HYDROZOA: DYNAMIC HYBRID-PARALLEL DNN TRAINING ON SERVERLESS CONTAINERS

Runsheng Benson Guo 1  Victor Guo 1  Antonio Kim 1  Joshua Hildred 1  Khuzaima Daudjee 1

ABSTRACT
Deep Neural Networks (DNNs) are often trained in parallel on a cluster of virtual machines (VMs) so as to reduce training time. However, this requires explicit cluster management, which is cumbersome and often results in costly overprovisioning of resources. Training DNNs on serverless compute is an attractive alternative that is receiving growing interest. In a serverless environment, users do not need to handle cluster management and can scale compute resources at a fine-grained level while paying for resources only when actively used. Despite these potential benefits, existing serverless systems for DNN training are ineffective because they are limited to CPU-based training and bottlenecked by expensive distributed communication. We present Hydrozoa, a system that trains DNNs on serverless containers with a hybrid-parallel architecture that flexibly combines data- and model-parallelism. Hydrozoa supports GPU-based training and leverages hybrid-parallelism and serverless resource scaling to achieve up to 155.5× and 5.4× higher throughput-per-dollar compared to existing serverless and VM-based training systems. Hydrozoa also allows users to implement dynamic worker-scaling policies during training. We show that dynamic worker scaling improves statistical training efficiency and reduces training costs.

1 INTRODUCTION
Deep Neural Networks (DNNs) are used to solve problems in many different domains including image classification (He et al., 2016; Simonyan & Zisserman, 2015a), natural language processing (Collobert & Weston, 2008; Vaswani et al., 2017), self-driving cars (Luo et al., 2020; Yang et al., 2020), and query optimization (Krishnan et al., 2018; Marcus et al., 2019; Guo & Daudjee, 2020). State-of-the-art DNNs have progressively increased in both size and complexity, requiring more computational resources and time to train. Consequently, distributed training schemes such as data-parallelism and model-parallelism are popularly employed. In data-parallelism, workers train the model in parallel across partitions of the data, whereas in model-parallelism, the model is partitioned across workers (Huang et al., 2019). The effectiveness of each method depends on the model and the compute resources available. For example, (pipelined) model-parallelism can be effective for reducing communication overhead over existing-parallelism when training large DNN models. However, model-parallelism is difficult to scale since it relies on an effective model partition.

While distributed training can significantly reduce training time (Goyal et al., 2017), running such a task remains a challenge for many users. Distributed training is traditionally designed to run on a cluster of machines. Typically, virtual machines (VMs) from infrastructure-as-a-service (IaaS) are used for convenience over physical machines. However, this VM-based approach still requires configuring and managing a cluster in addition to deploying the training jobs. Cluster management is a time-consuming task, especially if resources need to be adjusted for different jobs. Often, clusters are suboptimally allocated and underutilized (Delimitrou & Kozyrakis, 2014), resulting in a waste of money on unused resources.

In the serverless computing paradigm, customers run program logic on cloud-managed servers. In contrast to VM-based approaches, users do not need to deal with the complexity of deploying, managing, and scaling their own servers. Code is automatically deployed and cluster resources are de-provisioned when execution finishes. Developers can scale serverless applications elastically while allocating resources at a fine-grained level to match resource requirements. Moreover, serverless services bill for only the time spent running program logic, which is cost-efficient compared to reserving a cluster that could be idle or underutilized for long periods of time.

Several systems have been developed for DNN training that take advantage of the elasticity, pricing model, and simplified management of serverless function platforms like

---

1 Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. Correspondence to: Runsheng Benson Guo <r9guo@uwaterloo.ca>.

Proceedings of the 5th MLSys Conference, Santa Clara, CA, USA, 2022. Copyright 2022 by the author(s).
AWS Lambda (lam, 2021), Google Cloud Functions (clo, 2021), and Azure Functions (azu, 2021). These systems train DNNs with data-parallelism, and demonstrate benefits in cost and training time over VM-based approaches using CPU-based training (Carreira et al., 2019; Feng et al., 2018; Wang et al., 2019). Serverless functions, however, are designed to be lightweight with limited computational power, memory and storage, and hence cannot be provisioned with GPUs. In addition, serverless functions cannot directly communicate with each other relying instead on an external storage channel for indirect communication, which incurs higher latencies. These limitations are problematic for DNN training, which is greatly accelerated by GPU-based computing and relies on frequent communication between workers. Consequently, state-of-the-art serverless systems for DNN training are still an order of magnitude slower than GPU-based training on VMs (Jiang et al., 2021).

In light of these concerns, we designed Hydrozoa, a serverless DNN training system that addresses the shortcomings of both VM-based and serverless systems. Hydrozoa achieves this by being the first system to combine serverless compute, hybrid-parallelism, and dynamic worker scaling:

(i) **Serverless Containers:** Training tasks in Hydrozoa are packaged as containers instead of functions. Containers are executed serverlessly on Azure Container Instances (aci, 2021). Like serverless functions, serverless containers can be scaled at a fine-grained level but importantly, they can be provisioned with significantly more compute resources including GPUs. Serverless containers can communicate directly with each other, thereby reducing communication costs.

(ii) **Hybrid-Parallel Training:** Hydrozoa supports data-parallel, model-parallel and hybrid-parallel training that flexibly combines data- and model-parallelism (Narayanan et al., 2019; Park et al., 2020). Hydrozoa employs a novel planner to optimize the degree of data- and model-parallelism to reap the benefits of both approaches. The planner includes a partitioning algorithm that automatically partitions a DNN model across workers for efficient model-parallel training.

(iii) **Dynamic Worker Scaling:** Hydrozoa leverages the elasticity of serverless compute to dynamically adjust the number of workers during training. Users can specify policies in Hydrozoa to control worker scaling behaviour to achieve high parallelism without sacrificing model convergence properties.

These three orthogonal components have limitations when used in isolation but when combined, effectively complement each other. Distributed communication between serverless resources can bottleneck training but this is mitigated in Hydrozoa with hybrid-parallelism, which reduces communication during training. Existing hybrid-parallel systems (Narayanan et al., 2019; Park et al., 2020) deployed on VM clusters are prone to resource over-provisioning, since tasks have specialized functionality with different resource requirements while VMs come in coarse-grained sizes. Hydrozoa avoids this shortcoming by right-sizing granularly-scalable serverless resources for each task. Dynamic worker scaling (Devarakonda et al., 2017; McCandlish et al., 2018) has been explored as a means to improve statistical training efficiency. However, existing methods are inefficient because they are deployed on fixed-size clusters that are over-provisioned to accommodate the maximum number of workers needed during training. Hydrozoa benefits from additional cost savings by dynamically scaling serverless resources in accordance with the number of workers that are actively training.

