the maximum batch size that fits in GPU memory) for six computer vision (CV), natural language processing (NLP), and speech workloads on an NVIDIA V100 GPU (Figure 4.1).[2] Section 4.6.1 provides details on workloads and methodology. We find that the energy-optimal batch size (Batch Size Opt. in Figure 4.1) can lead to 3.4%–65.0% lower energy consumption than the throughput-optimal one for the same target accuracy.

[2]We measure GPU power consumption by invoking the NVML [30].

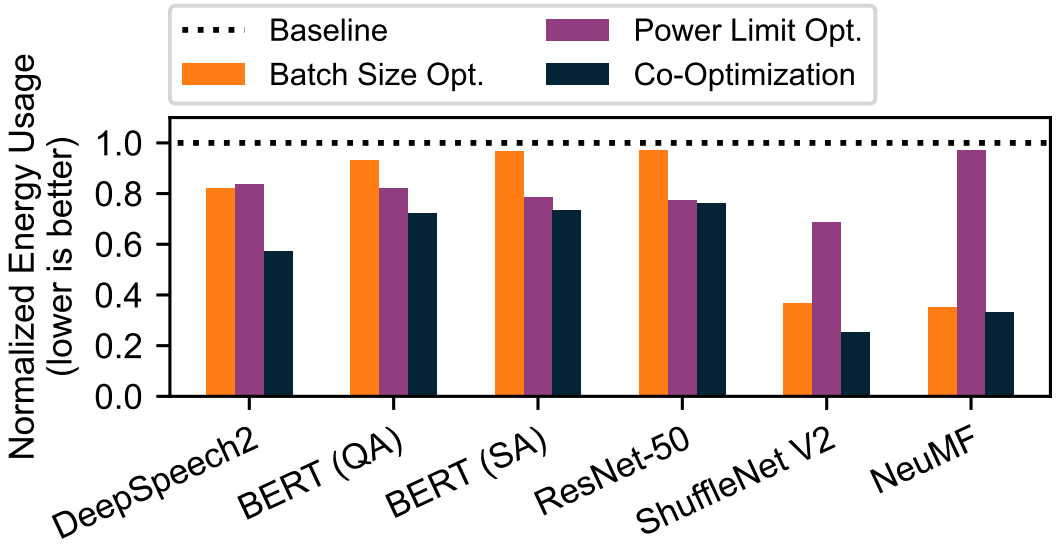
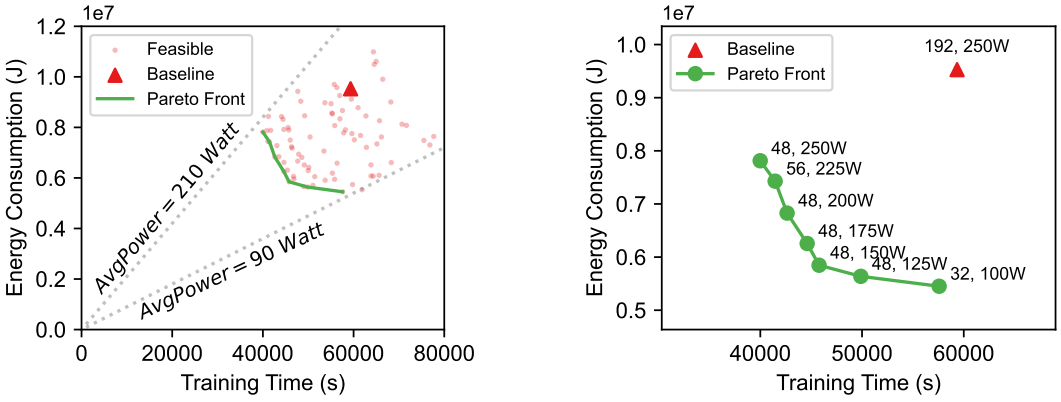


Figure 4.1: Energy usage normalized against baseline for DNN training, measured on NVIDIA V100 GPU. Baseline uses maximum power limit and throughput-optimal batch size.



(a) Energy-Time Tradeoff

(b) Pareto Front Zoom-in

Figure 4.2: DeepSpeech2 trained with LibriSpeech on NVIDIA V100: (a) ETA vs. TTA. The red dots indicate all feasible configurations. The two gray dotted lines indicate two boundaries characterized by average power consumption. The green line indicates the Pareto frontier over all configurations. (b) Zoom-in view on the Pareto frontier, with batch size and power limit annotated on each data point.

Impact of GPU power limit on energy efficiency GPU power consumption is affected by the frequency and voltage of the GPU core and VRAM on board [210, 121], which is regulated by the dynamic voltage and frequency scaling (DVFS) algorithm [159]. The default DVFS policy always sets the GPU core and memory frequency to its maximum, regardless of the workload. We

performed a sweep over a wide range of GPU power limits[3] for the aforementioned setup. We found that the optimal energy consumption (Power Limit Opt. in Figure 4.1) may happen at a lower power limit than the maximum and can reduce energy consumption by 3.0%–31.5%.

Joint optimization As Figure 4.1 shows, we can achieve even more energy savings (23.8%–74.7% reduction) if we jointly optimize both of these configurations. Note that we observed similar opportunities for reducing energy consumption for other generations of GPUs as well (Figure B.1 in Appendix B.1).

4.2.3 Energy-Performance Tradeoffs

Opportunities for reducing energy consumption during DNN training comes with a cost. When optimized for efficiency, DNN training performance (TTA) may be impacted. In the following, we characterize the tradeoff.

We define the total energy consumption of a DNN training job to reach its target accuracy as its *energy-to-accuracy* (ETA). Similar to TTA, ETA focuses on the process of end-to-end training. Their relationship can be captured as follows:

$$\texttt{ETA}(b, p) = \texttt{TTA}(b, p) \times \texttt{AvgPower}(b, p), \tag{4.1}$$

where p denotes the GPU power limit, b the batch size, and $\texttt{AvgPower}(b, p)$ the average power consumption during training with configuration (b, p). Note that the average power consumption is not the same as the GPU power limit.

It is worth noting that when changes in configuration (b, p) lead to increase of TTA, ETA does not always follow because $\texttt{AvgPower}(b, p)$ can decrease. Instead, there exists a tradeoff between ETA and TTA.

Tradeoff between ETA and TTA We characterize and elaborate on this tradeoff using DeepSpeech2 on LibriSpeech as an example (Figure 4.2). It shows a scatter plot of (TTA, ETA) for the

[3]From the minimum to the maximum power limit allowed by NVIDIA System Management Interface [31]; from 100W to 250W for NVIDIA V100.

experiments in Section 4.2.2. We observe similar results for other workloads as well (Figure B.2 in Appendix B.2).

Let us start with Figure 4.2a first, where each data point denotes the (TTA, ETA) of training the model for a certain configuration until it reaches target accuracy. While sweeping the configurations, we focus on the boundary of all feasible (TTA, ETA) pairs. We find them to be bounded by two straight lines characterizing the average GPU consumption. When the GPU is under heavy load, the data points of (TTA, ETA) appear closer to the upper limit of GPU power consumption, which in this workload is about 210W. On the other hand when GPU is nearly idle, the data points appear closer to the idle power consumption of 90W. More importantly, we find a curve along which all (TTA, ETA) pairs achieves Pareto optimality [60], for which we cannot improve ETA without sacrificing TTA, and vice versa.

Now let us take a closer look at the Pareto frontier in Figure 4.2b, with the configurations used during training annotated along each data point. We highlight two takeaways:

1. These results clarify why baseline configurations can lead to suboptimal energy efficiency (§4.2). Moreover, it shows that blindly going for high batch size and power limit configurations can lead to suboptimal TTA as well.
2. There exists a tradeoff between ETA and TTA, with different optimums for each. The configuration optimizing the ETA (b =32, p =100W) is different from that optimizing TTA (b =48, p =250W).

4.3 ZEUS Overview

ZEUS is an optimization framework that navigates the ETA-TTA tradeoff by automatically configuring the batch size and GPU power limit of recurring DNN training jobs. It enables developers to optimize energy and/or performance metrics using a single knob.

4.3.1 Optimization Metric

Defining a good cost metric for developers to express their priority in this tradeoff is critical for designing ZEUS. We propose a simple cost metric:

$$C(b,p) = \eta \cdot \texttt{ETA}(b,p) + (1-\eta) \cdot \texttt{MAXPOWER} \cdot \texttt{TTA}(b,p) \tag{4.2}$$

Here η is a user-defined parameter to specify the relative importance of energy efficiency and training performance (training throughput). When $\eta = 0$, we are only optimizing for time consumption, whereas when $\eta = 1$, we are only optimizing for energy consumption. `MAXPOWER` is the maximum power limit supported by the GPU used. This is introduced to unify the units of measure in this equation, so that any $0 < \eta < 1$ can strike a balance between ETA and TTA.

Note that this cost metric can potentially be extended beyond a simple linear function to a user-defined function.

4.3.2 Challenges in Picking the Optimal Configuration

Combining Equation 4.1 and 4.2, we have:

$$C = (\eta \cdot \texttt{AvgPower}(b,p) + (1-\eta) \cdot \texttt{MAXPOWER}) \cdot \texttt{TTA}(b,p). \tag{4.3}$$

Picking the optimal configuration(s) to minimize the energy-time cost C for DNN training is challenging because the search space $[b \times p]$ is large and obtaining the cost of each configuration is difficult. This is because both the average power consumption $\texttt{AvgPower}(b,p)$ and DNN training duration `TTA` are hard to predict without actually running a configuration to completion, as explained below.

- **Complex power consumption model:** The total energy consumption of a GPU is affected in a non-linear fashion by both the characteristics of a workload such as the number of instructions and memory accesses, as well as GPU hardware configurations such as the frequency and voltage of the GPU core and VRAM on board [121, 48]. Existing efforts estimate GPU energy consumption based on instruction- or kernel-level information [102, 152], which is hard to get before training completes. These energy-prediction models are architecture-specific and static

too.

- **Stochastic nature of DNN training:** Modeling and predicting the duration for training a specific model to target accuracy (TTA) is known to be difficult in practice [86]. Moreover, randomness introduced into model initialization and data loading leads to variations of TTA, even when the same job is run on the same GPU with the same configuration – TTA variations can be as large as 14% [66].

Fortunately, DNN training jobs often recur in production [212]. This provides opportunities for empirical estimation through repeated measurements across recurrences of the same training job.

4.3.3 Architectural Overview

At a high-level, ZEUS takes a novel online exploration-exploitation approach to minimize the aggregate cost of recurrent DNN training jobs. ZEUS resolves the aforementioned challenges with three key design choices:

1. A fully online solution without offline profiling, allowing ZEUS to immediately begin optimizing incoming jobs.
2. A just-in-time (JIT) online profiler, which efficiently profiles the energy characteristics of the training job online.
3. Adopting multi-armed-bandit (MAB) with Thompson sampling, which allows us to embrace the stochastic nature of DL training and optimize under uncertainty with bounded cost difference with the optimal choice while also adapting to changing workloads such as data drift.

Workflow of ZEUS Figure 4.3 shows an overview of the high-level workflow of ZEUS. In a production environment, users submit ❶ recurrent DNN training jobs (a tuple of data, model, optimizer, and the target model accuracy) to ZEUS, along with a set of feasible batch sizes $\mathcal{B}$ and power limits $\mathcal{P}$ to explore. ZEUS then predicts ❷ the optimal batch size and power limit configuration based on past execution history, and launches ❸ the training job with such configuration. During and after the training process, statistics about DNN training (e.g., validation accuracy) and GPU power consumption are collected and fed back ❹ to the ZEUS optimizer. ZEUS optimizer learns from the feedback and adjusts the cost prediction model. The training job will be terminated upon either

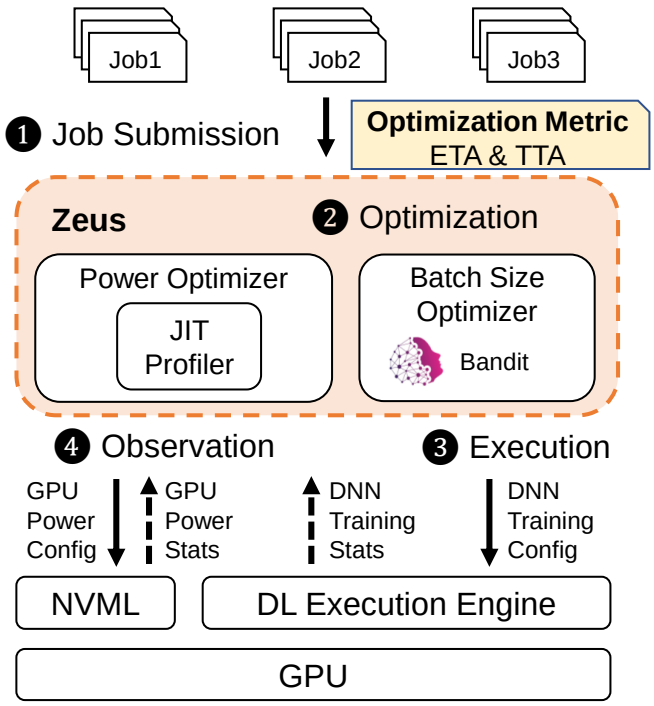


Figure 4.3: ZEUS Workflow.

reaching target accuracy or exceeding a predefined stopping threshold before reaching target. The whole process is an automated feedback loop that minimizes the key objective of energy-time cost.

Building ZEUS requires both algorithm designs and systems support. Next we describe the core optimization algorithm details (§4.4) and ZEUS implementation highlights (§4.5).

4.4 ZEUS Algorithm Design

In this section, we delve into the details of how ZEUS selects the best batch size and GPU power limit to optimize the overall cost of recurrent DNN training tasks. We first present the optimization problem formulation and how we decouple the optimizations of batch size and power limit (§4.4.1). Next, we show how to optimize power limit (§4.4.2) and batch size (§4.4.3) under the decoupled framework. We conclude by discussing how we address common challenging scenarios (§4.4.4).

4.4.1 Problem Formulation

The objective of ZEUS is to minimize the cost of a recurring job by automatically exploring the feasible set of batch sizes $\mathcal{B}$ and power limits $\mathcal{P}$. In essence, we neither want to incur too much cost searching for the optimal configuration, nor do we want to miss it. Minimizing the *cumulative* cost of the job over recurrences captures the implicit trade-off between exploration and exploitation. Put formally in terms of the cost function defined by Equation (4.2), our objective becomes

$$\begin{aligned} \min_{b,p} \quad & \sum_{t=1}^{T} C(b_t, p_t; \eta) \\ \text{s.t.} \quad & b_t \in \mathcal{B}, p_t \in \mathcal{P}, \forall t \in [1, T], \end{aligned} \tag{4.4}$$

where b_t and p_t denote the batch size and power limit chosen at the tth recurrence of the job, respectively.

This is a challenging problem without modification, mainly because the size of the search space can be in the order of hundreds, and each value of $C(b, p; \eta)$ inside the search space can only be obtained by running DNN training until it reaches the target metric. However, further expanding the cost function from Equation (4.3) allows us to *decouple* explorations of batch size and power limit, making the problem more tractable:

$$\begin{aligned} & C(b, p; \eta) \\ & = (\eta \cdot \texttt{AvgPower}(b, p) + (1 - \eta) \cdot \texttt{MAXPOWER}) \cdot \texttt{TTA}(b, p) \\ & = \texttt{Epochs}(b) \cdot \frac{\eta \cdot \texttt{AvgPower}(b, p) + (1 - \eta) \cdot \texttt{MAXPOWER}}{\texttt{Throughput}(b, p)}. \end{aligned} \tag{4.5}$$

where $\texttt{Epochs(b)}$ denotes the number of epochs needed to reach the target, and $\texttt{Throughput(b, p)}$ epochs per second.

We find two key insights that allow the decoupling of batch size b and power limit p:

1. Given b, $\texttt{AvgPower}(b, p)$ and $\texttt{Throughput}(b, p)$ can be profiled quickly during training for all possible choices of p. This is due to the iterative nature of DNN training, yielding stable power and throughput estimations even with a small number of iterations.
2. $\texttt{Epochs}(b)$ is not affected by the choice of p as changing the power limit does not change what

is computed.

This implies that the optimal power limit, given any batch size, can be determined independently based on online profiling. Moreover, since any choice of batch size is automatically accompanied by the optimal power limit, our search space is reduced to the set of batch sizes $\mathcal{B}$.

Formally put, we have decoupled the problem in Equation (4.4) into an equivalent two-level optimization problem

$$\min_{b \in \mathcal{B}} \sum_{t=1}^{T} \texttt{Epochs}(b_t) \cdot \texttt{EpochCost}(b_t; \eta) \tag{4.6}$$

where

$$\begin{aligned} &\texttt{EpochCost}(b_t; \eta) \\ &= \min_{p_t \in \mathcal{P}} \frac{\eta \cdot \texttt{AvgPower}(b_t, p_t) + (1-\eta) \cdot \texttt{MAXPOWER}}{\texttt{Throughput}(b_t, p_t)}. \end{aligned} \tag{4.7}$$

When a job arrives, ZEUS will first decide which batch size to use based on Equation (4.6) (§4.4.3). Then, based on the batch size, ZEUS will pick the optimal power limit based on Equation (4.7) (§4.4.2).

4.4.2 Optimizing the Power Limit

We start with how ZEUS determines the optimal power limit based on Equation (4.7), given a choice of the batch size. As highlighted earlier, We leverage the iterative nature of DNN training and the recurrent nature of jobs in production DNN training workflows.

When a job with batch size decision b is submitted, our just-in-time (JIT) profiler is triggered and checks if this batch size had been profiled before. For a batch size b that was never tried, it profiles $\texttt{AvgPower}(b, p)$ and $\texttt{Throughput}(b, p)$ for all possible power limits p during the first epoch of the job by partitioning the epoch into slices at iteration boundaries and dynamically changing the GPU power limit for each slice. The profile information is fed back to ZEUS, and the optimal power limit of the batch size is determined by solving Equation (4.7). The rest of the epochs are executed with the optimal power limit. Our *online* JIT profiling approach consumes strictly less time and energy compared to offline profiling before running the job, because the profiling process itself contributes to training without affecting its accuracy.

4.4.3 Optimizing the Batch Size

Now we focus on how ZEUS determines the batch size b_t for each job recurrence t that optimizes Equation (4.6). As seen in Section 4.4.2, $\texttt{EpochCost}(\texttt{b}_\texttt{t}; \eta)$ is a cheap and deterministic function that identifies the optimal power limit for any batch size b_t and returns the optimal cost of one epoch. Thus, we may limit our exploration to choosing the optimal batch size because whichever batch size we choose, the optimal power limit will accompany it.

