Topic: Distributed and Federated learning

PipeDream: Generalized Pipeline Parallelism for DNN Training & A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters

Presenters: Yabin Dong, Haofeng Zhu, Hanchi Zhang

PipeDream: Generalized Pipeline Parallelism for DNN Training

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, Matei Zaharia

Microsoft Research, Carnegie Mellon University, Stanford University

SOSP 2019 - Symposium on Operating Systems Principles

Overview

- Introduction
  - Data parallelism
  - Model parallelism
  - Pipelining (GPipe)
- PipeDream
  - Principles
  - Challenges
- Implementation and Evaluation

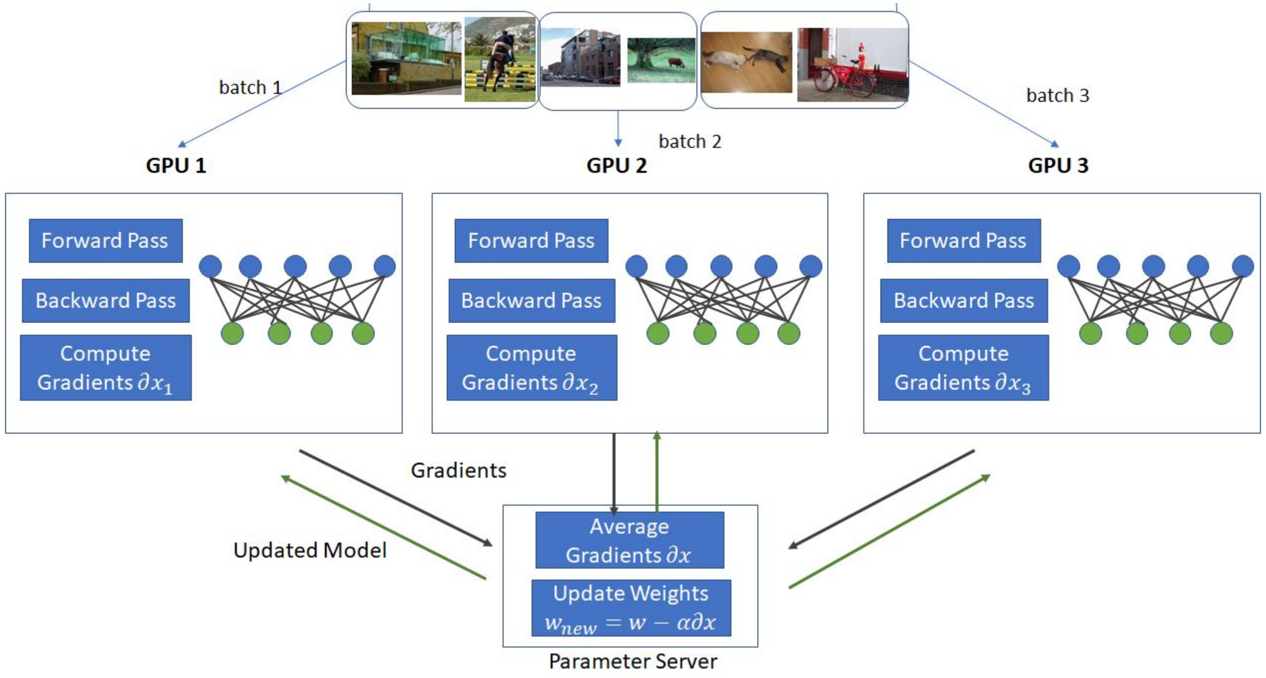
Distributed Training Approaches

- Intra-batch
  - Data parallelism (Data Partition)
  - Model parallelism (Model Partition)
  - Hybrid (e.g., For AlexNet, data partition in the convolutional layers)

- Inter-batch
  - Pipelining (GPipe)

- PipeDream: integrated intra-batch and inter-batch

Data parallelism



- Each worker will have the whole model but part of data

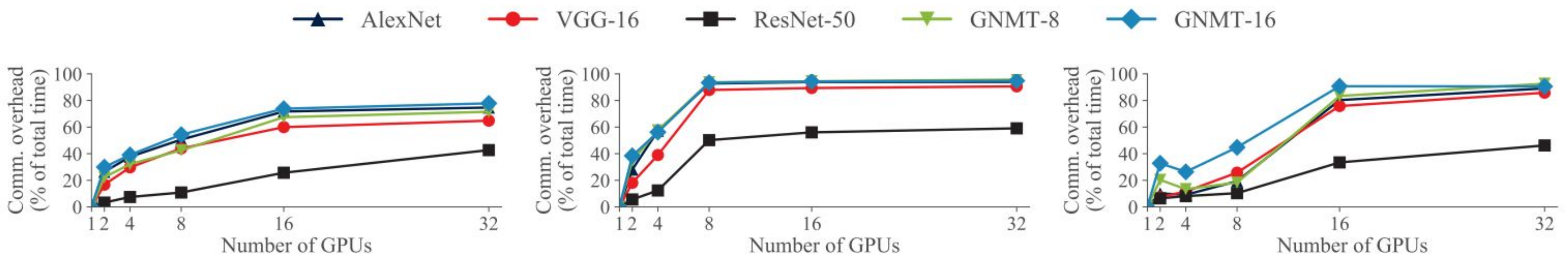
- Client-worker architecture (Parameter server)
- peer-to-peer architecture (ring all reduce algorithm)

The figure is from Telesens blog (https://www.telesens.co/2017/12/25/understanding-data-parallelism-in-machine-learning/).