However, existing implementations of the aforementioned components cannot be trivially combined. For example, existing distributed training logic in MXNet and Pytorch assume the cluster size is fixed, and cannot handle workers on serverless compute dynamically joining and leaving during training. Likewise, existing hybrid-parallel implementations are built on top of these training libraries and are incompatible with serverless compute. Hydrozoa implements custom distributed training logic on top of MXNet that seamlessly integrates all three components.

Our experiments demonstrate that Hydrozoa delivers significant performance gains over both existing serverless approaches and industrial-strength VM-based training by Azure ML and SageMaker. Hydrozoa trains models with up to 38.1× higher throughput than existing serverless systems and 2.7× higher than VM-based systems while incurring a fraction of the cost and handling all cluster management.

## 2 BACKGROUND

This section describes the common approaches for parallelizing DNN training and the effect of batch size on training. We also present an overview of the benefits and challenges of DNN training in different cloud environments.

### 2.1 Data-Parallelism & Effect of Batch Size

Data-parallelism distributes training by placing a copy of the DNN on each worker, which computes model updates on its own subset of the training data. Workers synchronize model updates after every iteration of training, using either parameter servers (Li et al., 2014) or allreduce (Sergeev & Bals, 2018).

The global batch size of inputs processed during every iteration scales linearly with the number of workers. A larger batch size enables training with more workers in parallel
and hence leads to shorter training times to process the same number of input examples. However, it has been shown that larger batch sizes result in statistically inefficient training (Goyal et al., 2017). A larger batch size requires more epochs of training to reach a given accuracy and converges to a lower final accuracy. Adjusting the batch size during training can preserve the statistical efficiency of small batch sizes while achieving the training speed of large batch sizes (Devarakonda et al., 2017; Mai et al., 2020; McCandlish et al., 2018). For example, AdaBatch (Devarakonda et al., 2017) starts at a small batch size and gradually increases the batch size on a fixed schedule. KungFu (Mai et al., 2020) enables users to implement such strategies by dynamically adjusting the number of workers to maintain the desired batch size. However, it assumes a fixed size cluster of machines that must be over-provisioned to accommodate the maximum number of workers required during training. Hydrozoa avoids paying for idle resources by dynamically scaling compute to match real-time resource requirements.

### 2.2 Model-Parallelism

In model-parallelism, the model is partitioned into *shards*. Each worker is assigned a shard and handles the computation for that shard. Model-parallelism requires prudent partitioning of the DNN to ensure that the partition splits the computational work evenly while minimizing cross-worker communication. While there are an exponential number of partitions to consider and selecting an effective partition is challenging, model-parallelism generally requires much less data to be sent across workers than in data-parallelism (Narayanan et al., 2019).

### 2.3 DNN Training on Clusters

Data- and model-parallelism are effective for speeding up DNN training on a cluster of machines (Goyal et al., 2017; Narayanan et al., 2019). However, using clusters comes with challenges in management and code deployment. Users must decide what type of resources and how many to provision. Moreover, cluster resources need to be scaled to accommodate changes in training workload. This requires significant effort and is ameliorated in practice by spending extra money to over-provision resources so as to satisfy peak usage. Drivers and libraries need to be installed and machines need to be configured to communicate with each other prior to deploying code. These challenges detract deep learning practitioners from the task at hand and waste money on underutilized resources.

### 2.4 DNN Training on Serverless Functions

Training in a serverless environment confers several benefits. Serverless functions abstract away the complexity of managing machines and deploying code. Functions can be scaled without explicit cluster management. Resources can be adjusted individually per-function and at a fine-grained level to fit the requirements of the workload at hand. Moreover, the customer pays for compute resources only when they are actively being used.

**Existing Serverless Approaches** There have been several systems developed for DNN training using serverless functions (e.g. (Feng et al., 2018; Carreira et al., 2019)). All of these proposals distribute training with data-parallelism. Because serverless functions cannot communicate with each other directly, existing approaches pass data indirectly using an external storage service like AWS S3 (Jiang et al., 2021). Since there is an extra hop required for serverless communication, communication remains a bottleneck for these approaches. Furthermore, serverless function platforms impose strict resource limits on memory, CPU, storage, and cannot use GPUs. Jiang et al. report that existing serverless systems are unable to compete with VM-based training with GPUs in both performance and cost.

### 2.5 DNN Training on MLaaS

ML-as-a-service (MLaaS) provides a Cloud suite of industrial-strength tools for building machine learning pipelines. In particular, Azure ML (aml, 2021) and AWS SageMaker (sm, 2021) offer services for DNN training on a cloud-managed cluster of VMs. Users benefit from the convenience of serverless computing while also enjoying the performance of VM-based training. However, users are still susceptible to over-provisioning resources on MLaaS. Resources cannot be scaled dynamically, and must be provisioned in coarse-grained increments of a VM size, which come in limited configurations and are unlikely to fit the specific resource requirements of the workload at hand. For example, the AWS p2.xlarge VM offers 1 K80 GPU and 61 GB RAM. The next size up is p2.8xlarge, with 8 K80 GPUs and 488 GB RAM. It is highly unlikely either of these VMs match the resource requirements of a training job, resulting in users paying for extra resources they do not need.

### 3 Hydrozoa Design

Hydrozoa trains DNN models with hybrid-parallelism and dynamic worker scaling on top of serverless containers. With this architecture, Hydrozoa is able to avoid the traditional pitfalls of serverless DNN training, including ineffi-

<table>
<thead>
<tr>
<th>Platform</th>
<th>GPUs</th>
<th>Direct Communication</th>
<th>Fine-Grained Scaling</th>
<th>Pay Per-Use</th>
<th>Management Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM (IaaS)</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM (MLaaS)</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serverless Functions</td>
<td>✅</td>
<td>✅</td>
<td>✓</td>
<td>✅</td>
<td>✓</td>
</tr>
<tr>
<td>Serverless Containers</td>
<td>✅</td>
<td>✅</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table 1. Platform comparison for DNN training.*
cient communication and limited compute resources. At the same time, it leverages the benefits of serverless computing:
fine-grained scalability, zero cluster management overhead,
and a pay-per-use billing model. The rest of this section
describes the main components of Hydrozoa’s design.

3.1 Serverless Containers

Containers run applications in a virtual environment, pack-
aged with their libraries and dependencies. Azure Container
Instances (ACI) (aci, 2021) is a platform that allows users to
execute containers in a serverless fashion, much like server-
less functions. A user submits a container along with a set
of resource requests, and ACI will provision the requested
resources and manage the execution of the container on
those resources. Users are charged based on the GPU, CPU
and memory resources used for the execution of the con-
tainer on a per second basis. Hence, ACI enables code to
be executed serverless with as much ease, scalability and
cost-efficiency as serverless functions. Moreover, ACI does
not have the same stringent start-up requirements as server-
less functions, and can be configured with both GPUs and a
public IP for communication. We refer to this approach as
serverless containers. Hydrozoa is built on top of serverless
containers in ACI. We next describe why this design choice
is particularly suitable for DNN training.