Due to the unpredictable and stochastic nature of DNN training, picking out the optimal batch size without adequate exploration is difficult. Hence, a good solution must (1) incorporate such nature of DNN training into its exploration process, and (2) intelligently tradeoff the cost of exploring for potentially better batch sizes and the gain of exploiting batch sizes that are already known to be good.

Grid search is suboptimal We argue that exhaustively going through all batch sizes and selecting the one with the smallest cost is still suboptimal due to the stochastic nature of DNN training. That is, because the cost of a DNN training job can differ even when executed with the exact same configurations, it must be modeled as a *cost distribution* with unknown mean and variance. Although performing several trials for each batch size may yield a better estimation of the mean cost, such a strategy leads to *high exploration cost* because it does not quickly rule out obviously suboptimal batch sizes.

Multi-Armed Bandit formulation ZEUS aims to explore the cost of different batch sizes and converge to the optimal batch size, while not incurring too much exploration cost.

ZEUS formulates the problem as a multi-armed bandit (MAB) with T trials and B arms, where each trial corresponds to a recurrence of the job and each arm to a batch size in $\mathcal{B}$. MAB is a good fit to our problem scenario in that it models each arm's cost as a probability distribution, and it balances exploration and exploitation as it estimates the parameters of the cost distribution in each arm. Here, the cost of each arm is modeled as a Gaussian distribution [184] due to its representational flexibility.

Input: Batch sizes $\mathcal{B}$
Belief posterior parameters $\hat{\mu}_b$ and $\hat{\sigma}_b^2$
Output: Batch size to run b^*

Function `Predict` ($\mathcal{B}$, $\hat{\mu}_b$, $\hat{\sigma}_b^2$) **:**
1 **foreach** batch size $b \in \mathcal{B}$ **do**
/* Sample from the belief distribution */
2 Sample $\hat{\theta}_b \sim \mathcal{N}(\hat{\mu}_b, \hat{\sigma}_b^2)$
/* Select the arm with smallest mean cost sample */
3 $b^* \leftarrow \arg\min_b \hat{\theta}_b$

Pseudocode. 4: Gaussian Thompson Sampling: Choosing the next batch size to run (`Predict`)

The objective of the MAB formulation is to minimize the *cumulative cost regret* defined as

$$\sum_{t=1}^{T} \texttt{Regret}(b_t; \eta) \tag{4.8}$$

where the regret of choosing b_t in the is defined as

$$\begin{aligned} &\texttt{Regret}(b_t; \eta) \\ &= \texttt{Epochs}(b_t) \cdot \texttt{EpochCost}(b_t; \eta) - \min_{b,p} \ \texttt{Cost}(b, p; \eta) \end{aligned} \tag{4.9}$$

Minimizing cumulative cost regret aligns with our objective in Equation (4.6).

Thompson Sampling We adopt the Thompson Sampling [184] policy for the MAB formulation to tradeoff exploration and exploitation, not only because it is known to perform well in practice [62, 184] and had successful adoption recently [156, 144], but also because its modeling assumptions fit our problem scenario well.

At a high level, Thompson Sampling is an online procedure that refines its *belief* about the *mean cost* of each arm (batch size) based on experience. At each recurrence, the belief is used to pick the arm with the lowest estimated mean cost (Algorithm 4), and the belief is updated based on the actual cost observed (Algorithm 5).

Specifically, the cost distribution is modeled as a Gaussian distribution with unknown mean θ_b. Then, the belief about θ_b is modeled with its conjugate prior distribution, which is also a Gaussian

Input: Batch size b and observed cost C
Previous cost observations C_b for b
Belief prior parameters $\hat{\mu}_0$ and $\hat{\sigma}_0^2$
Output: Belief posterior parameters $\hat{\mu}_b$ and $\hat{\sigma}_b^2$

Function `Observe` (b, C, $\mathcal{C}_b$, $\hat{\mu}_0$, $\hat{\sigma}_0^2$) **:**
/* Add the most recent cost to history */
1 $\mathcal{C}_b \leftarrow \mathcal{C}_b \cup \{C\}$
/* Compute the variance of the cost */
2 $\tilde{\sigma}^2 \leftarrow Var\left(\mathcal{C}_b\right)$
/* Compute the belief distribution's posterior variance */
3 $\hat{\sigma}_b^2 \leftarrow \left(\frac{1}{\hat{\sigma}_0^2} + \frac{|C_b|}{\tilde{\sigma}^2}\right)^{-1}$
/* Compute the belief distribution's posterior mean */
4 $\hat{\mu}_b \leftarrow \hat{\sigma}_b^2 \left(\frac{\hat{\mu}_0}{\hat{\sigma}_0^2} + \frac{Sum(\mathcal{C}_b)}{\tilde{\sigma}^2}\right)$

Pseudocode. 5: Gaussian Thompson Sampling: Updating the belief distribution (`Observe`)

distribution [75]. That is, $\theta_b \sim \mathcal{N}(\hat{\mu}_b, \hat{\sigma}_b^2)$. Here it is important to note that $1/\hat{\sigma}_b^2$ can be thought as of how confident Thompson Sampling is in its belief about that arm, with the confidence increasing as it accumulates more observations of the cost of choosing that arm. Then, Thompson Sampling automatically balances exploration and exploitation by choosing the arm with the smallest mean cost sample $\hat{\theta}_b \sim \mathcal{N}(\hat{\mu}_b, \hat{\sigma}_b^2)$ (Algorithm 4). With low confidence (high variance), $\hat{\theta}_b$ will be dispersed across a larger range of potential costs, having a higher chance of getting chosen even if some of its initial observations showed high cost. In contrast, when the arms observed a lot of cost samples and the confidence is high (low variance), $\hat{\theta}_b$ is likely to be centered around the mean observed cost, allowing the exploitation of arms that are known to be good. After the actual cost of an arm is observed, the belief parameters of that arm is updated using the Bayes Rule [184] (Algorithm 5).

The belief prior parameters $\hat{\mu}_0$ and $\hat{\sigma}_0^2$ reflect prior belief about the mean cost of using the batch size for training and the confidence of such belief. Hence, the choice of prior parameters serve as a way to initialize the arms such that they reflect prior knowledge about the cost of each arm. If such information is not available, which is our default assumption, it is also possible to initialze the arms with a flat prior that assumes no prior knowledge – in our case, this is a Gaussian distribution with zero mean and infinite variance.

In contrast to grid search, our formulation using MAB and Thompson Sampling meets the two

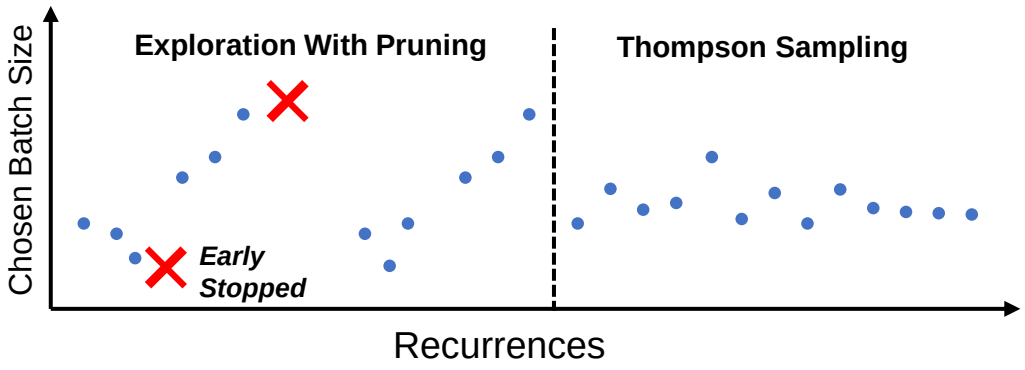


Figure 4.4: An example of batch size pruning during exploration stage. Each point is a recurrence.

requirements mentioned earlier. That is, MAB inherently incorporates the stochastic nature of DNN training in that it models cost as a probability distribution. Moreover, Thompson Sampling can quickly rule out batch sizes that are obviously suboptimal because the probability of a smaller mean cost being sampled from an arm that observed noticeably large cost is low.

4.4.4 Extensions

Handling unknown cost variance Unlike conventional Gaussian Thompson Sampling applications, we may not assume that the variances of the cost of each arm are known. That is, the cost variance (i.e., how much the cost will fluctuate even when training is run with the same batch size) is not known before any observation. Moreover, the cost variance depends not only on the batch size, but also on the DNN's robustness to the randomness in parameter initialization and data loading, making it difficult to quantify at the time the MAB is constructed. Hence, our approach is to *learn* the cost variance as we observe cost samples (Line 2 in Algorithm 5).

Handling stragglers during exploration There may be cases where an exploratory job does not reach the target metric within a reasonable amount of time, especially during the earlier exploration stage. To handle this, we employ *early stopping* and *pruning*. The intuition is that if a batch size does not reach the target metric even after incurring an exceedingly large cost, it is highly unlikely to be the optimal one.

For early stopping, we define a cost threshold $\beta \cdot \min_t C_t$, meaning that when the cost of the current job is to exceed β times the minimum cost observed so far, we stop the job and retry with another batch size. Here β is a parameter to account for the stochastic nature of DL training. By

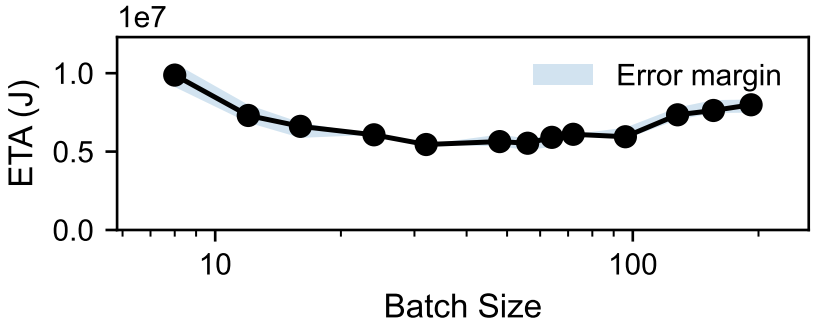


Figure 4.5: ETA of each batch size for DeepSpeech2 trained on LibriSpeech. Plots for rest of the workloads are in the Appendix B.4.

default we choose $\beta = 2$, with which we should be able to tolerate variations of TTA between different runs of the same configuration, which is usually less than the 14% [66].

For pruning, as illustrated in Figure 4.4, we begin with the default batch size provided by the user and first try smaller batch sizes until we meet the minimum batch size or a batch size that fails to reach the target metric before the early stopping threshold. The same process is repeated for batch sizes larger than the default batch size. Then, only the batch sizes that reached the target metric are kept in the batch size set we explore. After performing an initial round of pruning, the default batch size is updated to be the one with the smallest cost observed, and we performing pruning once more starting from the new default batch size.

The intuition behind our batch size pruning approach is the convexity we observe in the BS-ETA curve around the optimal batch size (See Figure 4.5). Moreover, pruning allows ZEUS to quickly rule out batch sizes that are noticeably suboptimal (typically too large, leading to more training epochs and loss of accuracy [125, 82], or too small, yielding gradients that are too noisy [183]), thus cutting down the cost of exploration.

The overall process is depicted in Algorithm 6.

Handling concurrent job submissions Classic multi-armed bandit scenarios assume that the MAB immediately observes the cost of pulling an arm. However, in a DNN training cluster, recurring jobs may overlap in their execution when a later job starts before the completion of an earlier job. In this case, the MAB does not get to observe the cost of the earlier job at the time it has to decide the batch size for the later job. For deterministic policies like [142, 51], this leads to duplication exploration of the same batch size back-to-back, reducing the efficiency of exploration.

Input: Set of batch sizes $\mathcal{B}$
Default batch size b_0
Belief prior parameters $\hat{\mu}_0$ and $\hat{\sigma}_0^2$

/* Pruning phase */
Recurrence $t \leftarrow 0$
repeat *2* **times**
    Explore b_0
    Explore $b < b_0$ until convergence failure
    Explore $b > b_0$ until convergence failure
    $\mathcal{B} \leftarrow \{b : b \text{ converged}\}$
    $b_0 \leftarrow b$ with smallest cost observed
    $t \leftarrow t + |\mathcal{B}|$
/* Thompson Sampling phase */
while $t \leq T$ **do**
    $b^* \leftarrow Predict(\mathcal{B}, \hat{\mu}_b, \hat{\sigma}_b^2 \; \forall b \in \mathcal{B})$
    Run job with batch size b^* and add cost to $\mathcal{C}_b$
    /* Update our belief of the mean cost */
    $\hat{\mu}_b, \hat{\sigma}_b^2 \leftarrow Observe(b, \mathcal{C}_b, \hat{\mu}_0, \hat{\sigma}_0^2)$
    $t \leftarrow t + 1$

Pseudocode. 6: Gaussian Thompson Sampling Batch Size Optimizer.

However, Thompson Sampling naturally mitigates this problem without modification because deciding the next batch size to explore (`Predict`) is a random function. That is, because Thompson Sampling *samples* the estimated mean cost from each arm's belief distribution and returns the arm with the lowest sampled value, concurrent jobs have a high probability of running different batch sizes even if there was no information gained between the invocations of `Predict`. This is especially the case during the early stage of Thompson Sampling when the arms' belief distributions have large variances (low confidence), losing little exploration efficiency.

During the short initial exploration phase, we run concurrent job submissions with the best-known batch size at that time. As the best batch size constantly updates throughout the exploration stage, this strategy fairly distributes the additional exploration opportunities from concurrent job submissions to batch sizes that are known to converge. We evaluate ZEUS's efficacy on handling concurrent job submissions in Section 4.6.3.

```python
from zeus import ZeusDataLoader

train_loader = ZeusDataLoader(
    train_set, batch_size, max_epochs, target_metric)
eval_loader = ZeusDataLoader(eval_set, batch_size)

for epoch in train_loader.epochs():  # may early stop
    for batch in train_loader:
        # Learn from batch
    for batch in eval_loader:
        # Evaluate on batch
    train_loader.report_metric(validation_metric)
```

Listing 4.1: ZEUS Integration Example

Handling data drift In production training clusters, the data on which the model is trained shifts, which is one of the reasons why re-training is triggered [151, 147]. The implication of drift in the perspective of the MAB is that the cost distribution of each arm is non-stationary.

Thompson Sampling allows a simple modification that allows us to handle non-stationary cost distributions. Since older cost observations become less and less relevant, we only operate on a window of N most recent cost observations [53], and the belief distributions will not take old observations into account. When old history entries are evicted, computing the new parameters of the arm is also cheap thanks to the conjugate prior property. This way, ZEUS transparently adapts to data drifts in an *online* manner, as we show in Section 4.6.4.

4.5 ZEUS Implementation

ZEUS is implemented as a Python library that can be imported into DNN training scripts. We have integrated ZEUS into PyTorch [171] by extending the `DataLoader` class, i.e., `ZeusDataLoader`. The class profiles power and throughput online by slicing epochs in iteration boundaries and invoking the NVML [30] library for power configuration and profiling. We have observed that five seconds of profiling for each power limit is enough to yield stable results. With the information, the optimal power limit can be automatically determined and applied. Moreover, `ZeusDataLoader` monitors the cost incurred by training and early stops the job if needed. Listing 4.1 shows an example training loop integrated with ZEUS.

Task	Dataset	Model	Optimizer	b_0	Target Metric
Speech Recognition	LibriSpeech [169]	DeepSpeech2 [91]	AdamW [150]	192	WER = 40.0%
Question Answering	SQuAD [180]	BERT (QA) [71]	AdamW [150]	32	F1 = 84.0
Sentiment Analysis	Sentiment140 [81]	BERT (SA) [71]	AdamW [150]	128	Acc. = 84%
Image Classification	ImageNet [70]	ResNet-50 [96]	Adadelta [219]	256	Acc. = 65%
Image Classification	CIFAR-100 [132]	ShuffleNet-v2 [153]	Adadelta [219]	1024	Acc. = 60%
Recommendation	MovieLens-1M [92]	NeuMF [97]	Adam [127]	1024	NDCG = 0.41

Table 4.1: Models and datasets used in our evaluation. The provided target metrics is the target for each training job. Here b_0 denotes the default batch size presented in the original work when feasible, otherwise we choose the maximum batch size which can consistently reach the target. The BERT(QA) and BERT(SA) means fine-tuning BERT on the tasks of question answering and sentiment analysis, respectively.

Observer Mode `ZeusDataLoader` supports an *Observer Mode*, where it profiles the power consumption and throughput of each power limit and determines the optimal one, but keeps the power limit at the maximum. By doing so, without affecting time or energy consumption, `ZeusDataLoader` reports how much time and energy the job *would have* consumed if the power limit were the optimal one, allowing the user to get an idea of the impact of using ZEUS. We believe that such a feature can encourage ZEUS's adoption by informing users of its potential savings.

4.6 Evaluation

We evaluate ZEUS's effectiveness in terms of navigating the energy-time tradeoff. Our key findings are as follows:

1. ZEUS reduces energy consumption by 15.3%–75.8%. It achieves this by trading off small performance for jobs that are already throughput-optimal; otherwise, it reduces training time by up to 60.1% too (§4.6.2).
2. ZEUS converges to the optimal configuration in a timely fashion (§4.6.2).
3. ZEUS can handle workloads with data drift well (§4.6.4) and overall incurs low overhead (§4.6.5).
4. ZEUS scales to multi-GPU settings (§4.6.6) and provides consistent savings across four generations of GPUs (§4.6.7).

Node	GPU Specification		Host Specification	
HPE Apollo	Model	A40 PCIe	CPU	AMD EPYC 7513
6500 Gen10 Plus	VRAM	48GB	RAM	512GB DDR4-3200
A40 × 4	mArch.	Ampere	Disk	960GB NVMe SSD
CloudLab [12]	Model	V100 PCIe	CPU	AMD EPYC 7542
r7525	VRAM	32GB	RAM	512GB DDR4-3200
V100 × 2	mArch.	Volta	Disk	2TB 7200rpm HDD
Chameleon	Model	RTX6000	CPU	Xeon Gold 6126
Cloud [124]	VRAM	24GB	RAM	192GB
RTX6000	mArch.	Turing	Disk	256GB SSD
Chameleon	Model	P100	CPU	Xeon E5-2670 v3
Cloud [124]	VRAM	16GB	RAM	128GB
P100 × 2	mArch.	Pascal	Disk	1TB HDD

Table 4.2: Hardware used in the evaluation.

4.6.1 Experimental Setup

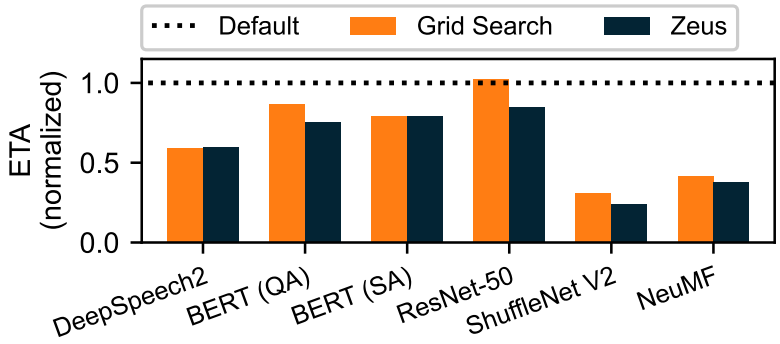
Testbed Setup We evaluate ZEUS with four generations of NVIDIA GPUs as specified in Table 4.2.

Workloads Table 4.1 summarizes our workloads. The default batch size (b_0) is chosen from the original model publication when available; otherwise, it is set to be the maximum batch size which consistently achieves the target accuracy. See Appendix B.5 for more details.

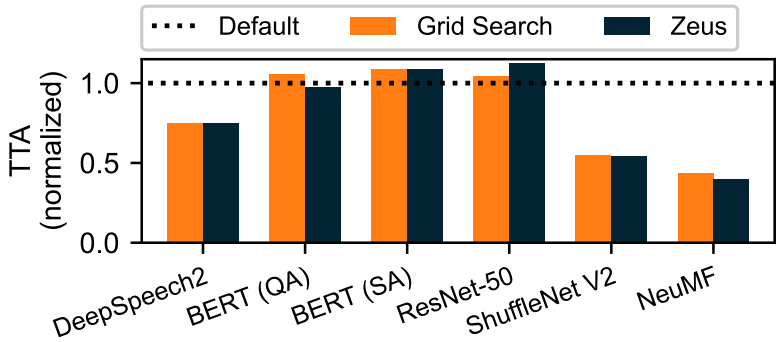
Baselines We compare against the following baselines:

1. *Default ($b = b_0$, $p =$ `MAXPOWER`).* This is often the default configuration used by practitioners, where the GPU power limit is set to, or rather *not changed from*, the maximum. This is the most conservative baseline with no exploration.
2. *Grid Search with Pruning.* This one tries out one configuration of (b, p) for each recurrence of the job and selects the best one. We optimize naïve grid search by having it prune out batch sizes that failed to reach the target metric.

Metric Our primary metrics are ETA (energy consumption) and TTA (training time). Ideally, we want to reduce both; but due to their tradeoff, sometimes it may not be possible to simultaneously do both.



(a) Energy Consumption



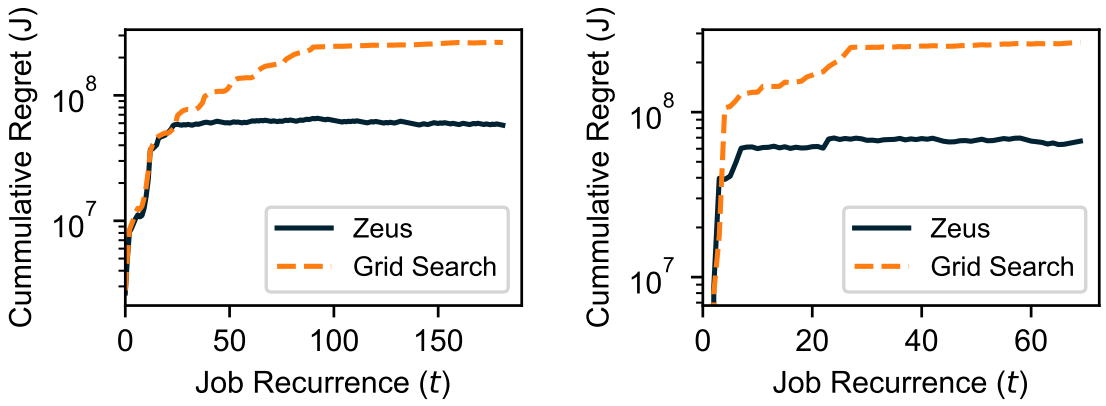
(b) Training Time

Figure 4.6: ZEUS reduces energy consumption for all workloads. (a) energy consumption, (b) training time of each workload, normalized by the Default baseline. Results are computed with the last ten recurrences, capturing the knobs each method converged to.

Defaults All experiments are done on NVIDIA V100 GPUs, unless otherwise mentioned. By default, we highlight $\eta = 0.5$ to strike a balance between ETA and TTA. Later, we sweep η from 0 to 1 (§4.6.7). The early-stopping threshold β is set to 2, and we also sweep β from 1.5 to 5 (§4.6.7). See Appendix B.6 for details on experimental methodology and parameter choices.

4.6.2 ZEUS Performance

In this section, we evaluate the performance of ZEUS in terms of energy consumption and training time as well as the convergence characteristics of our Multi-Armed Bandit algorithm. Each experiment is run across multiple recurrences of DNN training jobs. We select the recurrence number to be $2 \cdot |\mathcal{B}| \cdot |\mathcal{P}|$, so that the Grid Search baseline finishes exploration and also has plenty of chances to exploit its choice.



(a) DeepSpeech2 (b) ResNet-50

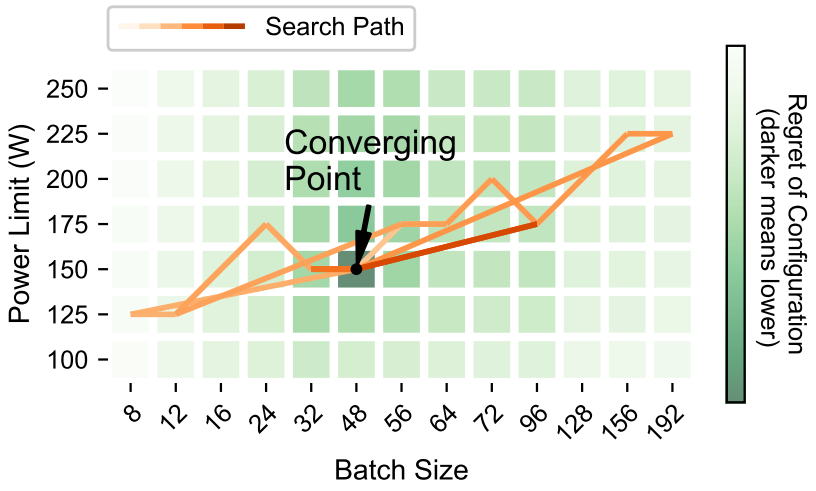
Figure 4.7: Cumulative regret of Zeus vs. Grid Search for (a) DeepSpeech2 and (b) ResNet-50.

Improvements in ETA Figure 4.6a shows the energy consumption (ETA) of the last ten recurrences of ZEUS and Grid Search w.r.t. the Default baseline, aiming to compare the final point each approach converged to. ZEUS reduces energy consumption (ETA) by up to 15.3%–75.8% w.r.t. the baseline. This is also comparable to the reduction we found by exhaustively searching through all the configurations in Section 4.2 as well as by using Grid Search.

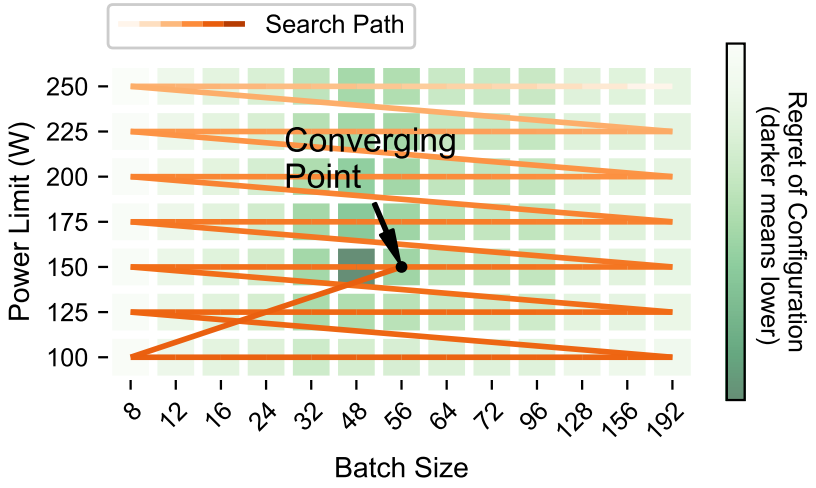
Tradeoff with TTA Figure 4.6b shows the time consumption (TTA) of the last ten recurrences of ZEUS and Grid Search w.r.t. the Default baseline. Even though ZEUS reduces training time by up to 60.1%, for some workloads TTA is increased by 12.8% (Figure 4.6b). This is due to the tradeoff between ETA and TTA, which is the central focus of this project. This is especially true for workloads with close to throughput-optimal b_0, where there is little room for TTA improvement.

Cumulative regret While ZEUS and Grid Search perform close to each other, ZEUS use significantly smaller amount of resources to converge. As a bandit-based solution, the effectiveness of our algorithm can be quantified via regret, the difference between the decision selected and the optimal choice (Equation 4.9 in Section 4.4.3).

Figure 4.7 shows the cumulative regret of ZEUS and Grid Search for DeepSpeech2 and ResNet-50. The optimal configuration is chosen by an exhaustive parameter sweep. We observe that in both workloads, ZEUS is able to achieve better regret from the first job recurrence. ZEUS reaches the plateau in the cumulative regret earlier than Grid Search, which means it converges to the optimal



(a) ZEUS



(b) Grid Search

Figure 4.8: Search paths of (a) ZEUS and (b) Grid Search for Deepspeech2. The heatmap in the background shows the regret of each (Batch Size, Power Limit) configuration. Darker background denotes lower regret and therefore better configuration. The colored line with shifting color shows the search path, with darker color being later recurrences.

solution earlier. We observe similar results for other workload as well (Appendix B.7). In the worst case, Grid Search uses 72× more energy than ZEUS in finding Pareto-optimal solutions.

Convergence to a Pareto-optimal configuration Despite having no information about the application beforehand, ZEUS learns the energy characteristics of it online in a few iterations. Figure 4.8 shows the search path of ZEUS and Grid Search during training DeepSpeech2. Due to the decoupling in optimization of power limit and batch size, ZEUS explores the configuration

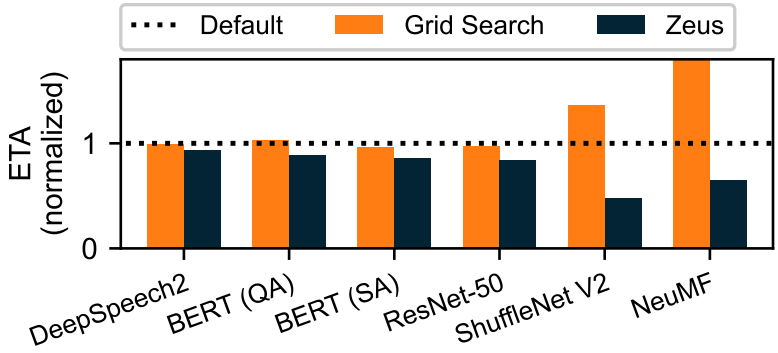
space more efficiently and converges to the optimal configuration much faster. We observe similar results for other workloads (see Appendix B.8). In contrast, in Figure 4.8b we observe that Grid Search may not even converge to optimal configuration. This is due to the stochastic nature of DNN training, with even the same batch size yielding different energy and time consumptions. Hence, Grid Search may choose a suboptimal configuration when a suboptimal configuration luckily yields good energy and time consumptions.

4.6.3 Trace-Driven Simulation Using the Alibaba Trace

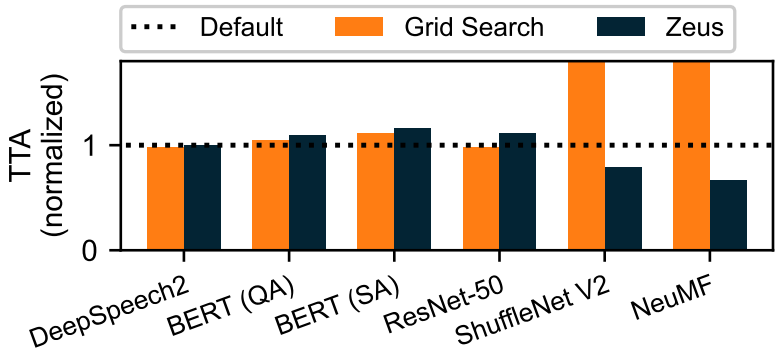
Here we evaluate how ZEUS can save energy and time consumption for DNN training in large clusters. We run trace-driven simulation using the Alibaba trace [212] which contains over 1.2 million jobs spanning a period of two months. The Alibaba GPU cluster trace to be suitable for our evaluation due to two reasons. First, the trace identifies recurring jobs, and each job is annotated with its group ID. Second, jobs inside the same group show overlap in their execution, allowing us to evaluate ZEUS's capability of handling concurrent job submissions with Thompson Sampling.

In order to assign job groups to the workload (Table 4.1) that best resembles its runtime, we remove jobs that did not successfully terminate and run K-Means clustering [94] on the mean job runtime of each group to form six clusters. Then, we match the six clusters with our six workloads in the order of their mean runtime. When running simulation, in order to capture the intra-cluster runtime variation of each job, we scale the job runtime with the ratio of the job's original runtime to its cluster's mean runtime. We compare ZEUS with Default and Grid Search and plot the results in Figure 4.9.

Figure 4.9a shows the cumulative energy consumption of training using all three approaches. ZEUS outperforms both baselines for workloads of all types and sizes. Note that there are scenarios where the Grid Search performs worse than Default, due to it wasting too much energy and time during the exploration stage. Thanks to ZEUS's *early stopping* and quick online power optimization, its energy and time cost during exploration stage of ZEUS is significantly reduced. Across all the models, ZEUS reduces training energy usage by 7%–52%. Figure 4.9b shows the training time using ZEUS to be increased by at most 16%, and in many cases even decreased by up to 33%. Finally, similar to earlier experiments, ZEUS had significantly lower cumulative regret than Grid Search.



(a) Energy Consumption



(b) Training Time

Figure 4.9: ZEUS reduces energy consumption for all jobs in the Alibaba cluster trace [212], compared to Grid Search and Default. (a) Energy consumption with ZEUS comparing against baselines, (b) Training time of each type of workload. Both are normalized by the Default baseline.

4.6.4 Handling Data Drift

While there are previous works that attempt to identify and address data drift in general ML settings [151], existing datasets are classification tasks based on small feature vectors [93, 56], completely synthetic [77, 106], or closed [154].

Therefore, we create and open-source a new sentiment analysis dataset called *Capriccio* that is suitable for evaluating DNN models. Capriccio consists of 1.6 million tweets over three months from the Sentiment140 [81] dataset, labeled with sentiment scores and the timestamp of the tweet. We emulate data drift by capturing a sliding window of 500,000 tweets (roughly the amount of tweets in one month) at a time and moving the window forward by each day, generating 38 slices. We skip empty dates to avoid having identical slices.

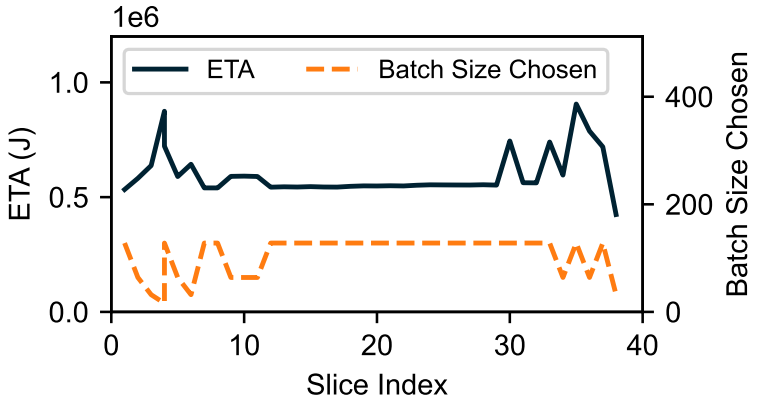


Figure 4.10: Energy consumption of training BERT with ZEUS on drifting dataset Capriccio and the batch size chosen for each slice.

We train BERT [71] on Capriccio with ZEUS configured with a window size of 10, roughly corresponding to a time frame of two weeks on Twitter. We plot the selected batch size for each recurrence (slice) and its corresponding ETA of training in Figure 4.10. We observe that the changes in ETA trigger the exploration of a batch size that is different from the converged optimal one, at slice 36.

4.6.5 Overhead of JIT Profiling

Measurements with the Deepspeech2 model using the default batch size b_0 show that JIT profiling results in a 0.01% increase in energy consumption and a 0.03% increase in time consumption. Such a tiny overhead is possible because the time needed to profile all power limits is very small (less than one minute) while one epoch of training spans hours (which is typical for DL workloads). Measurements on ShuffleNet-v2, which has much shorter epoch duration, show that JIT profiling results in a 0.6% increase in terms of time consumption and a 2.8% reduction in energy consumption.

4.6.6 Scaling to Multi-GPU

While the primary focus of this project is on single-GPU setting, in this section, we show that ZEUS can be extended to (single-node) multi-GPU training settings by profiling the power consumption of all GPUs that participate in training. Extensions to distributed multi-GPU setup

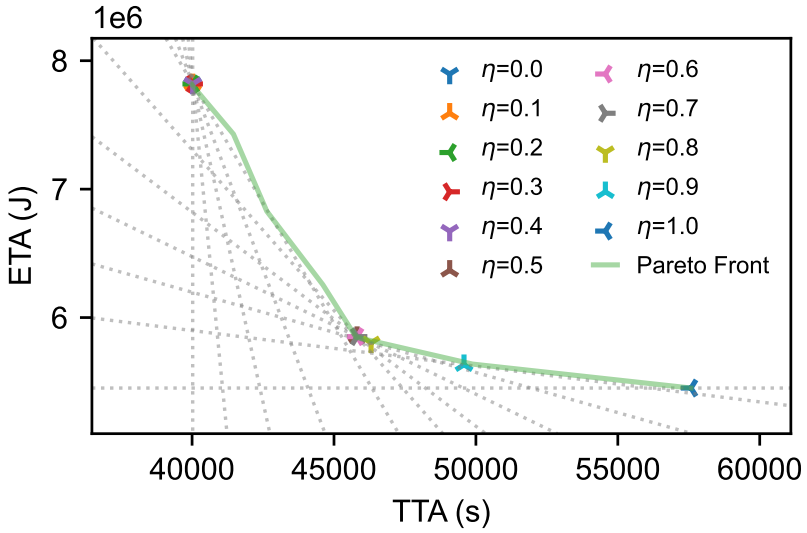


Figure 4.11: Pareto Front of DeepSpeech2 and how η nativates it.

that involves the network is a potential future work.