GPU Compute DNN training is much faster on GPUs
compared to CPUs since DNN computations are highly par-
allelizable, making it a highly suitable workload for exploit-
ing the high compute throughput and memory bandwidth
in GPUs. Serverless containers in ACI can be provisioned
with GPUs.

Direct Communication Serverless function platforms do
not support direct communication between functions. Func-
tions must rely on external storage systems such as S3 to
indirectly exchange information, which incurs significant
latency. This is problematic for distributed DNN training,
which requires frequent communication between workers.
This is not an issue in ACI, since containers can communi-
icate with each other directly through public IP addresses.

Fine-Grained Scaling In ACI, containers can be provi-
sioned resources at a fine-grained level. Memory can be
configured in GB increments; GPUs and vCPUs can be
scaled independently. Hydrozoa right-sizes resource alloca-
tions to fit the specific compute requirements of each job,
saving costs by avoiding resource over-provisioning.

Pay-Per-Use ACI charges for resources only while they
are in use, and on a per-second basis. On the other hand,
VMs reserved on IaaS are billed regardless of whether or
not they are actively being used, and often on a per-hour
basis. The pay-per-use billing model provides yet another
avenue in which costs can be cut.

Management Free Training DNNs on IaaS requires pro-
visioning and configuring VMs, launching the distributed
tasks, and deprovisioning the VMs when done. This process
is time-consuming and prone to human error, especially
when launching multiple training jobs or scaling resources
on-the-fly. Hydrozoa automates this process on ACI.

Table 1 summarizes the availability of these features across
different platforms. Serverless containers is the only plat-
form that supports all of the features.

3.2 Hybrid-Parallelism

Hydrozoa trains DNNs with hybrid-parallelism – a combina-
tion of data- and model-parallelism. Figure 1 illustrates this
training architecture on a model partitioned into 3 shards.
Workers 1-1, 1-2, and 1-3 are each responsible for training
one model shard. They form a worker group, training one
copy of the model with model-parallelism. Data-parallelism
is then added on top of model-parallelism by instantiating
multiple such worker groups. Workers training the same
shard across different worker groups synchronize model
updates with each other using allreduce (Sergeev & Balso,
2018). The parallelism strategy in Figure 1 corresponds
to 2× data-parallelism applied on top of 3-shard model-
parallelism. We denote this as 2×3 for short and use this
notation in the rest of this paper. Since model-parallelism is
a special case of hybrid-parallelism where there is no data-
parallelism, and vice-versa. Hydrozoa is also able to train
models with pure data-parallelism or model-parallelism. Hy-
drozoa’s planner optimizes the parallelism strategy for each
training job. Section 3.4 provides details on the planner.

Model Partitioning To perform model-parallel training,
Hydrozoa first partitions the layers of a model $M$ into shards
$p_1, p_2, \ldots, p_k$ such that evaluating the model is equivalent
to evaluating the composition of the shards. In other words,
$M = p_k \circ \ldots \circ p_2 \circ p_1$. The model partition can have a
significant impact on the speed of model-parallel training.
However, finding an effective partition manually is difficult
since there are an exponential number of partitions to con-
Pipelined Model-Parallelism When model-parallelism is implemented naively, sequential dependencies in forward and backward propagation permit only one worker to perform computation at a time. Instead, we use the pipeline-parallelism approach that splits a minibatch of inputs into (smaller) microbatches (Huang et al., 2019; Narayanan et al., 2019), then pipeline microbatches across the workers. We further optimize the pipeline with workers prefetching activation and gradient inputs asynchronously during computation.

Figure 2 demonstrates pipelining for a model partitioned into 3 shards and a minibatch of data split into 3 microbatches. The pipeline starts on worker 1 with forward propagation on shard 1 with microbatch 1. The results can be passed to worker 2, which begins forward propagation on shard 2 with microbatch 1. At the same time, worker 1 can begin forward propagation on microbatch 2. Similarly, during backward propagation, worker 2 can begin backward propagation with microbatch 1 while worker 3 performs backward propagation with microbatch 2.

With the prefetching optimization, workers receive activation and gradient values as soon as they become available. With a well-balanced model partition, forward and backward propagation can start on subsequent microbatches without communication delay. In Figure 2, the gradients for microbatch 2 are transferred from worker 3 to worker 2 while worker 2 is doing backward propagation on microbatch 1. Consequently, worker 2 is able to start backward propagation on microbatch 2 immediately after microbatch 1.

3.3 Partitioning Algorithm

Hydrozoa’s partitioning algorithm aims to partition the model into shards $p_1, p_2, \ldots, p_k$ in a way that minimizes the total training time of a minibatch of inputs. To minimize training time, the partitioning algorithm ensures that computation is evenly distributed across each worker while minimizing the data communicated between workers. Partitions that evenly distribute the computation enable efficient microbatch pipelining, and minimizing the amount of data exchanged avoids communication bottlenecks.

Profiling The partitioning algorithm first profiles information about the model’s per-layer computation time, output sizes, and network speed. Let the profiled forward and backward propagation times for layer $i$ be $f_i$ and $b_i$. Then, for any shard $p_l$ that spans layers $x$ through $y$, the forward and backward propagation time of $p_l$ can be estimated as

$$f_{p_l} = \sum_{i=x}^{y} f_i$$

and

$$b_{p_l} = \sum_{i=x}^{y} b_i.$$  

The time $c_{p_l}$ to transfer activations and gradients between shard $p_{l-1}$ and shard $p_l$ is estimated to be $a_{p_{l-1}} / D$, where $a_{p_{l-1}}$ is the activation size of the last layer in shard $p_{l-1}$ and $D$ is network bandwidth.

Algorithm The partitioning algorithm aims to produce the partition that minimizes the runtime of processing a minibatch of inputs. This runtime is modeled with the equation below, which incorporates communication time, computation time, and pipeline stalls. $t_k$ and $r_1$ model the time between consecutive batches of forward and backward propagation.

$$\sum_{i=1}^{k} (f_{p_i} + b_{p_i}) + 2 \sum_{i=2}^{k} c_{p_i} + (M - 1)(t_k + r_1).$$

The derivation of this model is described in Appendix A.1. We use dynamic programming to find a partition that minimizes this runtime model. Details of this are provided in Appendix A.2. Our dynamic programming algorithm returns the overall best partition found in addition to the best partition found for each possible partition size. This allows the planner to consider partitions of different sizes when choosing a hybrid-parallelism strategy.