Extending to multi-GPU allows us to compare our energy and time consumption with Pollux [176], a state-of-the-art distributed cluster scheduler that dynamically tunes the batch size *during* DNN training in order to maximize *goodput*. Training LibriSpeech on four NVIDIA A40 GPUs with Deepspeech2, ZEUS consumes 12% more time but 21% less energy, comparing favorably. We especially note that while Pollux does not take energy into account, ZEUS allows the user to select a different time-energy tradeoff point (e.g., speed up training but consume more energy) by selecting an appropriate η.

4.6.7 Sensitivity Analysis and Ablation Studies

Impact of η To characterize the impact of η as defined in Equation 4.2, we perform a sweep of $0 \leq \eta \leq 1$ when training DeepSpeech2 and plot the resulting optimal (TTA, ETA) in Figure 4.11. We also plot the corresponding Pareto Front for reference. We observe that the resulting (TTA, ETA) data points falls closely to the Pareto Front. Moreover, we plot the lines along which the C in Equation 4.2 is a constant, shown as the dotted lines. As expected, these lines form an envelope around the Pareto Front. Additional sensitivity analysis for η can be found in Appendix B.9.

Impact of early-stopping threshold β To study impact of the early-stopping threshold β, we sweep β from 1.5 to 5 and measure the cumulative ETA across all jobs. We calculate the difference

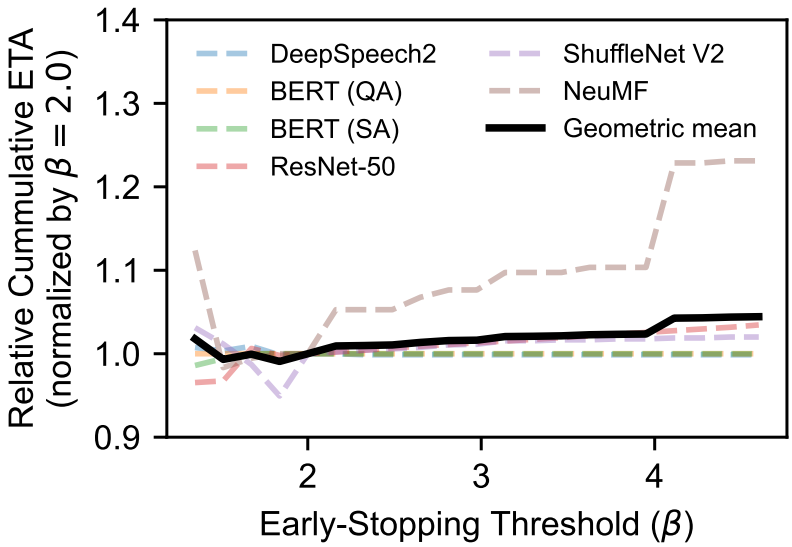


Figure 4.12: Relative cumulative energy consumption of ZEUS across all jobs, w.r.t. the early-stopping threshold β.

in ETA relative to our default choice of $\beta = 2.0$, and plot the result of all jobs as well as a geometric mean across all jobs in Figure 4.12. The result shows that the default $\beta = 2.0$ chosen by ZEUS achieves the lowest geometric mean across all jobs. The intuition behind this is: when the β is too low, ZEUS prematurely stops exploratory runs, reducing the effectiveness of exploration. In contrast, when β is too high, it dilutes the benefit of early stopping which leads to inflated energy consumption during exploration.

Impact of individual components In order to show the gains from each component, we show the degradation of removing one component from ZEUS: no early stopping (setting β to infinity), no pruning (keeping a batch size that didn't reach the target accuracy), and no JIT profiling (profiling each power limit in different recurrences). Figure 4.13 shows the slowdown relative to ZEUS after disabling these components. We observe that the performance of ZEUS benefits mostly from early stopping.

Impact of GPU models Figure 4.14 shows the geometric mean of ETA normalized against Default across all jobs. We observe a consistent ETA reduction of ZEUS across four generations of NVIDIA GPUs. More in Appendix B.10.

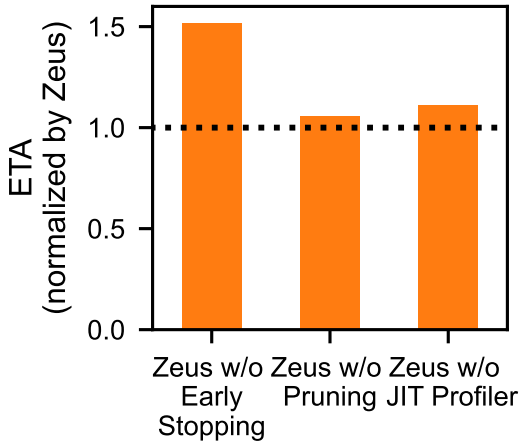


Figure 4.13: Performance breakdown of ZEUS.

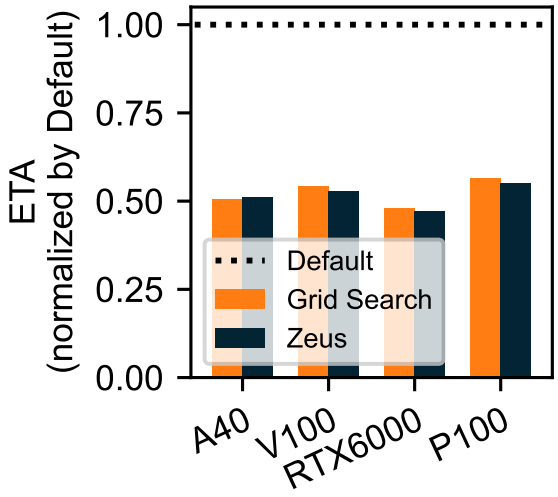


Figure 4.14: Normalized ETA w.r.t. GPU models.

4.7 Related Work

DNN Training A large body of recent studies focus on creating fast kernels for tensor operations [115, 209, 221, 63], efficiently placing data and/or computation [165, 179, 217, 141], and optimizing communication [208, 173]. However, most of them optimize for TTA and are oblivious of the energy impact of the techniques employed. These works can be applied together with ZEUS, potentially accelerating training while making it energy efficient.

Another recent effort in reducing TTA (without considering energy) in multi-GPU DNN training is Pollux [176]. Pollux dynamically changes the batch size *during* training based on the Gradient Noise Scale (GNS) [158]. However, GNS does not theoretically capture the generalization of the model [158] and can only be efficiently approximated when there are more than one GPUs participating in training. ZEUS, on the other hand, optimizes and trades-off TTA and ETA by tuning the batch size *across* job recurrences and does not alter the model's convergence characteristics.

Energy Measurement for Deep Learning A recent line of research has analysed the energy consumption [172] as well as the environmental impact [194, 137] for training large DNN models inside a cluster. On the device-level, benchmarking efforts have been made to understand the energy efficiency and performance of training DNN on GPUs and other accelerators [211]. Several Python frameworks have been built, for measurement [98, 58] and prediction [47] of energy consumption

for DNN training. ZEUS takes a similar software-based approach to measure power consumption via NVML [30], in order to perform JIT profiling of DNN training jobs.

Energy Optimization for Deep Learning Existing work has investigated energy-accuracy trade-off in the context of DNN inference with new neural network architecture [206] and algorithm-hardware co-design [196], and training strategies such as warm-start [50] and gradient-matching-based data subset selection [126]. Other works optimize energy for DNN training on multiple GPUs with scheduling [123] and task mapping [131]. ZEUS complements these solutions as it can be plug in transparently into these frameworks.

Several works have studied the impact of GPU DVFS and power configuration on energy consumption and performance of DNN training [222, 207, 197, 54, 131], wherein they focus on the tradeoff between the transient metric of system throughput and power consumption. They rely on offline modeling and profiling. In contrast, ZEUS focuses on the more realistic end-to-end metric of Energy-to-accuracy, and is fully online.

BatchSizer [164] introduces batch size as a control knob to optimize for energy efficiency of DNN inference. ZEUS focuses on DNN training, and takes a holistic approach, optimizing both GPU and job configurations together.

GPU Power Modeling Recent GPU architecture efforts have built program-level energy consumption models of GPU using machine learning [215], with consideration of DVFS [210]. A recent work AccelWattch [121] models GPU power consumption at cycle-level granularity. They require offline profiling to model and predict energy consumption. In contrast, ZEUS is an online adaptive mechanism.

4.8 Conclusion

In this work, we sought to understand and optimize the energy consumption of DNN training on GPUs. We identified the tradeoff between energy consumption and training time, and demonstrated that common practices can lead to inefficient energy usage. ZEUS is an online optimization framework for recurring DNN training jobs that *finds* the Pareto frontier and allows users to *navigate* the frontier by automatically tuning the batch size and GPU power limit of their jobs.

ZEUS outperforms the state-of-the-art in terms of energy usage for diverse workloads and real cluster traces by continuously adapting to dynamic workload changes such as data drift. We earnestly hope that ZEUS will inspire the community to consider energy as a first-class resource in DNN optimization.

CHAPTER 5

Conclusions

This dissertation focuses on bringing application-awareness into infrastructures and optimizing the end-to-end performance and efficiency of them. In this dissertation, we explore the opportunities and propose different means of co-optimizing applications and infrastructures for a wide variety of big data applications. We highlight two ways of implementing application-awareness: (a) white-box design, where the application explicitly exports its context to the infrastructure, and (b) black-box design, where the infrastructure infers application-level information instead. The former approach depends on an accurate model of the relationship between application-level objectives and infrastructure-level knobs as well as the availability of detailed application-level context, and it also requires changing the application. The latter approach infers application-level context from real-time signals such as performance counters, and performs empirical optimization without an analytical model. It does not change the application, making it more transparent for adoption and adaptive to new applications. We discuss how these advantages and disadvantages of the two approaches manifest in three popular big data use cases. The insights from this dissertation promote introducing application-awareness and can help future system designers choosing the approach of implementing it.

Application-Awareness for Data Analytics TERRA introduces the abstraction of DataBundle, which effectively captures application-level context. We forward this context into the infrastructure and update the scheduling algorithm to make informed decisions. By adopting the white-box application-awareness, TERRA improves average Job Completion Time by up to $3.43\times$ for the smallest and up to $32.04\times$ for the largest deployment.

Application-Awareness for Transaction Processing KAYAK shows that transaction processing systems can achieve higher throughput and lower latency with the help of black-box application-awareness. In this project, we show that by proactively and adaptively combining RPC *and* KV together, overall throughput and CPU utilization can be improved. We propose an algorithm that dynamically adjusts the rate of requests and the RPC fraction to improve overall request throughput while meeting latency SLO requirements.

Application-Awareness for Deep Learning Training ZEUS is an online optimization framework for recurring DNN training jobs that *finds* the Pareto frontier and allows users to *navigate* the frontier by automatically tuning the batch size and GPU power limit of their jobs. ZEUS outperforms the state-of-the-art in terms of energy usage for diverse workloads and real cluster traces by continuously adapting to dynamic workload changes such as data drift.

5.1 Limitations

TERRA TERRA assumes a cooperative scheduling policy where each job submitted has the same level of priority, and focuses on improving the collective metric of Job Completion Time (JCT) of all jobs. In reality, jobs may be submitted with different priorities and should be scheduled accordingly. In addition, TERRA assumes the knowledge of the topology of the underlying Wide-Area Network (WAN) as well as the capacity of the links.

KAYAK In KAYAK, we consider rate limiting and admission control because simply moving up the latency curve cannot push the servers beyond their physical capacity. This is because if a tenant specifies a strict SLO target, requests that miss this target are essentially failed requests. As such, we assume that the tenants will implement a mechanism such as resubmitting requests which are dropped due to rate limiting, after adapting to a slower rate. At a slower timescale, tenants should also increase provisioning to avoid slowing down, so that in the worst case scenario all requests can still be executed on all application servers within the SLO target. In practice, rate limiting has to be coordinated with the reprovision of the capacity at the cluster level.

ZEUS In ZEUS, we introduce a parameter η to balance the tradeoff between time consumption and energy cost for DL training, which simplifies the tradeoff. Even though the user can specify the η to prioritize either saving time or energy cost, picking the appropriate η is non-intuitive and may require some tuning. In addition, ZEUS assumes the DL training jobs to be recurrently submitted, and it performs better with more recurrences.

5.2 Future Work

Topology-Aware Query Planning for HMCA TERRA performs scheduling, path selection, and bandwidth allocation for inter-site shuffles of HMCA job, under a given query execution plan. The query planning and task placement (e.g. for instance finding out what is the best site to perform some aggregation of intermediate data) are out of the scope for this project. Existing work [201, 202] does not take the WAN topology into account when performing query optimizations, leaving a gap in this field. Future research of query optimization for HMCA applications should consider the heterogeneity of WAN topology.

Automatic Code Generation for Storage Functions Currently, developers using KAYAK have to code the same application logic again in the storage functions running inside the storage server. This duplication complicates the development process and increases the chance of human errors and bugs. Automatically generating server-side storage functions from code written using traditional KV API can alleviate this problem. This would be an interesting avenue for future work as it makes adoption and migration easier.

Cluster-wide Energy Optimization for Distributed Deep Learning ZEUS monitors GPU energy consumption using a software-based approach via NVIDIA Management Library [30]. Due to lack of reliable energy measurement tools for other components such as RAM, CPU, disk and NIC, the scope of ZEUS is limited to single-host deep learning (DL) training. Further research is needed to find ways to measure and monitor energy consumption for those components, in order to enable energy optimization for multi-host distributed DL training.

5.3 Final Remarks

As new applications and hardware emerge, the endeavor of designing appropriate software systems to orchestrate the two will never end. In this dissertation, we focus on bringing application-awareness to the underlying software systems and hardware, and highlight two approaches of implementing application-awareness. We discuss the advantages and disadvantages of each approach, within the context of three popular big data use cases. We believe that my preliminary work can be a reference point and stepping stone for future research on building more efficient big data systems.

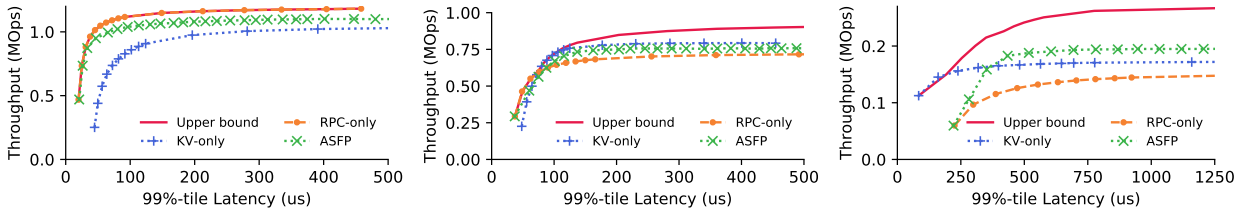
APPENDIX A

Kayak Appendices

A.1 Supplemental Measurements

A.1.1 Supplemental Measurements for Graph Traversal Workload

We repeat the same parameter sweep as in §3.2.2 but with the number of storage accesses per request set to 4 and 8. We vary the RPC fraction from 0 to 1 and measure the overall throughput and end-to-end latency of all requests. In doing so, we obtain the throughput-latency measurements for all possible execution configurations, as shown in Figure A.1 and Figure A.3. The optimal RPC fraction during these sweeps are shown in Figure A.2 and Figure A.4. We note that the observation we make in §3.2.2 still holds in these measurements.



(a) Light (100ns/RTT) (b) Medium (1μs/RTT) (c) Heavy (10μs/RTT)

Figure A.1: Throughput w.r.t. SLO (99%-tile latency) for graph traversal with four storage accesses per request.

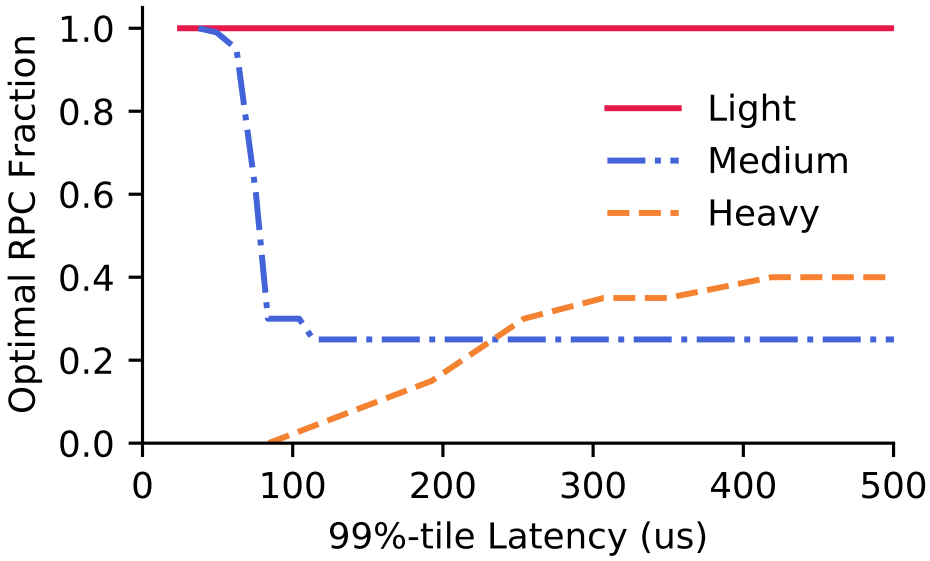
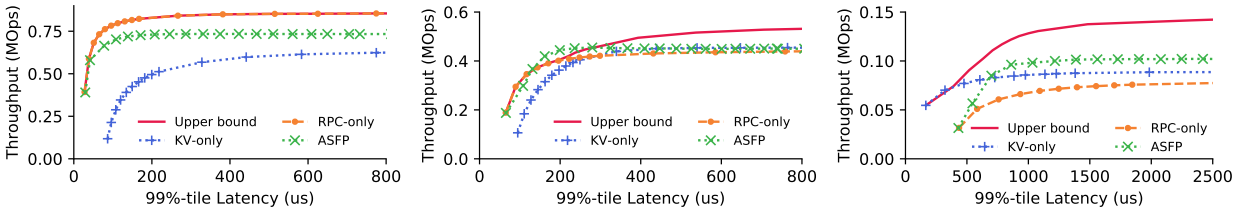


Figure A.2: Optimal RPC fraction w.r.t. SLO (99%-tile latency) for graph traversal with four storage accesses per request.