Theorem 1 Hydrozoa’s partitioning algorithm can produce a partition with runtime arbitrarily close to the optimal solution. Proof: In Appendix A.3.

Selecting Size and Number of Microbatches The optimal partition depends on the microbatch size and the number of microbatches. Hydrozoa performs an exponential search to select these hyperparameters, since they can have a big impact on performance, but are tedious to tune. The search generates partitions for microbatch sizes in increasing powers of 2, until they no longer fit in GPU memory. Pipelining is more efficient as the number of microbatches increases. Thus, for each microbatch size, we use binary search to maximize the number of microbatches within the GPU memory limits. The partitioning algorithm then returns the best partition produced across all the sizes and number of microbatches considered.

3.4 Planner

Hydrozoa’s planner considers the exponential search space of hybrid-parallel training strategies and selects one that is optimized for the training workload, VM type, and desired number of VMs to use for training. The planner first uses the partitioning algorithm to produce the most efficient partitions for each partition size. For each such partition,
the planner considers the hybrid-parallelism strategies using that partition for model-parallelism, and maximizes the level of data-parallelism while not exceeding the desired VMs. The planner uses a model for hybrid-parallel runtime to select the strategy with the lowest estimated runtime.

Hybrid-Parallelism Runtime Model   When processing a minibatch with hybrid-parallelism, Hydrozoa first runs model-parallelism within each worker group and then synchronizes weights across worker groups through data-parallelism. Thus, Hydrozoa’s model for hybrid-parallelism runtime combines the runtime model for model-parallelism described earlier, and a runtime model for data-parallelism.

Suppose we were training a model with parallelism-strategy \( A \times B \) (data-parallelism of \( A \) and model-parallelism of \( B \)). Worker \( w_{ij} \) corresponds to the worker on the \( i \)th worker group processing the \( j \)th partition. Moreover, let \( D \) be the network bandwidth, \( |p_k| \) be the size of partition \( p_k \) in bytes, and \( \alpha, \beta \) be model parameters.

We use the runtime model from the partitioning algorithm to compute the model-parallelism runtime of \( w_{ij} \) as \( T_{ij}^{(M)} \).

We use the following equation to model data-parallelism (implemented in Hydrozoa with ring allreduce) runtime:

\[
T_{ij}^{(D)} = (B - 1)(\frac{|p_j|}{B \cdot D} + \alpha \frac{|p_j|}{B} + \beta)
\]

The term \( \frac{|p_j|}{B \cdot D} \) models the data transfer time for each of the \( 2(B - 1) \) rounds in allreduce. \( \alpha \frac{|p_j|}{B} + \beta \) models any other overhead that occurs during every round of communication, such as serialization and deserialization. The planner learns \( \alpha, \beta \) through linear regression after collecting some runtime statistics on dummy data.

Thus, the total runtime of \( w_{ij} \) is then

\[
T_{ij} = T_{ij}^{(M)} + T_{ij}^{(D)}
\]

Then the entire round of hybrid-parallelism ends when all workers finish, which is modeled as

\[
T = \max_{i,j} T_{ij}
\]

Algorithm 1. Parallelism Strategy Selection

1. procedure SELECTHYBRIDSTRATEGY(clusterSize)
2.   bestStrategyTime ← inf
3.   for 1 ≤ mp ≤ clusterSize do
4.     dp ← \( \lfloor \text{clusterSize}/mp \rfloor \)
5.     runtime ← runtimeModel(dp, mp)
6.     if runtime < bestStrategyTime then
7.       bestStrategyTime ← runtime
8.       bestStrategy ← (dp, mp)
9. return bestStrategy

Selecting Parallelism Strategy   Algorithm 1 shows how the planner selects a hybrid-parallelism strategy for a desired cluster size based on the runtime model. It considers all possible combinations of data- and model-parallelism that fit within the cluster size and returns the strategy with the lowest estimated runtime.

3.5 Dynamic Worker Scaling

Hydrozoa allows users to specify policies for adjusting the number of workers at each epoch boundary. Hydrozoa adjusts serverless resources dynamically in response to decisions to scale up or down workers, and coordinates workers such that they are aware of changes in the training topology. Users can use worker scaling in Hydrozoa to implement strategies that improve statistical training efficiency (Devarakonda et al., 2017; McCandlish et al., 2018). Dynamic worker scaling would be difficult to implement otherwise, since the user would need to manually scale the cluster, and popular ML frameworks such as MXNet and PyTorch assume the number of workers are fixed throughout training.

3.6 API

Hydrozoa exposes an API for data loading, planning, hybrid-parallel training, and worker scaling. It was used to train image-classification and language models in Section 5.
4 HYDROZOA IMPLEMENTATION

Hydrozoa’s implementation consists of a (1) worker task, (2) planner, and (3) coordinator. Worker tasks execute training logic for a model shard and run on containers in ACI. The planner and coordinator also run as separate ACI containers. The coordinator manages the worker tasks throughout the training process. Training metadata and results are stored on S3. Figure 3 illustrates Hydrozoa’s architecture.

4.1 Worker

Each worker is part of a worker group and assigned a model shard. The worker performs model-parallelism within its worker group and data-parallelism with the corresponding shard in other worker groups. The machine learning library MXNet is used to implement the training process.

Communication  Workers exchange activations and gradients with each other through ZeroMQ (zer, 2021), a high-speed messaging library. We choose to use ZeroMQ since it is lightweight, heavily optimized, and offers convenient abstractions for implementing complex distributed communication patterns. Each worker is connected to the coordinator for querying network topology information and neighbouring workers for exchanging data during hybrid-parallel training.

Pipelined Model-Parallelism with Prefetching  Activations for microbatches are cached during forward propagation so that they can be reused in backward propagation. Separate threads on each worker handle the transfer of gradient and activation values asynchronously with forward and backward propagation. Likewise, a separate thread is responsible for prefetching training data.

4.2 Planner

The planner runs on ACI so that profiled computational and network statistics are consistent with the actual training environment. After determining the model partitioning and hybrid-parallel strategy selection, the planner collects memory utilization statistics for each shard, which is used by the coordinator for resource allocation decisions. In practice, the planner runs within a few minutes in all of our experiments, which is an insignificant amount of time compared to the overall training time of the models.

4.3 Coordinator

The Coordinator orchestrates the training job, running as a container on minimal resources in ACI. It is responsible for starting worker tasks with the appropriate configuration based on model partition information and training hyperparameters. The coordinator right-sizes the memory allocation for workers based on data profiled from the planner. Workers share their IP and port information with the coordinator that forwards this information to other workers that need it.

Dynamic Worker Scaling  When dynamic worker scaling is enabled, the coordinator is responsible for adjusting provisioned resources as the number of workers change. GPU containers on ACI can take several minutes to start up, so the coordinator will pre-start workers to ensure they are ready when actually needed. The coordinator collects information about worker startup time and training speeds to anticipate when to pre-start workers. The coordinator also informs existing workers about new workers so that they are aware of changes in network topology for allreduce and are assigned new subsets of the dataset.

5 PERFORMANCE EVALUATION

This section evaluates the performance of Hydrozoa on DNN training for image classification and question answering tasks. In Section 5.2, we demonstrate why Hydrozoa is superior to existing systems built on serverless functions. In Section 5.3, we show that Hydrozoa is able to improve on VM-based training throughput while costing significantly less through hybrid-parallelism and fine-grained resource allocation. Finally, in Section 5.4, we explore the performance benefits achievable through dynamic worker scaling.

5.1 Experimental Setup

Models: For evaluation, we train 5 popularly used models for image classification and question answering. We use AlexNet (Krizhevsky et al., 2012), ResNet-34, ResNet-152 (He et al., 2016), VGG19 (Simonyan & Zisserman, 2015b), and BERT-12 (Devlin et al., 2019), featuring varying sizes and computational requirements shown in Table 2.

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Parameters</th>
<th>FLOPs / Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Classification</td>
<td>AlexNet</td>
<td>57M</td>
<td>0.7G</td>
</tr>
<tr>
<td></td>
<td>ResNet-34</td>
<td>21.5M</td>
<td>3.7G</td>
</tr>
<tr>
<td></td>
<td>ResNet-152</td>
<td>58.5M</td>
<td>11.3G</td>
</tr>
<tr>
<td></td>
<td>VGG19</td>
<td>143.7M</td>
<td>19.6G</td>
</tr>
</tbody>
</table>

| Question Answering | BERT-12   | 108.9M     | 65.3G         |

Table 2. A comparison of the models used for evaluation.

<table>
<thead>
<tr>
<th>vCPU</th>
<th>Memory (GB)</th>
<th>K80</th>
<th>V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Container</td>
<td>$0.04050</td>
<td>$0.00445</td>
<td>N/A</td>
</tr>
<tr>
<td>GPU Container</td>
<td>$0.03165</td>
<td>$0.00425</td>
<td>$0.36</td>
</tr>
</tbody>
</table>

Table 3. Cost per hour for ACI resources used by Hydrozoa.
We define throughput-per-dollar as the number of inputs the system can process per dollar spent, i.e., $T/C$, where $C$ is the per-second cost of running the system. Throughput is measured as the number of inputs processed per second.

Datasets: We use the CIFAR-10 (Krizhevsky, 2012) dataset for image classification and SQuAD (Rajpurkar et al., 2016) for question answering. CIFAR-10 images are scaled to standard size $3 \times 224 \times 224$ colour images.

Batch Sizes & Evaluation Methodology: For data-parallel training, we use the largest per-GPU minibatch that fits within memory to optimize utilization of the GPUs. For hybrid-parallel training, Hydrozoa selects the microbatch size and number of microbatches processed per worker group. These numbers are reported in Table 5.

The main evaluation metrics we use for system comparison are throughput and throughput-per-dollar. Throughput ($T$) is measured as the number of inputs processed per second. We define throughput-per-dollar as the number of inputs the system can process per dollar spent, i.e., $T/C$, where $C$ is the per-second cost of running the system. Throughput is used to compare the raw speed of each system, while throughput-per-dollar enables comparing performance normalized by cost. Results for system comparisons are averaged over at least 3 independent experimental runs. Costs are computed based on current Cloud pricing in the US East region.

5.2 Comparison to Training on Serverless Functions

In Section 2.4, we identified the main limitations of training DNNs on serverless functions to be (1) indirect communication, and (2) the inability to train on GPUs. We demonstrate that Hydrozoa breaks away from these limitations to improve performance significantly using these benchmarks:

(i) Hydrozoa-Lambda (H-L): A re-implementation of Hydrozoa using serverless functions. H-L is built on Lambda, passes data through S3, and is representative of state-of-the-art serverless training systems (Jiang et al., 2021). Each worker is configured to have the maximum compute of 6 vCPUs and 10GB RAM, which cannot be scaled independently on Lambda.

(ii) Hydrozoa-CPU (H-C): Hydrozoa, but limited to training on CPUs. We configure each worker to have 4 vCPUs that is the maximum for containers without GPUs. This runs on serverless containers, and can benefit from direct communication between workers.

(iii) Hydrozoa-GPU (H-G): Hydrozoa, using a single worker with a K80 GPU and 4 vCPUs.

We compare data-parallelism throughput and throughput-per-dollar for H-L and H-C as the number of workers are scaled from 1 to 16, on ResNet-34 in Figure 4, and AlexNet in Figure 5. We also plot the performance of H-G with a single worker as a baseline to compare against. The cost of ACI resources used by Hydrozoa are given in Table 3. Note that ACI charges more for vCPU and memory resources on CPU-only containers. For Lambda, each worker provisioned with 10 GB of RAM costs $0.60 per hour.

Hydrozoa-Lambda vs Hydrozoa-CPU As the number of workers increases, the throughput gap between H-L and H-C widens since H-L scales poorly. With 16 workers, the throughput of H-C is $1.34 \times$ higher than H-L for ResNet-34 (Fig. 4). Poorer scaling can be attributed to differences in communication. Messages in H-L are exchanged indirectly through S3, requiring higher latency than direct communication. This inefficiency is amplified as the number of workers increases, since communication during allreduce increases with the number of workers. This is also echoed in throughput-per-dollar. While the throughput-per-dollar increases for H-L with an increase in workers, it stays relatively constant for H-C due to near-linear scaling. On AlexNet (Fig. 5), the throughput scaling of H-L is even worse. This is not surprising since AlexNet is relatively large in size compared to computation – it is almost 3 times as large as ResNet-34 in size but requires 80% fewer FLOPs to train. With 16 workers, the throughput of H-C is $4.6 \times$ higher than that of H-L on AlexNet. Cheaper resources on Azure ML over Lambda yield even higher throughput-per-dollar improvements of $4.1 \times$ and $11.5 \times$ for these models.

Hydrozoa-CPU vs Hydrozoa-GPU Comparing H-C to H-G demonstrates the benefit of using GPUs over CPUs. H-G with a single K80 GPU delivers $1.8 \times$ and $2.5 \times$ higher...
throughput than H-C with 16 workers for ResNet-34 and AlexNet, respectively. This performance also comes at significantly lower cost – compared to H-C with 16 workers, the throughput-per-dollar of H-G is $9.8 \times$ and $13.5 \times$ higher. From both a performance and cost perspective, GPUs should be the choice over CPUs to accelerate DNN training.