(a) Light (100ns/RTT) (b) Medium (1μs/RTT) (c) Heavy (10μs/RTT)

Figure A.3: Throughput w.r.t. SLO (99%-tile latency) for graph traversal with eight storage accesses per request.

A.2 Analysis of Algorithms

A.2.1 Problem Formulation

We aim to solve the following optimization problem

$$\begin{aligned} \max_{X} \quad & R \\ s.t. \quad & T(X, R) \leq t_0 \end{aligned} \tag{P}$$

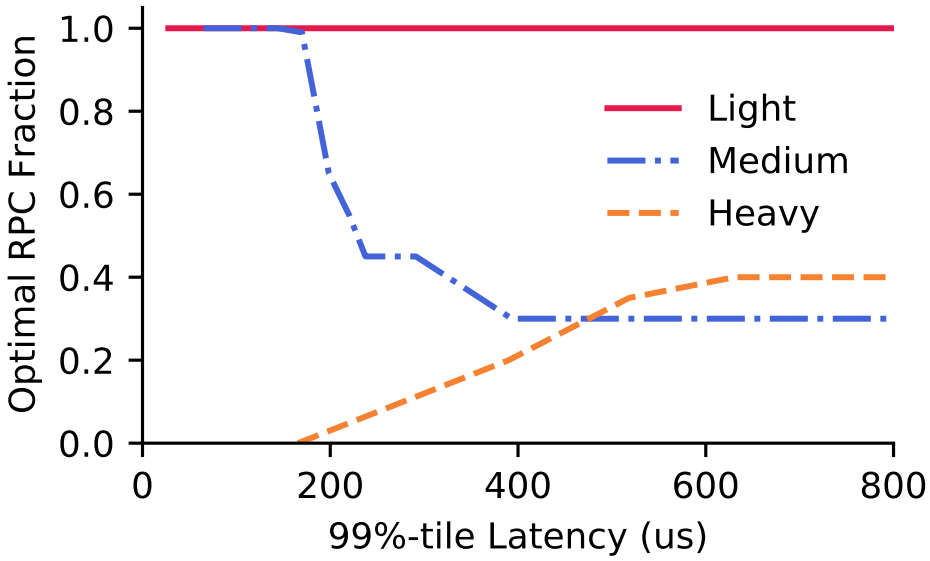


Figure A.4: Optimal RPC fraction w.r.t. SLO (99%-tile latency) for for graph traversal with eight storage accesses per request.

where we assume:

Assumption 1. *Fix X, $F_X(R) := T(X, R)$ is monotonic increasing and twice differentiable.*

Assumption 2. *For any X, there exist R_1 and R_2 such that $\min_X T(X, R_1) \leq t_0 \leq \min_X T(X, R_2)$.*

A.2.2 Algorithm I: X-R Dual Loop Control

We adopt the following iterative algorithm to solve the problem:

Algorithm I (X-R Dual Loop Control) For $k = 1, \ldots, K$, we alternatively update X_k and R_k by

- $X_k = \arg\min_X T(X, R_k)$;
- $R_{k+1} = R_k + \eta\,(t_0 - T(X_k, R_k))$.

We have the following theorem to characterize the convergence of the above algorithm.

Theorem 1. *Suppose in addition we have,*

1. *$T(X, R)$ is twice differentiable and lower bounded.*

2. *Fix R, $F_R(X) := T(X, R)$ is μ-strongly convex*[1]*, L-smooth*[2] *and coercive.*[3]

3. *For all X, we have $0 < \alpha \leq \frac{\partial T(X,R)}{\partial R} \leq \beta$.*

If we set $0 < \eta < \frac{1}{\beta}$, then

$$|R_K - R_*| \leq (1 - \eta\alpha)^K \cdot |R_0 - R_*| .$$

In the following we elaborate the proof for Theorem 1. We begin with introducing a series of lemmas. Let us denote

$$H(R) = \min_X T(X, R).$$

Lemma 1. *For all $R \neq S$,*

$$0 < \alpha \leq \frac{H(R) - H(S)}{R - S} \leq \beta.$$

Moreover, the above inequality implies $H(R)$ is monotonic increasing and continuous.

Proof. Without loss of generality let $R > S$. Let $X = \arg\min_X T(X, R)$ and $Y = \arg\min_X T(X, S)$. Then

$$\frac{H(R) - H(S)}{R - S} = \frac{T(X, R) - T(Y, S)}{R - S}$$
$$\begin{cases} \leq \frac{T(Y,R)-T(Y,S)}{R-S} = \frac{\partial T(Y,P)}{\partial R} \leq \beta, \\ \geq \frac{T(X,R)-T(X,S)}{R-S} = \frac{\partial T(X,Q)}{\partial R} \geq \alpha > 0. \end{cases}$$

Here $P, Q \in (S, R)$ are given by mean-value theorem.

□

Lemma 2. *There exists an unique R_* such that $H(R_*) = t_0$. Moreover, this R_* gives the maximum of the original optimization problem.*

Proof. We have already shown that $H(R)$ is monotonic and continuous. Recall that there exists R_1 and R_2 such that $H(R_1) \leq t_0 \leq H(R_2)$, thus there exists an unique R_* such that $H(R_*) = t_0$.

[1] $f(x)$ is μ-strongly convex, if for all x and y, it holds that $f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \frac{\mu}{2} \|y - x\|_2^2$.

[2] $f(x)$ is L-smooth, if for all x and y, it holds that $f(x) \leq f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2} \|y - x\|_2^2$. In general, if $f(x)$ is twice-differentiable and x is restricted in a bounded domain, then $f(x)$ is L-smooth in that domain, for some finite L.

[3] $f(x)$ is coercive if $f(x) \to +\infty$ as $\|x\| \to +\infty$. A strongly convex and coercive function admits an unique and finite minimum point.

For any R so that $R > R_*$, we have $H(R) > H(R_*) = t_0$ by monotonicity, thus R does not meet the constraint. Therefore R_* is the maximum of the optimization problem.

□

Proof of Theorem 1. With $H(R) := \min_X T(X, R)$, we can rephrase Algorithm I as

$$R_{k+1} = R_k + \eta \left(H(R_*) - H(R_k) \right).$$

Let R_0 be the initialization. We next show the convergence of this iteration.

Case I: $R_0 < R_*$**.** If $R_k < R_*$, then

$$\begin{aligned} R_* - R_{k+1} &= R_* - R_k - \eta \left(H(R_*) - H(R_k) \right) \\ &\begin{cases} \leq R_* - R_k - \eta\alpha \left(R_* - R_k \right) = (1 - \eta\alpha) \left(R_* - R_k \right) \\ \geq R_* - R_k - \eta\beta \left(R_* - R_k \right) = (1 - \eta\beta) \left(R_* - R_k \right) \end{cases} \end{aligned}$$

that is $0 \leq (1 - \eta\beta) \left(R_* - R_k \right) \leq R_* - R_{k+1} \leq (1 - \eta\alpha) \left(R_* - R_k \right)$. Using this recursion, if $R_0 < R_*$, we have

$$0 \leq (1 - \eta\beta)^K \left(R_* - R_0 \right) \leq R_* - R_K \leq (1 - \eta\alpha)^K \left(R_* - R_0 \right).$$

Case II: $R_0 > R_*$**.** If $R_k > R_*$, then

$$\begin{aligned} & R_{k+1} - R_* \\ &= R_k + \eta \left(H(R_*) - H(R_k) \right) - R_* \\ &= R_k - R_* - \eta \left(H(R_k) - H(R_*) \right) \\ &\begin{cases} \leq R_k - R_* - \eta\alpha \left(R_k - R_* \right) = (1 - \eta\alpha) \left(R_k - R_* \right) \\ \geq R_k - R_* - \eta\beta \left(R_k - R_* \right) = (1 - \eta\beta) \left(R_k - R_* \right) \end{cases} \end{aligned}$$

that is $0 \leq (1-\eta\beta)(R_k - R_*) \leq R_{k+1} - R_* \leq (1-\eta\alpha)(R_k - R_*)$. Using this recursion, if $R_0 > R_*$, we have

$$0 \leq (1-\eta\beta)^K (R_0 - R_*) \leq R_K - R_* \leq (1-\eta\alpha)^K (R_0 - R_*).$$

To sum up, when $\eta < \frac{1}{\beta}$ (hence smaller than $\frac{1}{\alpha}$), we have

$$|R_K - R_*| \leq \mathcal{O}\left((1-\eta\alpha)^K\right).$$

□

A.2.3 Algorithm II: R-X Dual Loop Control

Let us take a closer look at the optimization problem (P) under Assumption 1 and Assumption 2. First we observe the maximal must be attended at the boundary

$$T(X, R) = t_0. \tag{A.1}$$

Second the boundary constraint Eq. (A.1) implicitly defines a function $R(X)$, where

$$T(X, R(X)) = t_0.$$

We highlight that $R(X)$ is indeed well defined, since under Assumption 1 and Assumption 2, for any X, there exists an unique $R(X)$ that satisfies the boundary constraint.

With the above observations, we may rephrase the optimization problem (P) as

$$\max_X \quad R(X) \tag{P'}$$

where $R(X)$ is implicitly defined by the boundary constraint. In the following we discussion algorithms that solve problem (P').

Our challenge it that we do not have direct access to $R(X)$; instead at each fast loop step, we

have an estimation to $R(X)$, denoted as $R_k(x)$, which approximately satisfies

$$T(X, R_k(X)) \approx t_0.$$

In this set up we can perform *stochastic gradient ascent* (SGA, or *online gradient ascent*) for $R_k(X)$. We summarize the algorithm in the following.

Algorithm II (R-X Dual Loop Control) For $k = 1, \ldots, K$, we respectively update X_k and R_k by

1. Apply rate control so that the latency approximates SLO, i.e.,
   R_k *be such that* $T(X_k, R_k) \approx t_0$*;*
2. Use gradient ascent to search for the optimal X_k, i.e.,
   $X_{k+1} = X_k + \eta \frac{\mathrm{d}R_k}{\mathrm{d}X}$*, where* $T(X, R_k) = t_0$*, and* η *is a positive stepsize.*

There is a rich literature for the theory of online learning when $R_k(X)$ is concave, e.g., see [168]. For completeness, we introduce the following theorem to characterize the behavior of the above algorithm.

Theorem 2. *Suppose* $R_k(X)$ *is concave. Consider the iterates of SGA, i.e.,*

$$X_{k+1} = X_k + \eta \nabla R_k(X_k).$$

Then we have the following bound for the regret

$$\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) \leq \frac{\|X_1 - X_*\|_2^2}{2\eta} + \frac{\eta}{2} \sum_{k=1}^{K} \|\nabla R_k(X_k)\|_2^2 .$$

If in addition we assume $\|\nabla R_k(X)\|_2 \leq L$*, and set*

$$\eta = \frac{\|X_1 - X_*\|_2}{L\sqrt{K}},$$

then

$$\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) \leq C \cdot \sqrt{K}, \tag{A.2}$$

where $C := L \|X_1 - X_*\|_2$ is a constant depends on initialization and gradient bound.

Remark. The sublinear regret bound implies SAG behaviors nearly optimal on average: we see this by setting $X_* = \arg\max_X \sum_{k=1}^K R_k(X)$, and noticing that

$$\frac{1}{K}\sum_{k=1}^K R_k(X_*) - \frac{1}{K}\sum_{k=1}^K R_k(X_k) \leq \mathcal{O}\left(\frac{1}{\sqrt{K}}\right) \to 0.$$

More concisely, in our algorithm, $\{R_k(X)\}_{k=1}^K$ corresponds to a sequence of inaccurate estimations to the true implicit function $R(X)$ — even so the theorem guarantees a sublinear regret bound, which implies that our algorithm behaviors nearly as good as one can ever expect under the estimations, no matter how inaccurate they could be.

Furthermore, if for each k, $R_k(X)$ is an *unbiased estimator* to the true concave function $R(X)$, i.e., $\mathbb{E}R_k(X) = R(X)$, then $\bar{X} = \frac{1}{K}\sum_{k=1}^K X_k$ converges to the maximal of $R(X)$ in expectation: we see this by choosing $X_* = \arg\min_X R(X)$ and noticing that

$$\begin{aligned}
\mathbb{E}\left[R(X_*) - R(\bar{X})\right] &\leq \frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[R(X_*) - R(X_k)\right] \\
&= \frac{1}{K}\sum_{k=1}^K \mathbb{E}\left[R_k(X_*) - R_k(X_k)\right] \\
&\leq C \cdot \frac{1}{\sqrt{K}} \to 0.
\end{aligned}$$

Proof of Theorem 2. We first notice the following ascent lemma

$$\begin{aligned}
&\|X_{k+1} - X_*\|_2^2 \\
&= \|X_k + \eta \nabla R_k(X_k) - X_*\|_2^2 \\
&= \|X_k - X_*\|_2^2 + \eta^2 \|\nabla R_k(X_k)\|_2^2 + 2\eta \langle \nabla R_k(X_k), X_k - X_* \rangle \\
&\leq \|X_k - X_*\|_2^2 + \eta^2 \|\nabla R_k(X_k)\|_2^2 + 2\eta \left(R_k(X_k) - R_k(X_*)\right),
\end{aligned}$$

where the last inequality is due to the assumption that $R_k(X)$ is concave. Next we re-arrange the

terms and take telescope summation,

$$\begin{aligned}
&\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) \\
&\leq \sum_{k=1}^{K} \frac{1}{2\eta} \left(\|X_k - X_*\|_2^2 - \|X_{k+1} - X_*\|_2^2\right) + \sum_{k=1}^{K} \frac{\eta}{2} \|\nabla R_k(X_k)\|_2^2 \\
&= \frac{1}{2\eta} \left(\|X_1 - X_*\|_2^2 - \|X_{K+1} - X_*\|_2^2\right) + \sum_{t=1}^{T} \frac{\eta}{2} \|\nabla R_k(X_k)\|_2^2 \\
&\leq \frac{1}{2\eta} \|X_1 - X_*\|_2^2 + \sum_{k=1}^{K} \frac{\eta}{2} \|\nabla R_k(X_k)\|_2^2 ,
\end{aligned}$$

which gives the first regret bound.

If further we have $\|\nabla R_k(X)\|_2 \leq L$, then

$$\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) \leq \frac{1}{2\eta} \|X_1 - X_*\|_2^2 + \frac{\eta}{2} L^2 K,$$

by setting $\eta = \frac{\|X_1 - X_*\|_2}{L\sqrt{K}}$ we obtain

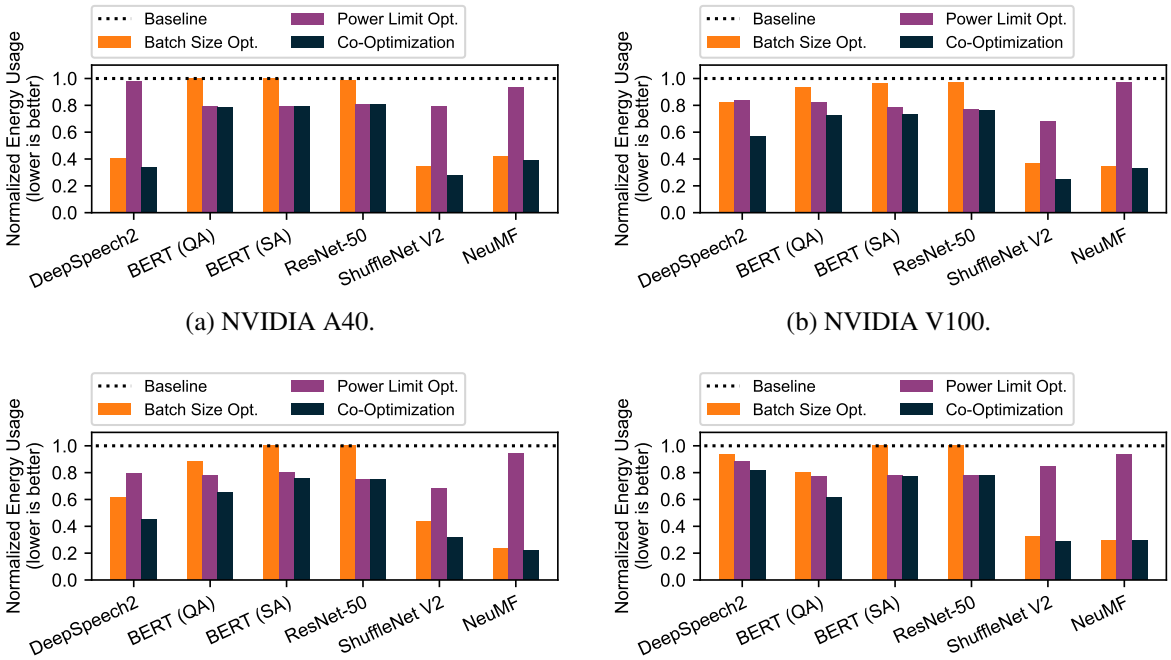
$$\begin{aligned}
\sum_{k=1}^{K} \left(R_k(X_*) - R_k(X_k)\right) &\leq \frac{1}{2\eta} \|X_1 - X_*\|_2^2 + \frac{\eta}{2} L^2 K \\
&\leq L \|X_1 - X_*\|_2 \cdot \sqrt{K}.
\end{aligned}$$

□

APPENDIX B

Zeus Appendices

B.1 Energy Savings Potential on GPUs



(a) NVIDIA A40.

(b) NVIDIA V100.

(c) NVIDIA RTX6000.

(d) NVIDIA P100.

Figure B.1: Energy usage normalized against Baseline for DNN training, measured on (a) NVIDIA A40 GPU, (b) NVIDIA V100 GPU, (c) NVIDIA RTX6000 GPU and (d) NVIDIA P100 GPU.

Figure B.1 shows the potential for energy savings on four different generations of NVIDIA GPUs: Ampere (A40), Volta (V100), Turing (RTX6000), and Pascal (P100). All four generations show that there are sufficient potential for energy savings, motivating ZEUS.

B.2 TTA vs. ETA for All Workloads

Figure B.2 plots the Pareto Front for all six workloads and the baseline (default batch size and maximum power limit) is shown as a red triangle. Note that the axes do not start from zero in order to zoom into the Pareto Front. Data points were gathered on an NVIDIA V100 GPU.