These experiments show why Hydrozoa is superior to state-of-the-art DNN training systems on serverless functions. Hydrozoa is able to scale better with an increasing number of workers due to more efficient communication. Moreover, by training on GPUs instead of CPUs, Hydrozoa is able to further boost throughput while reducing costs. As a result, H-G is able to train ResNet-34 and AlexNet with throughput-per-dollar that is $40.4 \times$ and $155.5 \times$ higher, respectively, than H-L.

5.3 Comparison to Training on VMs

We compare Hydrozoa to VM-based training in Azure ML and AWS SageMaker:

(i) Azure ML: We run distributed training in PyTorch with Horovod (Sergeev & Balso, 2018), that, like Hydrozoa, employs data-parallelism with allreduce.¹

(ii) AWS SageMaker: We also train with PyTorch on SageMaker in order to use their optimized distributed training API, which supports data-parallelism, model-parallelism, and also hybrid-parallelism.

Azure ML and SageMaker are MLaaS services on Azure and AWS, so the deployment and execution of the distributed training jobs are managed by their respective cloud service provider. There should be no performance difference between training on Azure ML/SageMaker and running the equivalent training logic on a self-managed cluster of VMs provisioned from IaaS. In fact, training performance is expected to be better on SageMaker because we are able to take advantage of their custom distributed training libraries that are industrial-strength and optimized for AWS’ network infrastructure. Experiments are run on clusters with K80 and V100 GPUs. Details on the clusters used for evaluation are provided in Appendix B.1.

Unlike Hydrozoa, Azure ML and SageMaker do not select a parallelism strategy for the user. We tried all parallelism strategies for these systems and compared the results obtained from their best-performing configuration to Hydrozoa. The results for K80 clusters are summarized in Table 5. Results for V100 clusters are provided in Appendix B.2. Hydrozoa achieves better throughput and throughput-per-dollar through a combination of fine-grained resource allocation and hybrid-parallelism. We break down the contribution of each of these factors next.

Fine-Grained Resource Allocation: ResNet-34 is a relatively small model with low computational requirements, so data-parallelism is more effective than model- and hybrid-parallelism for all systems. Figure 6 compares the throughput and throughput-per-dollar as the number of workers used for data-parallelism are scaled. Hydrozoa’s throughput scales similarly to the VM-based benchmarks, since they all use the same parallelism strategy. However, Hydrozoa attains this performance at much lower cost due to its fine-grained resource allocation. For example, the p2.xlarge VM in SageMaker is the smallest sized VM containing a K80 GPU, yet it comes with 61 GB RAM. Since training occurs on the GPU, only a fraction of that memory is used. Hydrozoa uses profiling data to allocate no more resources than needed – a K80, 4 VCPUs, and 6 GB of RAM per worker. Consequently, each worker in Hydrozoa costs $0.512/hour compared to more than double the price tag of $1.125/hour for each SageMaker VM. With these cost savings, Hydrozoa achieves $1.81 \times$ and $2.26 \times$ higher throughput-per-dollar than Azure ML and SageMaker, respectively.

Hybrid-Parallelism: ResNet-152, VGG19, and BERT-12 are much larger models than ResNet-34, causing communication to be a bottleneck in data-parallel training. Hydrozoa mitigates this issue by using hybrid-parallelism to greatly reduce the communication overhead while balancing computation evenly across workers. Azure ML supports only data-parallelism. On the other hand, SageMaker supports hybrid-parallelism but it was ineffective for training any of the models other than ResNet-152.

Figure 7 compares the throughput of Hydrozoa to SageMaker for model- and hybrid-parallelism on BERT-12. The left graph compares the two systems as the partition size for model-parallelism increases. Hydrozoa’s partitioning algorithm is able to produce partitions that scale throughput effectively up to 4 shards. 8 shards improves the throughput marginally over 4. SageMaker’s model-parallelism cannot

¹Azure ML does not support MXNet.
In this section we show how dynamic worker scaling in Hydrozoa achieves performance gains and cost savings. We implemented an adaptation of AdaBatch (Devarakonda et al., 2017) into Hydrozoa in which the equivalent batch size scaling is achieved by increasing the number of workers and not the per-worker minibatch size.

We train ResNet-34 on CIFAR-10 with the Adam Optimizer, an initial learning rate of $10^{-3}$ and a minibatch size of 256 per worker. Our baselines are data-parallelism with 1 (DP-1) and 16 workers (DP-16). All experiments are on Hydrozoa so that the performance results are attributed solely to worker scaling and not affected by system differences. We decay the learning rate every 5 epochs by a factor of 0.375 to avoid stagnation during training. We compare this to DP-(1-16), which starts with 1 worker, and gradually scales to 16 workers. Instead of decaying the learning rate by a factor of 0.375, we perform the equivalent of doubling the batch size and decaying the learning rate by a factor of 0.75 (Smith et al., 2018). We double the batch size by doubling the number of workers, which doubles the parallelism.

The training curves are shown in Figure 8. While DP-16 maximizes training speed, the statistical efficiency is poor. It reaches a lower accuracy for the same number of epochs processed compared to DP-1 and converges to a lower accuracy of 71% vs 85%. On the other hand, DP-1 achieves good statistical efficiency but is slow because there is only one worker training. DP-(1-16) achieves the best of both approaches. By scaling the number of workers, it can maintain the same statistical efficiency as DP-1 while reaching the training speed of DP-16. Thus, DP-(1-16) converges to an accuracy of 85% in about 45% less time than DP-1. Moreover, since DP-(1-16) is more statistically efficient than DP-16, it converges to an accuracy of 71% in fewer epochs than DP-16 and at 96% lower cost. While other scaling strategies can be implemented, this experiment demonstrates that dynamic worker scaling in Hydrozoa can bring significant benefits in training efficiency and cost.

### Table 4. Summary of results comparing Hydrozoa to VM-based training on Azure ML and AWS SageMaker on K80 clusters. Strategy refers to the data × model parallelism employed. DP and HP correspond to data- and hybrid-parallelism, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>Hydrozoa</th>
<th># Microbatches</th>
<th>Compared System</th>
<th>Compared Strategy</th>
<th>Throughput Speedup</th>
<th>Cost Reduction</th>
<th>Throughput / Dollar Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-34</td>
<td>DP - 16×1</td>
<td>1×256</td>
<td>Azure ML</td>
<td>DP - 16×1</td>
<td>1.03×</td>
<td>1.76×</td>
<td>1.81×</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>HP - 4×4</td>
<td>16×4</td>
<td>Azure ML</td>
<td>DP - 16×1</td>
<td>1.03×</td>
<td>1.70×</td>
<td>1.75×</td>
</tr>
<tr>
<td>VGG19</td>
<td>HP - 8×2</td>
<td>70×2</td>
<td>Azure ML</td>
<td>DP - 16×1</td>
<td>1.05×</td>
<td>2.15×</td>
<td>2.25×</td>
</tr>
<tr>
<td>BERT-12</td>
<td>HP - 4×4</td>
<td>13×4</td>
<td>Azure ML</td>
<td>DP - 16×1</td>
<td>1.70×</td>
<td>2.04×</td>
<td>3.47×</td>
</tr>
</tbody>
</table>
6 RELATED WORK

Serverless Computing Serverless computing has been applied in many areas, including IoT (Cheng et al., 2019), video processing (Ao et al., 2018), query execution (Perron et al., 2020), and machine learning. Multiple systems have been proposed for training DNN models on serverless functions with data-parallelism. In Sire (Wang et al., 2019), workers use S3 as a parameter server. (Feng et al., 2018) use multiple layers of parameter servers to reduce the data transfer time. (Carreira et al., 2019) reduces inter-worker communication latency by using a parameter server deployed on an EC2 cluster. (Jiang et al., 2021) compare the performance of serverless functions versus IaaS for a variety of ML tasks. They conclude that serverless functions are only effective for workloads requiring limited communications. This is generally not true for training DNNs, which can be large in size and require many rounds of communication until convergence. Hydrozoa is the first serverless system for DNN training that demonstrates performance competitive with GPU-based training on VMs for large DNNs.

Distributed DNN Training Data-parallelism is supported by most machine learning frameworks, including MXNet, PyTorch, and TensorFlow. Like Hydrozoa, GPipe (Huang et al., 2019) uses a synchronized pipeline-parallelism scheme, but details of its partitioning algorithm are not disclosed. PipeDream (Narayan et al., 2019), PipeDream-2BW (Narayan et al., 2021) and PipeMare (Yang et al., 2019) propose architectures for asynchronous pipeline-parallelism. PipeDream’s partitioning algorithm is optimized for asynchronous training and would be difficult to retrofit to Hydrozoa’s architecture. PipeDream and HetPipe (Park et al., 2020) both support variants of hybrid-parallelism, however they are susceptible to resource over-provisioning because unlike Hydrozoa, they cannot right-size resource allocations.

Dynamic Batch Scaling Proposals exist for gradually scaling to larger batch sizes during DNN training to improve training throughput while maintaining good model convergence properties (Devarakonda et al., 2017; Lee et al., 2019; McCandlish et al., 2018). However, most approaches scale the batch size without increasing the amount of resources used for training, limiting the parallelization possible on a larger batch size. KungFu (Mai et al., 2020) supports scaling the amount of workers during training to extract more performance benefit from larger batch sizes. However, it assumes there are free resources available to deploy new workers. Hydrozoa also supports worker scaling but dynamically adjusts allocated resources so users do not pay for idle compute.

7 CONCLUSION

We presented Hydrozoa, a serverless system for training DNNs. Hydrozoa overcomes existing limitations of serverless DNN training with a novel architecture that combines serverless containers with hybrid-parallel training and supports dynamic worker scaling. Hydrozoa attains higher throughput at a lower cost, achieving throughput-per-dollar improvements of up to 5.4× over existing VM-based training and 155.5× over serverless approaches while relieving the user from the burden of managing machine clusters.

8 ACKNOWLEDGMENTS

This project was supported by funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), Ontario Graduate Scholarship (OGS), and W4A Microsoft Azure Credits. We thank Kevin Li, Jerry Xie, and Andy Wang for their technical contributions during their undergraduate research assistantships. We also thank our anonymous reviewers for their valuable feedback.

REFERENCES


Hydrozoa: Dynamic Hybrid-Parallel DNN Training on Serverless Containers


Hydrozoa: Dynamic Hybrid-Parallel DNN Training on Serverless Containers


Hydrozoa: Dynamic Hybrid-Parallel DNN Training on Serverless Containers


Appendix

A Supplementary Material for Partitioning Algorithm

A.1 Model-Parallelism Runtime Model

In this section the runtime model for model-parallelism that is used by the partitioning algorithm is derived.

Suppose there are $k$ workers for shards $p_1, p_2, ... , p_k$. The forward propagation of a minibatch finishes when worker $k$ finishes forward propagation.

It can be seen from Figure 2 that worker $k$ starts forward propagation at time $\sum_{i=1}^{k-1} t_i + \sum_{i=2}^{k} c_{p_i}$. In this figure, notice that in worker 2 there can be a slight delay between the forward propagation of microbatches. This delay occurs whenever a worker needs to wait for the activations from a previous worker. In fact, the time elapsed between the start of two consecutive forward microbatches in worker 2 equals the forward time of a microbatch in worker 1. This can be generalized. Denote the time elapsed between two consecutive forward microbatches in worker $i$ as $t_i$ (as labeled in Figure 2). In general, $t_i = \max(\max_{1 \leq j \leq i} f_{p_j}, \max_{2 \leq j \leq i} c_{p_j})$ – which is equivalent to the maximum of forward propagation and communication times encountered up to the $i^{th}$ shard. If the number of microbatches is $M$, it follows that worker $i$ will require $(M - 1)t_i + f_{p_i}$ time to perform forward propagation. Hence, worker $k$ finishes forward propagation at time $\sum_{i=1}^{k} f_{p_i} + \sum_{i=2}^{k} c_{p_i} + (M - 1)t_k$; this is also when the forward propagation of the minibatch finishes.

Similarly, denote the time elapsed between consecutive backward microbatches in worker $i$ as $r_i$. It follows that $r_i = \max(\max_{1 \leq j \leq k} b_{p_j}, \max_{i+1 \leq j \leq k} c_{p_j})$ and by symmetry, the backpropagation of the minibatch requires a total time of $\sum_{i=1}^{k} b_{p_i} + \sum_{i=2}^{k} c_{p_i} + (M - 1)r_1$.

Hence, given any partition, the total runtime (forward plus backward propagation) for a minibatch on that partition can be modeled with the equation

$$\sum_{i=1}^{k} (f_{p_i} + b_{p_i}) + 2 \sum_{i=2}^{k} c_{p_i} + (M - 1)(t_k + r_1).$$

A.2 Dynamic Programming Optimization

The dynamic programming algorithm aims to find partitions that minimize the runtime model derived in A.1. Since $\sum_{i=1}^{k} (f_{p_i} + b_{p_i})$ is constant for all partitions, it suffices to minimize $2 \sum_{i=2}^{k} c_{p_i} + (M - 1)(t_k + r_1)$.