B.3 Job Recurrence in the Alibaba Cluster Trace

We perform independent analysis on the production GPU cluster trace provided by [212]. Figure B.3 shows the fraction of jobs that recur at least a certain number of times. For instance, 75% of the jobs are recurring 35 times or more. Based on this analysis and [212], we argue that DNN jobs recur in a production environment.

B.4 ETA w.r.t. Configurations for All Workloads

Figures B.4 and B.5 respectively show the ETA value when batch size and power limit are sweeped. Especially note the convexity of all BS-ETA curves, which justifies the design of our pruning exploration algorithm.

B.5 Choice of the Default Batch Size and Learning Rate

Our choice of the default batch size b_0 is to best-effort simulate the common practice of DNN training. Thus, whenever possible, we choose the default batch size from the original model publication assuming that the authors reasonably tuned it. If such method is not possible, we choose the maximum batch size which consistently achieves the target accuracy (we have this information available from the training trace collected in Appendix B.6).

In terms of learning rate, models trained with the Adadelta [219] optimizer do not require an initial learning rate. For optimizers that do require an initial learning rate, we made our best effort in choosing a batch size and learnig rate pair that achieves reasonable accuracies by experimenting with values from the original publication of the model and those discovered by popular DL frameworks

(e.g., Huggingface [213]).

After collecting the inital batch size and learning rate pairs, when we scale the batch size, we applied Square Root Scaling [100] for adaptive optimizers such as Adam [127] following recent theoretical results [85].

B.6 Experiment Methodology

Due to resource constraints and environmental concerns, we cannot afford to repeatedly train all of our workloads with various configurations end-to-end hundreds of times sequentially. However, similar to how ZEUS *decouples* the exploration of batch size and power limit, we may apply the same decoupling in our experimentation. That is, we instead take a trace-driven approach, where we collect two kinds of trace data:

1. Training trace. We train all possible combinations of models and batch sizes until convergence and record the number of epochs the model took to reach its target accuracy. We repeat this with four different random seeds for every combination to capture the stochasticity in DNN training.
2. Power trace. We use our JIT profiler to collect the throughput and average power consumption of all possible combinations of model, batch size, and power limit.

We then replay these traces when we need to train a model and reconstruct its TTA and ETA values in order to evaluate the decisions made by ZEUS and baselines. Moreover, since we have access to all the possible choices and their outcome, we also know the optimal choice. Therefore, with the traces, we can evaluate the regret achieved by ZEUS and baselines.

Note that ZEUS does not directly learn from these traces (which would be offline-profiling), but instead only learns from the *replay* of these traces in an online fashion. We will make all of our traces and artifacts available.

While the aforementioned trace-driven method is used widely throughout our evaluation, for the case of Cappricio, it is more expensive to construct the trace than to run ZEUS end-to-end. This is because Cappricio is essentially a set of 38 different datasets. Thus, we implement and run ZEUS end-to-end on Cappricio and report its results in Section 4.6.4.

B.7 Cumulative Regret of All Workloads

Figure B.6 shows the cumulative regret of ZEUS and Grid Search over job recurrences for all six workloads. In general, ZEUS converges to a better knob than Grid Search while being faster.

B.8 Search Paths for All Workloads

Figures B.7 and B.8 respectively show the search path of Zeus and Grid Search in the 2D configuration space. Thanks to the decoupling of batch size and power limit, ZEUS is able to more efficiently navigate the search space and converge to a knob while consuming less energy and time during exploration.

B.9 Additional Sensitivity Analysis

Figure B.9 compares both the energy consumption and training time for ZEUS against Default. We also calculate and plot the geometric mean across all jobs. The result shows that with higher η, ZEUS prioritizes reducing energy consumption over time, leading to higher improvement factor of energy, and vice versa.

B.10 Performance of ZEUS on All GPUs

Figure B.10 presents the energy and time consumption of all workloads on four different generations NVIDIA GPUs: Ampere (A40), Volta (V100), Turing (RTX6000), and Pascal (P100). The overall trends hold for all GPUs.

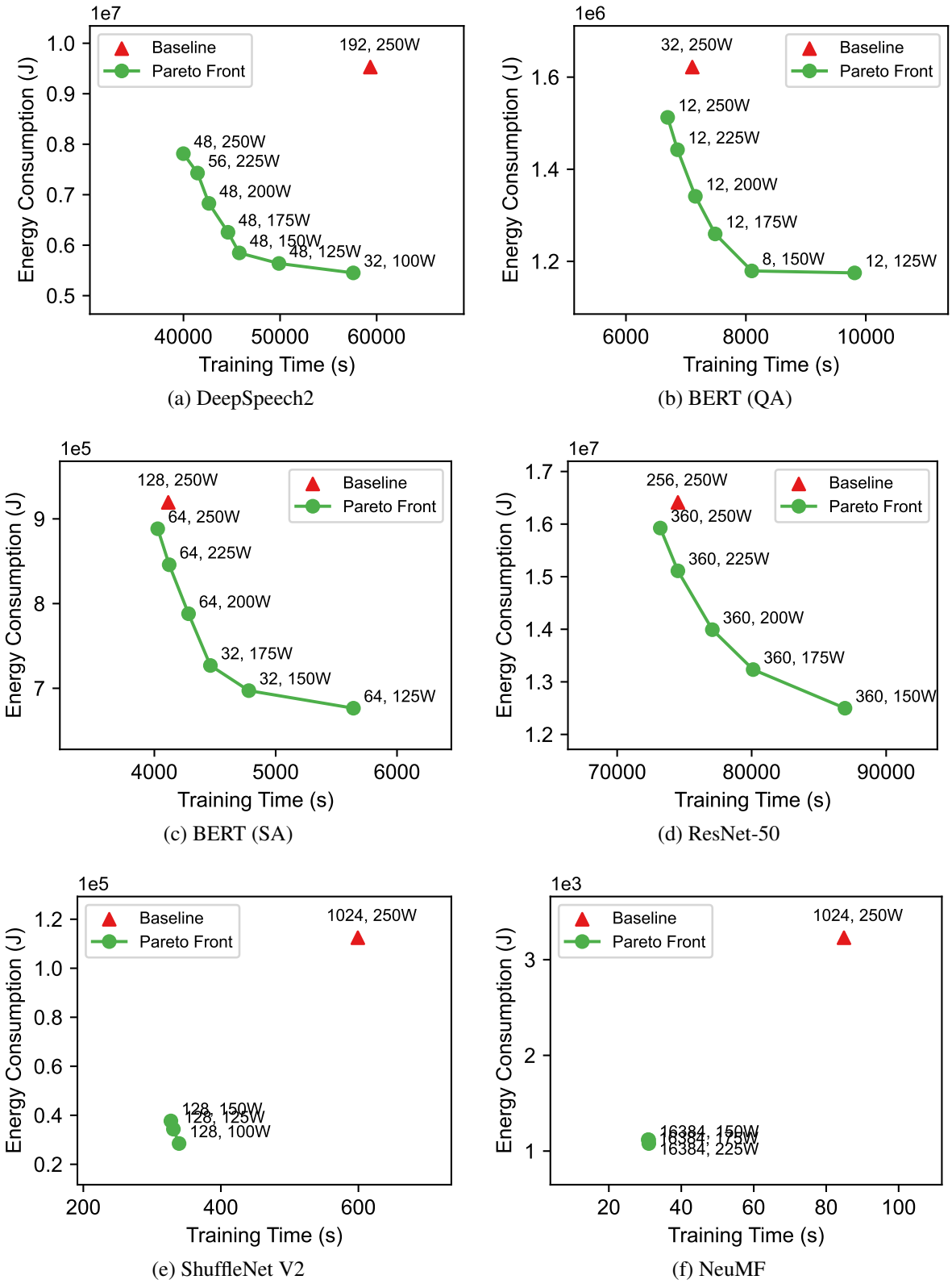


Figure B.2: ETA vs. TTA across all workloads, with Pareto Front and default configuration highlighted. Measured on an NVIDIA V100 GPU.

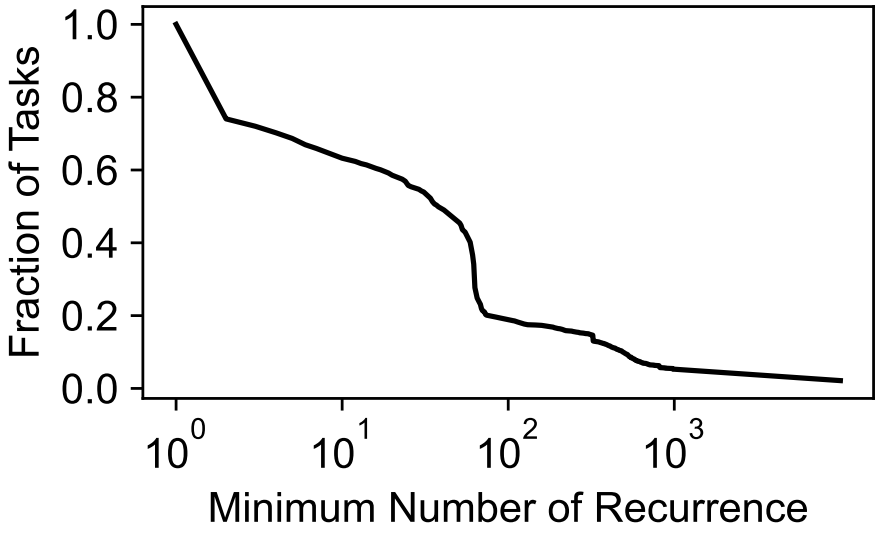


Figure B.3: CDF of job recurrence in the Alibaba MLaaS cluster trace [212].

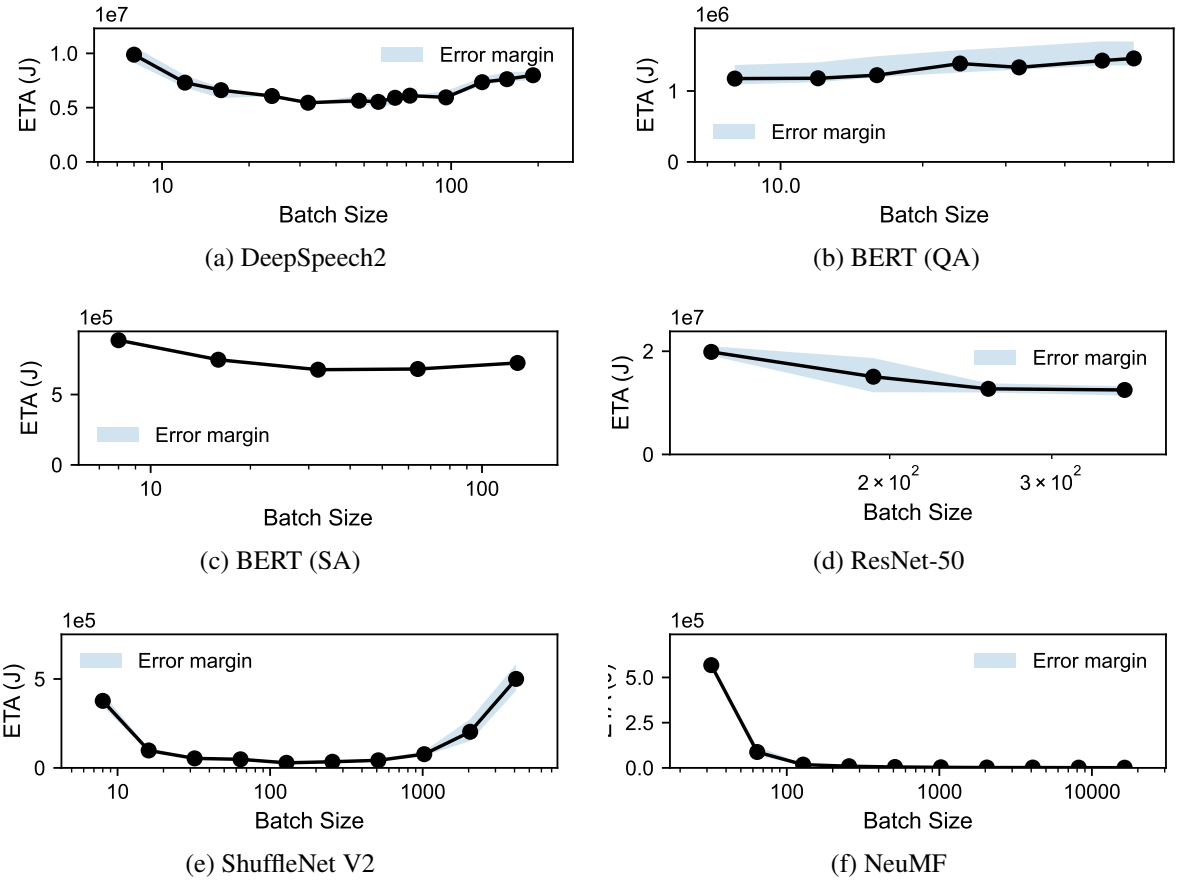
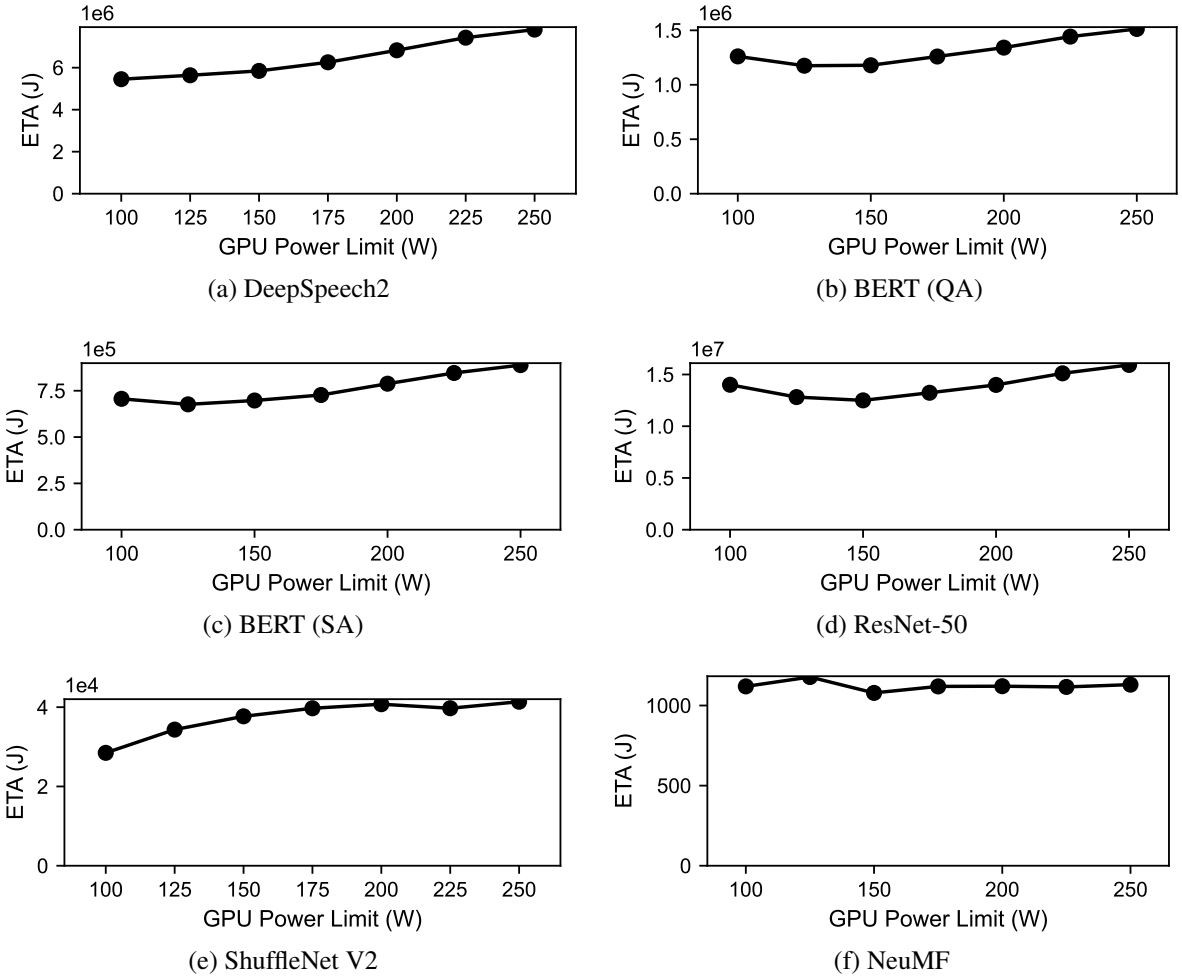


Figure B.4: ETA w.r.t batch size of different DNN training workload. The blue shade shows the error margin across repeated runs.