It is difficult to optimize this term jointly due to the interdependencies between $c_{p_i}, t_k, r_1$. However, if $t_k, r_1$ were fixed, the partition that minimizes $2 \sum_{i=2}^{k} c_{p_i}$ (communication time) could be found through dynamic programming. The optimal partition must then be one of the partitions produced from minimizing the communication time for some $t_k, r_1$ combination. However, there are an exponential number of $t_k, r_1$ combinations to consider, so to reduce the search space, Hydrozoa picks $V$ evenly spaced values as bounds for $t_k$ and similarly, another $V$ bounds are used for $r_1$. We denote the bounds for $t_k$ as $B_{1}^{(V)}, ..., B_{1}^{(V)}$, where $B_{1}^{(j)} \in [\min_{1 \leq i \leq L} f_i, \max(\sum_{i=1}^{L} f_i, \max_{2 \leq i \leq L} c_i)]$ for all $1 \leq j \leq V$, where $c_i$ is the communication time between layer $i - 1$ and $i$.

Notice that $[\min_{1 \leq i \leq L} f_i, \max(\sum_{i=1}^{L} f_i, \max_{2 \leq i \leq L} c_i)]$ is the interval of all possible values for $t_k$. The bounds for $r_1$ are denoted as $B_{2}^{(1)}, ..., B_{2}^{(V)}$ and are defined in a similar fashion.

For each combination of $B_{1}^{(j)}$ and $B_{2}^{(j)}$, the partitioning algorithm uses dynamic programming to find the partition that minimizes communication time while satisfying $t_k \leq B_{1}^{(j)}$ and $r_1 \leq B_{2}^{(j)}$. In the dynamic programming algorithm we iteratively solve for $C(i, j)$, which denotes the minimal communication time for a partition of the first $j$ layers in the model with $i$ shards. Using the optimal subproblem property we have that

$$C(i, j) = \min_{i-1 \leq u \leq j-1} C(i-1, u) + c_{u+1}$$

In the end, all partitions found from every combination of $B_{1}^{(j)}$ and $B_{2}^{(j)}$ are compared, and Hydrozoa selects the partition with the lowest estimated minibatch runtime.

Notice that the dynamic programming algorithm computes the partition with minimal communication for each possible partition size en-route to the final answer. In addition to the overall best solution, the partitioning algorithm will also keep track of and return the best solution per partition size.

A.3 Proof of Theorem 1

Theorem 1 Hydrozoa’s partitioning algorithm can produce a partition with runtime arbitrarily close to the optimal solution.

Proof. Let $\epsilon > 0$ be arbitrary. Suppose the optimal solution uses $k_0$ shards and partition $p_1, p_2, ... , p_{k_0}$, let $t_{k_0}$ and $r_1$ be the corresponding $t_k$ and $r_1$ values for the optimal
Table 5. Summary of results comparing Hydrozoa to VM-based training on Azure ML and AWS SageMaker on V100 clusters. Strategy refers to the data × model parallelism employed. DP and HP correspond to data- and hybrid-parallelism, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>Hydrozoa Strategy</th>
<th># Microbatches × Microbatch Size</th>
<th>Compared System</th>
<th>Compared Strategy</th>
<th>Throughput Speedup</th>
<th>Cost Reduction</th>
<th>Throughput / Dollar Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG19</td>
<td>HP - 1 × 4</td>
<td>34 × 4</td>
<td>Azure ML</td>
<td>DP - 16 × 1</td>
<td>1.89 ×</td>
<td>1.02 ×</td>
<td>1.91 ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-12</td>
<td>MP - 1 × 4</td>
<td>24 × 2</td>
<td>Azure ML</td>
<td>DP - 4 × 1</td>
<td>3.66 ×</td>
<td>1.02 ×</td>
<td>3.71 ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Cluster A, C for Azure ML and B, D for SageMaker.

\[ R' = \sum_{i=1}^{k} (f_{p_i} + b_{p_i}) + 2 \sum_{i=2}^{k_0} c_{p_i} + (M - 1)(t'_{k_0} + r'_1) \]

Now let \( B_1^{(i)} \) and \( B_2^{(j)} \) be the two smallest bounds such that \( t'_{k_0} \leq B_1^{(i)} \) and \( r'_1 \leq B_2^{(j)} \). Suppose the solution found from dynamic programming for bounds \( B_1^{(i)} \) and \( B_2^{(j)} \) was \( p_1, \ldots, p_k \). The runtime of this solution is at most

\[ R = \sum_{i=1}^{k} (f_{p_i} + b_{p_i}) + 2 \sum_{i=2}^{k} c_{p_i} + (M - 1)(B_1^{(i)} + B_2^{(j)}) \]

We can bound the runtime of this solution as follows:

\[
R - R' = 2 \sum_{i=2}^{k} c_{p_i} - 2 \sum_{i=2}^{k_0} c_{p_i} \\
+ (M - 1)(B_1^{(i)} + B_2^{(j)}) - (M - 1)(t'_{k_0} + r'_1)
\]

We know that \( 2 \sum_{i=2}^{k} c_{p_i} \leq 2 \sum_{i=2}^{k_0} c_{p_i} \) since the algorithm picks the partition with minimal communication time. Furthermore, \( B_1^{(i)} - t'_{k_0} \leq \frac{B_1^{(i)} - B_1^{(1)}}{V} \) and \( B_2^{(j)} - r'_1 \leq \frac{B_2^{(j)} - B_2^{(1)}}{V} \). From this it follows that \( R - R' \leq 2(M - 1) \frac{\Delta}{V} \), where \( \Delta = \max(B_1^{(V)} - B_1^{(1)}, B_2^{(V)} - B_2^{(1)}) \).

Therefore, we can choose \( V \geq \frac{2(M - 1)\Delta}{\epsilon} \) such that the partition selected by the partitioning algorithm is within \( \epsilon \) of the optimal partition and the theorem follows. Even with this theoretical guarantee, we found that simply selecting \( V = 1000 \) works well for the models we benchmarked.

### B Supplementary Material for Evaluation

#### B.1 Cluster Setup

Table 6 outlines the details of the clusters used for the experiments. Each cluster has either 16 × K80 GPUs, or 4 × of the relatively more powerful V100 GPUs. Since there were no VMs on AWS with a single V100 GPU, we used the p3.8xlarge instance, which contains 4 × V100 GPUs.

#### B.2 Results on V100 GPUs

Table 5 summarizes performance comparison of Hydrozoa to Azure ML and SageMaker on the V100 clusters. Since V100 has more compute power than K80 GPUs, we evaluate on the computationally-intensive VGG19 and BERT-12 models. As in the K80 experiments, Hydrozoa achieves higher throughput-per-dollar than Azure ML and SageMaker on both models. Since the AWS p3.8xlarge VM contains 4 GPUs locally, it is able to leverage NVLink for extremely high throughput GPU-to-GPU communication. Consequently, SageMaker was able to outperform Azure ML.