(a) DeepSpeech2

(b) BERT (QA)

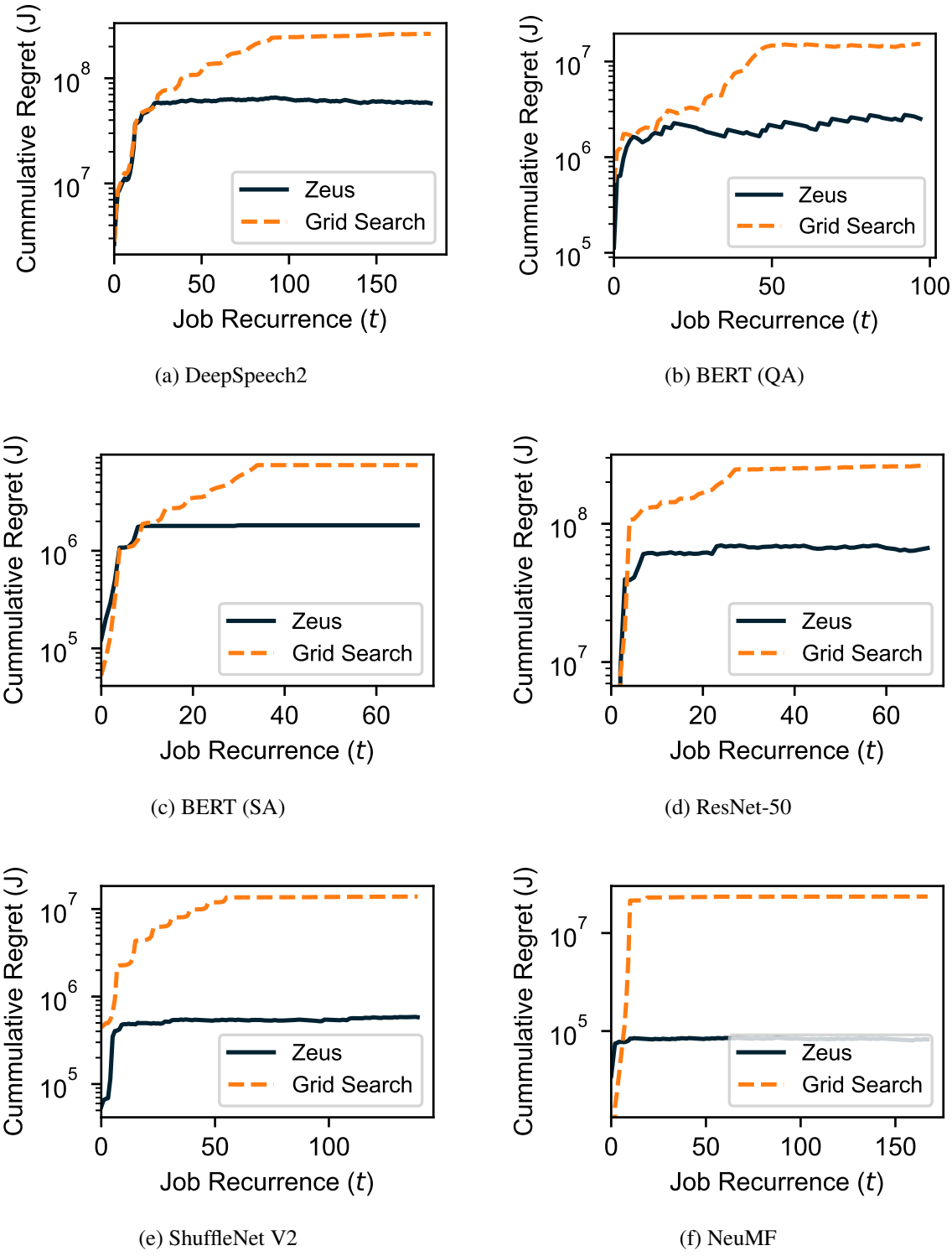
(c) BERT (SA)

(d) ResNet-50

(e) ShuffleNet V2

(f) NeuMF

Figure B.5: ETA w.r.t GPU power limit of different DNN training workload. Measured on an NVIDIA V100 GPU.



(a) DeepSpeech2

(b) BERT (QA)

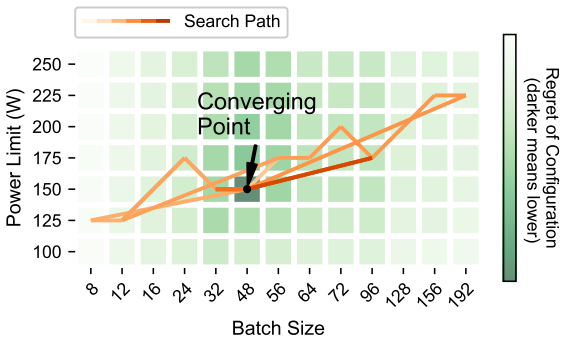
(c) BERT (SA)

(d) ResNet-50

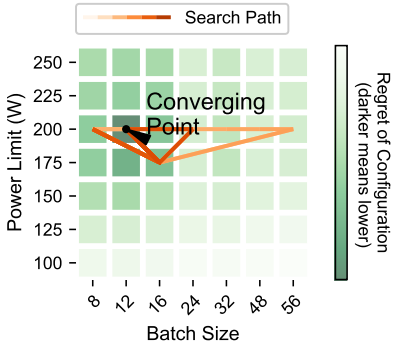
(e) ShuffleNet V2

(f) NeuMF

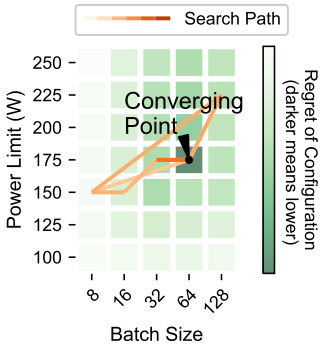
Figure B.6: Cumulative regret of Zeus vs. Grid Search across all workloads.



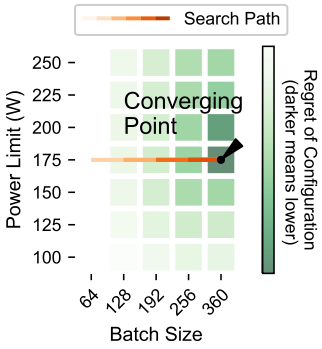
(a) DeepSpeech2



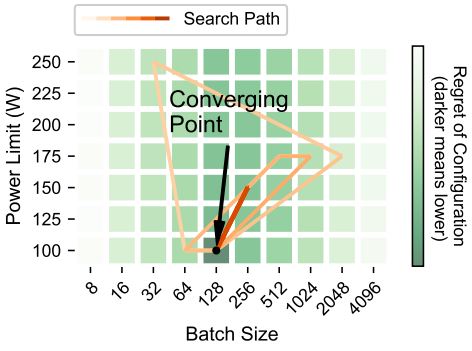
(b) BERT (QA)



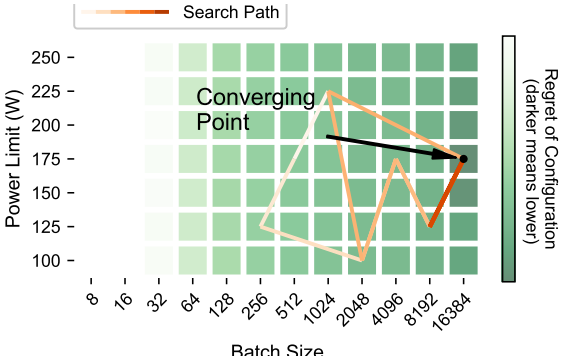
(c) BERT (SA)



(d) ResNet-50

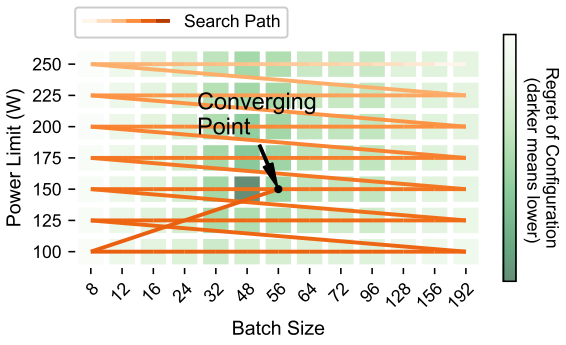


(e) ShuffleNet V2

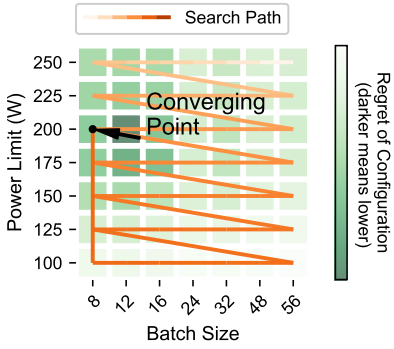


(f) NeuMF

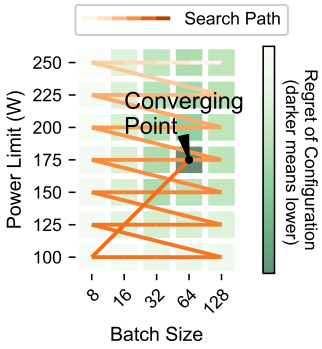
Figure B.7: Search path of Zeus across all workloads.



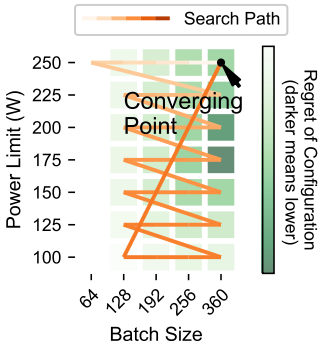
(a) DeepSpeech2



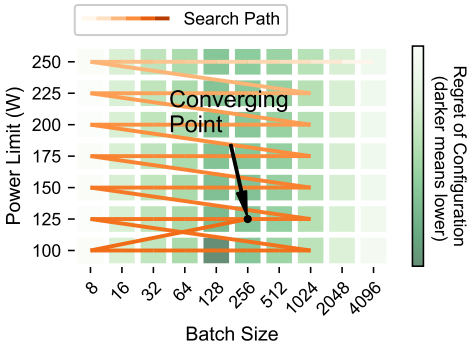
(b) BERT (QA)



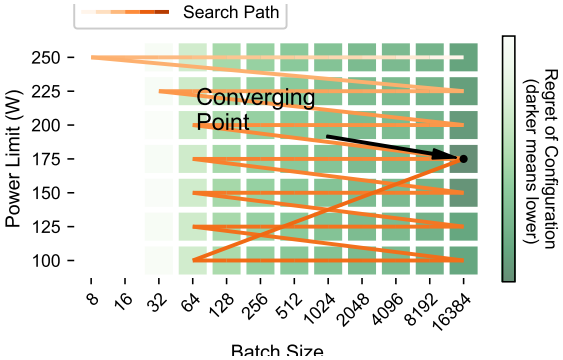
(c) BERT (SA)



(d) ResNet-50

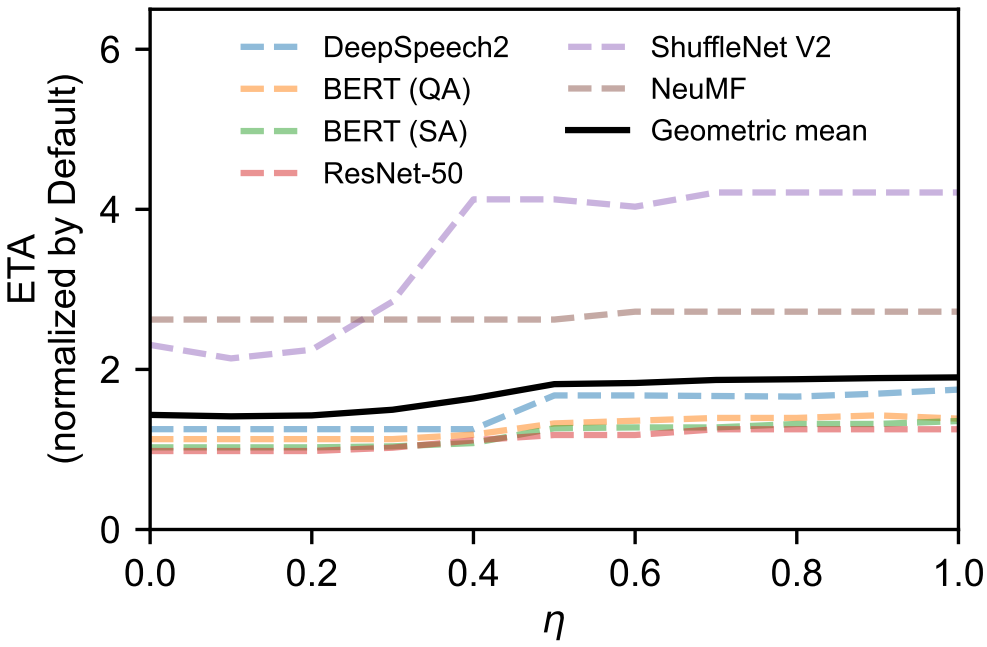


(e) ShuffleNet V2

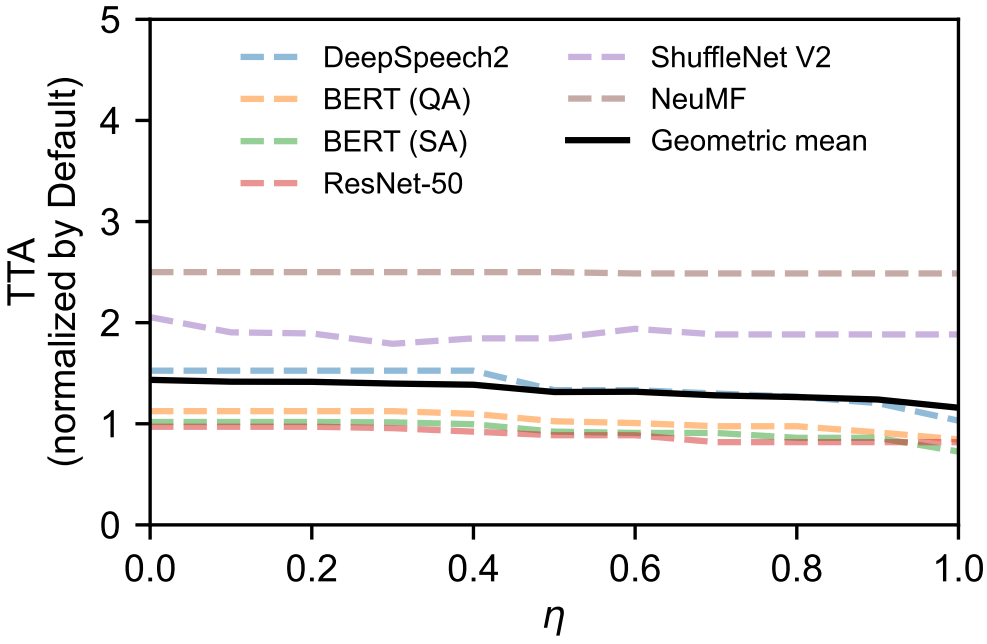


(f) NeuMF

Figure B.8: Search path of Grid Search across all workloads.

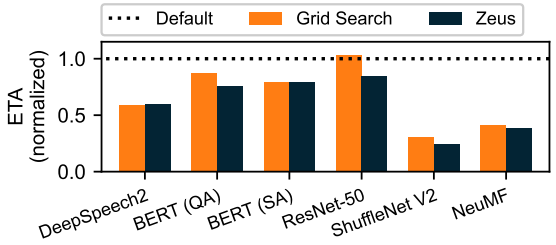


(a) ETA

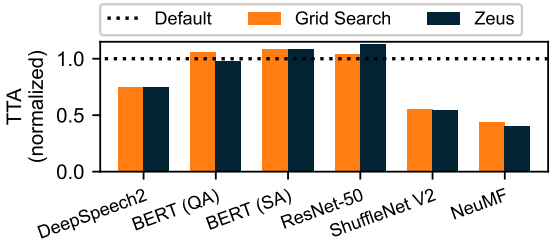


(b) TTA

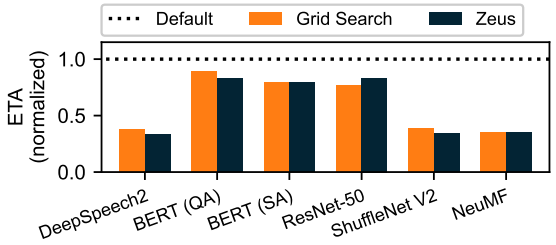
Figure B.9: Impact of priority knob η on ETA and TTA.



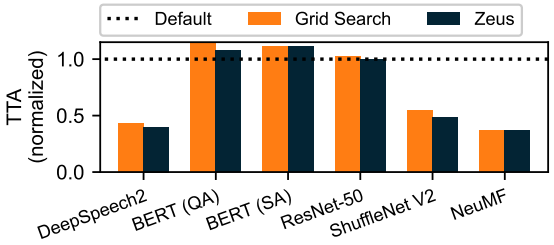
(a) Energy Consumption (V100)



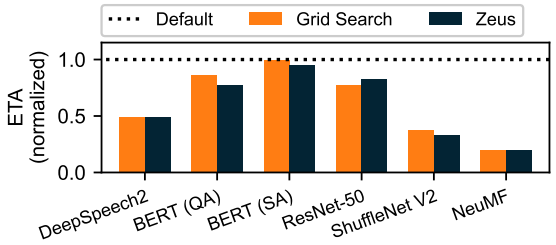
(b) Training Time (V100)



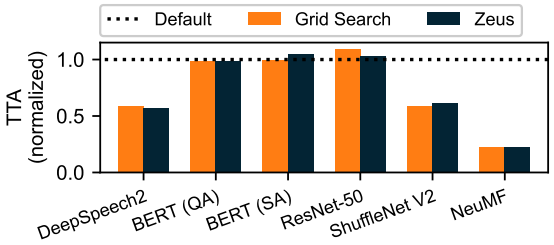
(c) Energy Consumption (A40)



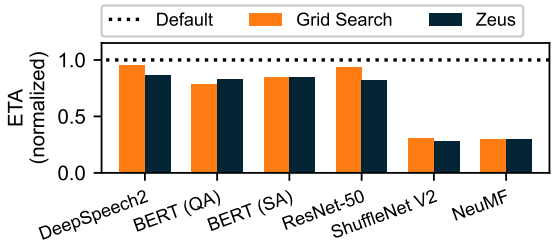
(d) Training Time (A40)



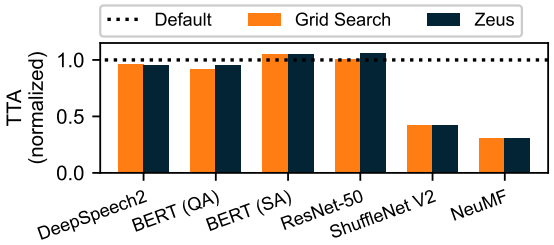
(e) Energy Consumption (RTX6000)



(f) Training Time (RTX6000)



(g) Energy Consumption (P100)



(h) Training Time (P100)

Figure B.10: Energy and time consumption of DNN training, normalized against Default for DNN training. Results measured on (a) NVIDIA A40 GPU, (b) NVIDIA V100 GPU, (c) NVIDIA RTX6000 GPU and (d) NVIDIA P100 GPU.

BIBLIOGRAPHY

[1] Alibaba Hybrid Cloud. https://www.alibabacloud.com/solutions/hybrid.

[2] Amazon datacenter locations. https://aws.amazon.com/about-aws/global-infrastructure/.

[3] Apache Hadoop. https://hadoop.apache.org/.

[4] Apache Hive. http://hive.apache.org.

[5] Apache Tez. http://tez.apache.org.

[6] AT&T MPLS backbone. http://www.topology-zoo.org/dataset.html.

[7] AWS Direct Connect. https://aws.amazon.com/directconnect/.

[8] AWS Hybrid Cloud Architecture. https://aws.amazon.com/enterprise/hybrid/.

[9] Azure ExpressRoute. https://azure.microsoft.com/en-us/services/expressroute/.

[10] Azure Hybrid Cloud. https://azure.microsoft.com/en-us/overview/hybrid-cloud/.

[11] Ceph. https://ceph.io/.

[12] CloudLab. https://cloudlab.us/.

[13] EC2 Direct Connect Pricing. https://aws.amazon.com/directconnect/pricing/.

[14] General Data Protection Regulation. https://gdpr-info.eu/.

[15] Google datacenter locations. http://www.google.com/about/datacenters/inside/locations/.

[16] Google Hybrid Cloud. https://cloud.google.com/anthos/.

[17] Guava: Google core library for Java. https://guava.dev/.

[18] How much electricity does an American home use? https://www.eia.gov/tools/faqs/faq.php?id=97&t=3.

[19] IBM Hybrid Cloud. https://www.ibm.com/it-infrastructure/z/capabilities/hybrid-cloud.

[20] Intel Rack Scale Design. https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html.

[21] Internet Users. http://www.internetlivestats.com.

[22] Kubernetes Adoption and Security Trends and Market Share for Containers. https://www.stackrox.com/kubernetes-adoption-and-security-trends-and-market-share-for-containers

[23] Memcached. https://memcached.org/.

[24] Microsoft Azure Regions. https://azure.microsoft.com/en-us/global-infrastructure/regions/.

[25] Microsoft SQL Server Stored Procedures. https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine.

[26] Microsoft SQL Server User-Defined Functions. https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions.

[27] Multi-cloud Statistics. https://enterprisersproject.com/article/2019/8/multi-cloud-statistics/.

[28] Multicloud Everything You Need to Know about the Biggest Trend tn Cloud Computing. https://www.zdnet.com/article/multicloud-everything-you-need-to-know-about-the-biggest-trend-in-clou

[29] MySQL Stored Procedures Tutorial. https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx/.

[30] NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml.

[31] NVIDIA System Management Interface. https://developer.nvidia.com/nvidia-system-management-interface.

[32] Oracle PL/SQL. https://www.oracle.com/database/technologies/application-development-PL/SQL.html.

[33] Private Cloud Deployments Lose Luster as Enterprises Lean Toward Public Multi-cloud Approaches Says. https://www.zdnet.com/article/private-cloud-deployments-lose-luster-as-enterprises-lean-toward-publi

[34] Redis. https://redis.io/.

[35] Statistical Workload Injector for MapReduce (SWIM). https://github.com/SWIMProjectUCB/SWIM.

[36] The Many Flavors of Multi-cloud. https://redislabs.com/blog/the-many-flavors-of-multi-cloud/.

[37] TPC Benchmark. http://www.tpc.org.

[38] Why Organizations Choose a Multicloud Strategy. https://www.gartner.com/smarterwithgartner/why-organizations-choose-a-multicloud-strategy/.

[39] Mesosphere 2018: Cloud native ecosystem report. https://www.ciosummits.com/Mesosphere-Apache-Mesos-2018-Survey-Report-Web.pdf, 2018.

[40] Heterogeneous Deployment Patterns with Kubernetes. https://cloud.google.com/solutions/heterogeneous-deployment-patterns-with-kubernetes, 2019.

[41] Idc forecasts worldwide spending on the internet of things to reach $745 billion in 2019. https://www.idc.com/getdoc.jsp?containerId, 2019.

[42] Atul Adya, Daniel Myers, Henry Qin, and Robert Grandl. Fast key-value stores: An idea whose time has come and gone. In *HotOS*, 2019.

[43] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings, 2012.

[44] Shipra Agrawal and Navin Goyal. Further optimal regret bounds for thompson sampling. In *Artificial intelligence and statistics*, pages 99–107. PMLR, 2013.

[45] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. Millwheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013.

[46] Rajagopal Ananthanarayanan, Venkatesh Basker, Sumit Das, Ashish Gupta, Haifeng Jiang, Tianhao Qiu, Alexey Reznichenko, Deomid Ryabkov, Manpreet Singh, and Shivakumar Venkataraman. Photon: Fault-tolerant and scalable joining of continuous data streams. In *SIGMOD*, 2013.

[47] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

[48] Yehia Arafa, Ammar ElWazir, Abdelrahman ElKanishy, Youssef Aly, Ayatelrahman Elsayed, Abdel-Hameed Badawy, Gopinath Chennupati, Stephan Eidenbenz, and Nandakishore Santhi. Verified instruction-level energy consumption measurement for nvidia gpus. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 60–70, 2020.

[49] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational data processing in Spark. In *SIGMOD*, 2015.

[50] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *NeurIPS*, 2020.

[51] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[52] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the killer microseconds. *Communications of the ACM*, 60(4):48–54, 2017.

[53] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. *NeurIPS*, 2014.

[54] Srikant Bharadwaj, Shomit Das, Yasuko Eckert, Mark Oskin, and Tushar Krishna. Dub: Dynamic underclocking and bypassing in nocs for heterogeneous gpu workloads. In *2021 15th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 49–54. IEEE, 2021.

[55] Ankit Bhardwaj, Chinmay Kulkarni, and Ryan Stutsman. Adaptive placement for in-memory storage functions. In *USENIX ATC*, 2020.

[56] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the first workshop on applications of pattern analysis*, pages 44–50. PMLR, 2010.

[57] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. Tao: Facebook's distributed data store for the social graph. In *USENIX ATC*, 2013.

[58] Qingqing Cao, Aruna Balasubramanian, and Niranjan Balasubramanian. Towards accurate and reliable energy measurement of NLP models. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 141–148, Online, November 2020. Association for Computational Linguistics.

[59] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180, 2020.

[60] Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

[61] Sai Rahul Chalamalasetti, Kevin Lim, Mitch Wright, Alvin AuYoung, Parthasarathy Ranganathan, and Martin Margala. An fpga memcached appliance. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 245–254, 2013.

[62] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. *NeurIPS*, 2011.

[63] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, 2018.

[64] M. Chowdhury, S. Khuller, M. Purohit, S. Yang, and J. You. Near optimal coflow scheduling in networks. In *SPAA*, 2019.

[65] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with Orchestra. In *SIGCOMM*, 2011.

[66] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Operating Systems Review*, 53(1):14–25, 2019.

[67] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally distributed database. *ACM TOCS*, 31(3):8, 2013.

[68] Yong Cui, Lian Wang, Xin Wang, Hongyi Wang, and Yining Wang. FMTCP: A fountain code-based multipath transmission control protocol. *IEEE/ACM Transactions on Networking (ToN)*, 23(2):465–478, 2015.

[69] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[70] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[71] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[72] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Röhm. Ycsb+ t: Benchmarking web-scale transactional databases. In *2014 IEEE 30th International Conference on Data Engineering Workshops*, pages 223–230. IEEE, 2014.

[73] John C Doyle, Bruce A Francis, and Allen R Tannenbaum. *Feedback Control Theory*. Courier Corporation, 2013.

[74] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *NSDI*, 2014.

[75] Daniel Fink. A compendium of conjugate priors. 1997.

[76] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, January 2013.

[77] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004.

[78] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *OSDI*, 2016.

[79] Roxana Geambasu, Amit A Levy, Tadayoshi Kohno, Arvind Krishnamurthy, and Henry M Levy. Comet: An active distributed key-value store. In *OSDI*, 2010.

[80] Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *SIGCOMM*, 2015.

[81] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[82] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

[83] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[84] 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*, 2017.

[85] Diego Granziol, Stefan Zohren, and Stephen Roberts. Learning rates as a function of batch size: A random matrix theory approach to neural network training. *arXiv preprint arXiv:2006.09092*, 2020.

[86] 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 *NSDI*, 2019.

[87] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *SIGCOMM*, 2016.

[88] Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Govind Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal, Sanjay Bhansali, Mingsheng Hong, Jamie Cameron, Masood Siddiqi, David Jones, Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, and Divyakant Agrawal. Mesa: Geo-replicated, near real-time, scalable data warehousing. *Proceedings of the VLDB Endowment*, 7(12):1259–1270, 2014.

[89] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. IEEE, 2020.

[90] Jing Han, Ee Haihong, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th international conference on pervasive computing and applications*, pages 363–366. IEEE, 2011.

[91] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[92] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[93] Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing. 1999.

[94] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[95] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[97] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[98] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

[99] Miro Hodak, Masha Gorkovenko, and Ajay Dholakia. Towards power efficiency in deep learning on data center hardware. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1814–1820. IEEE, 2019.

[100] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *NeurIPS*, 2017.

[101] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[102] Sunpyo Hong and Hyesoon Kim. An integrated gpu power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 280–289, 2010.

[103] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. Ganger, P. Gibbons, and O. Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. In *NSDI*, 2017.

[104] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, 2018.

[105] Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, and Jingren Zhou. Yugong: Geo-distributed data and job placement at scale. *Proceedings of the VLDB Endowment*, 12(12), 2019.

[106] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

[107] Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. Wide-area analytics with multiple resources. In *EuroSys*, 2018.

[108] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. In *SoCC*, 2015.

[109] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. Tcp≈rdma: Cpu-efficient remote storage access with i10. In *NSDI*, 2020.

[110] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[111] Hamidreza Jahanjou, Erez Kantor, and Rajmohan Rajaraman. Asymptotically optimal approximation algorithms for coflow scheduling. In *SPAA*, 2017.

[112] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[113] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[114] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *SIGCOMM*, 2016.

[115] 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 *SOSP*, 2019.

[116] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. Optimizing bulk transfers with software-defined optical WAN. In *SIGCOMM*, 2016.

[117] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter rpcs can be general and fast. In *NSDI*, 2019.

[118] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using rdma efficiently for key-value services. In *SIGCOMM*, 2014.

[119] Anuj Kalia, Michael Kaminsky, and David G Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided rdma datagram rpcs. In *OSDI*, 2016.

[120] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan PC Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, et al. H-store: a high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 1(2):1496–1499, 2008.

[121] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G Rogers, Tor M Aamodt, and Nikos Hardavellas. Accelwattch: A power modeling framework for modern gpus. In *MICRO-54*, 2021.

[122] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: optimizing neural network queries over video at scale. In *VLDB*, 2017.

[123] Dong-Ki Kang, Ki-Beom Lee, and Young-Chon Kim. Cost efficient gpu cluster management for training and inference of deep learning. *Energies*, 15(2):474, 2022.

[124] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S Gunawi, Cody Hammock, et al. Lessons learned from the chameleon testbed. In *USENIX ATC*, 2020.

[125] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[126] Krishnateja Killamsetty, S Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *ICML*, 2021.

[127] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[128] Leonard Kleinrock. Queueing systems. volume i: theory. 1975.

[129] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *EuroSys*, 2016.

[130] Konstantinos Kloudas, Margarida Mamede, Nuno Preguiça, and Rodrigo Rodrigues. Pixida: Optimizing data parallel jobs in wide-area data analytics. In *VLDB*, 2015.

[131] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *2013 IEEE 31st International Conference on computer design (ICCD)*, pages 349–356. IEEE, 2013.

[132] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[133] Chinmay Kulkarni, Sara Moore, Mazhar Naqvi, Tian Zhang, Robert Ricci, and Ryan Stutsman. Splinter: Bare-metal extensions for multi-tenant low-latency storage. In *NSDI*, 2018.

[134] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *SIGCOMM*, 2015.

[135] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *SIGCOMM*, 2020.

[136] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *NSDI*, 2018.

[137] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[138] F. Lai, J. You, X. Zhu, H. V. Madhyastha, and M. Chowdhury. Sol: Fast distributed computation over slow networks. In *NSDI*, 2020.

[139] Fan Lai, Mosharaf Chowdhury, and Harsha Madhyastha. To relay or not to relay for inter-cloud transfers? In *HotCloud*, 2018.

[140] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast distributed computation over slow networks. In *NSDI*, 2020.

[141] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *OSDI*, 2021.

[142] Tze Leung Lai, Herbert Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[143] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[144] Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, et al. Predictive and adaptive failure mitigation to avert production cloud vm interruptions. In *OSDI*, 2020.

[145] Wenxin Li, Xu Yuan, Keqiu Li, Heng Qi, Xiaobo Zhou, and Renhai Xu. Endpoint-flexible coflow scheduling across geo-distributed datacenters. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2466–2481, 2020.

[146] Wei Liang, Wenbo Yin, Ping Kang, and Lingli Wang. Memory efficient and high performance key-value store on fpga using cuckoo hashing. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.

[147] Weixin Liang and James Zou. Metashift: A dataset of datasets for evaluating contextual distribution shifts and training conflicts. *arXiv preprint arXiv:2202.06523*, 2022.

[148] Hyeontaek Lim, Dongsu Han, David G Andersen, and Michael Kaminsky. Mica: A holistic approach to fast in-memory key-value storage. In *NSDI*, 2014.

[149] S Liu, L Chen, and B Li. Siphon: Expediting inter-datacenter coflows in wide-area data analytics. In *USENIX ATC*, 2018.

[150] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[151] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[152] Cheng Luo and Reiji Suda. A performance and energy consumption analytical model for gpu. In *2011 IEEE ninth international conference on dependable, autonomic and secure computing*, pages 658–665. IEEE, 2011.

[153] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[154] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. In *MLSys*, January 2022.

[155] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. Cache craftiness for fast multicore key-value storage. In *EuroSys*. ACM, 2012.

[156] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *SIGMOD*, 2021.

[157] Charles Masson, Jee E Rim, and Homin K Lee. Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *Proceedings of the VLDB Endowment*, 12(12).

[158] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[159] Xinxin Mei, Qiang Wang, and Xiaowen Chu. A survey and measurement study of gpu dvfs on energy conservation. *Digital Communications and Networks*, 3(2):89–100, 2017.

[160] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *USENIX ATC*, 2013.

[161] Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen, and Jinyang Li. Balancing cpu and network in the cell distributed b-tree store. In *USENIX ATC*, 2016.

[162] Sparsh Mittal and Sumanth Umesh. A survey on hardware accelerators and optimization techniques for rnns. *Journal of Systems Architecture*, 112:101839, 2021.

[163] Thomas Moscibroda and Onur Mutlu. Distributed order scheduling and its application to multi-core DRAM controllers. In *PODC*, 2008.

[164] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. Batchsizer: Power-performance tradeoff for dnn inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 819–824, 2021.

[165] 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 *SOSP*, 2019.

[166] Mohammad Noormohammadpour, Cauligi S Raghavendra, Sriram Rao, and Srikanth Kandula. DCCast: Efficient point to multipoint transfers across datacenters. In *HotCloud*, 2017.

[167] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, et al. Storm: a fast transactional dataplane for remote data structures. In *SYSTOR*, 2019.

[168] Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

[169] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[170] Dongchul Park, Jianguo Wang, and Yang-Suk Kee. In-storage computing for hadoop mapreduce framework: Challenges and possibilities. *IEEE Transactions on Computers*, 2016.

[171] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

[172] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[173] 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 *SOSP*, 2019.

[174] George Prekas, Marios Kogias, and Edouard Bugnion. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *SOSP*, 2017.

[175] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Victor Bahl, and Ion Stoica. Low latency geo-distributed data analytics. In *SIGCOMM*, 2015.

[176] 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 *OSDI*, 2021.

[177] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S Pai, and Michael J Freedman. Aggregation and degradation in JetStream: Streaming analytics in the wide area. In *NSDI*, 2014.

[178] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM*, 2011.

[179] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2020.

[180] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[181] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[182] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. In *INM*, 2002.

[183] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[184] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[185] Simone Scardapane and Dianhui Wang. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2):e1200, 2017.

[186] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, nov 2020.

[187] Michael A Sevilla, Noah Watkins, Ivo Jimenez, Peter Alvaro, Shel Finkelstein, Jeff LeFevre, and Carlos Maltzahn. Malacology: A programmable storage system. In *EuroSys*, 2017.

[188] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[189] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[190] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *NSDI*, 2019.

[191] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[192] Michael Stonebraker and Greg Kemnitz. The postgres next generation database management system. *Communications of the ACM*, 34(10):78–92, 1991.

[193] Michael Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[194] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[195] Patrick Stuedi, Animesh Trivedi, and Bernard Metzler. Wimpy nodes with 10gbe: Leveraging one-sided operations in soft-rdma to boost memcached. In *USENIX ATC*, 2012.

[196] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M Rush, David Brooks, et al. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO*, 2021.

[197] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of gpu dvfs on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, pages 315–325, 2019.

[198] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[199] Ashish Thusoo, Raghotham Murthy, Joydeep Sen Sarma, Zheng Shao, Namit Jain, Prasad Chakka, Suresh Anthony, Hao Liu, and Ning Zhang. Hive – a petabyte scale data warehouse using Hadoop. In *ICDE*, 2010.

[200] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *SoCC*, 2013.

[201] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. Clarinet: WAN-aware optimization for analytics queries. In *OSDI*, 2016.

[202] Ashish Vulimiri, Carlo Curino, B Godfrey, J Padhye, and G Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI*, 2015.

[203] Ashish Vulimiri, Carlo Curino, Brighten Godfrey, Konstantinos Karanasos, and George Varghese. WANalytics: Analytics for a geo-distributed data-intensive world. In *CIDR*, 2015.

[204] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. WANalytics: Geo-distributed analytics for a data intensive world. In *SIGMOD*, 2015.

[205] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. Building an elastic query engine on disaggregated storage. In *NSDI*, 2020.

[206] Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. Alert: Accurate learning for energy and timeliness. In *USENIX ATC*, 2020.

[207] Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. Dynamic gpu energy optimization for machine learning training workloads. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[208] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, 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*, volume 2, pages 172–186, 2020.

[209] 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 *OSDI*, 2021.

[210] Qiang Wang and Xiaowen Chu. Gpgpu performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2865–2881, 2020.

[211] Yuxin Wang, Qiang Wang, Shaohuai Shi, Xin He, Zhenheng Tang, Kaiyong Zhao, and Xiaowen Chu. Benchmarking the performance and energy efficiency of ai accelerators for ai training. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 744–751. IEEE, 2020.

[212] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *NSDI*, 2022.

[213] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[214] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, James Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. Sustainable ai: Environmental implications, challenges and opportunities, 2021.

[215] Gene Wu, Joseph L Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. Gpgpu performance and power estimation using machine learning. In *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*, pages 564–576. IEEE, 2015.

[216] Xipeng Xiao, Alan Hannan, Brook Bailey, and Lionel M Ni. Traffic engineering with MPLS in the internet. *IEEE Network*, 14(2):28–33, 2000.

[217] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, et al. Gspmd: general and scalable parallelization for ml computation graphs. *arXiv preprint arXiv:2105.04663*, 2021.

[218] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[219] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[220] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing deadlines for inter-datacenter transfers. In *EuroSys*, 2015.

[221] 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 *OSDI*, 2020.

[222] Pengfei Zou, Ang Li, Kevin Barker, and Rong Ge. Indicator-directed dynamic power management for iterative workloads on gpu-accelerated systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 559–568. IEEE, 2020.