Data parallelism

- Significant communication overhead

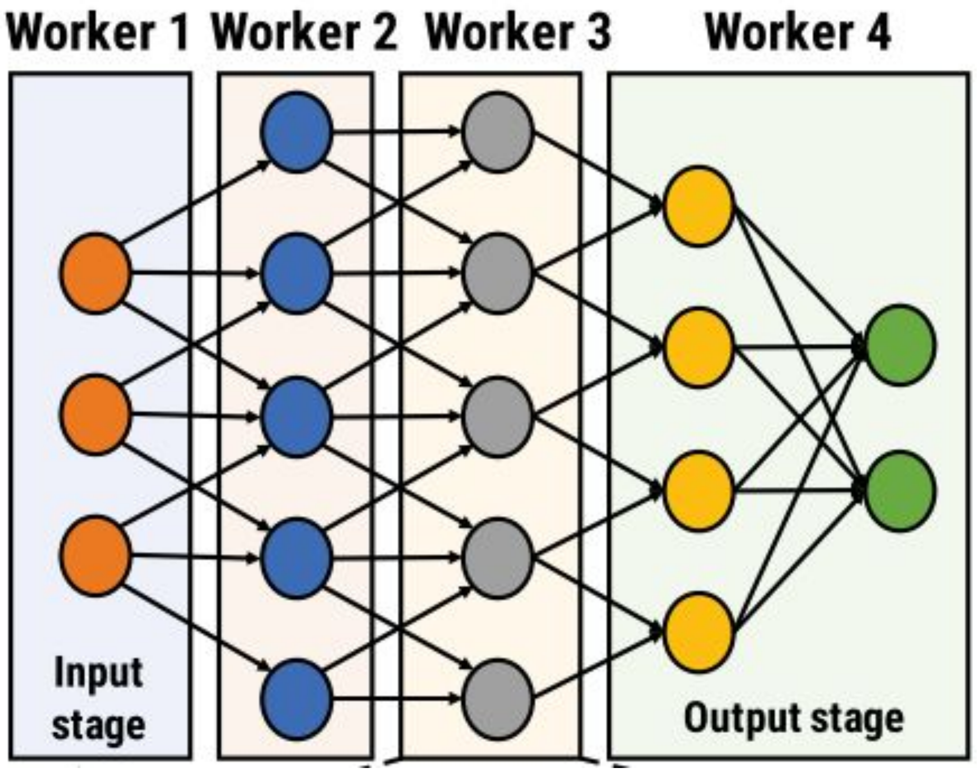
- Scales well for ResNet-50 (convolutional layers)
- Does not scale well for dense connected model

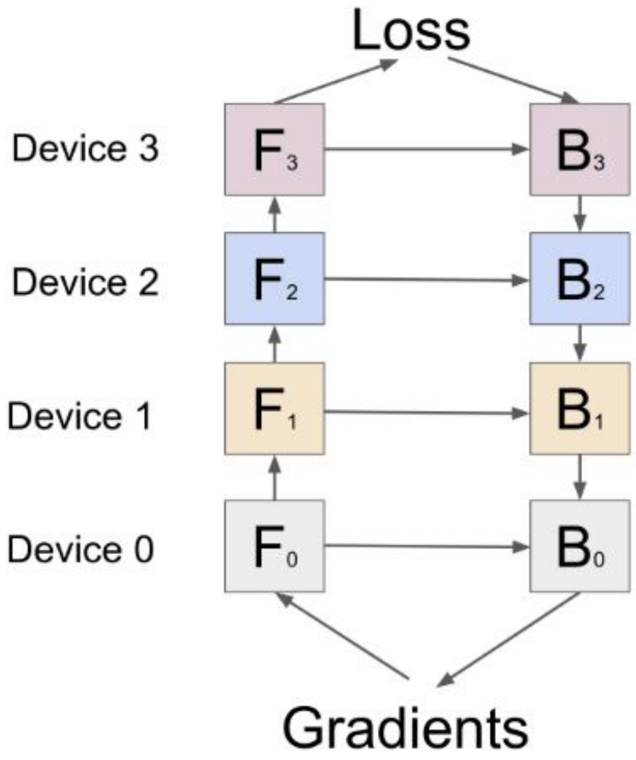


(a) Instances with 8 1080Tis (private cluster). (b) Instances with 4 V100s (Azure). (c) Instances with 8 V100s and NVLink (EC2).

Model Parallelism

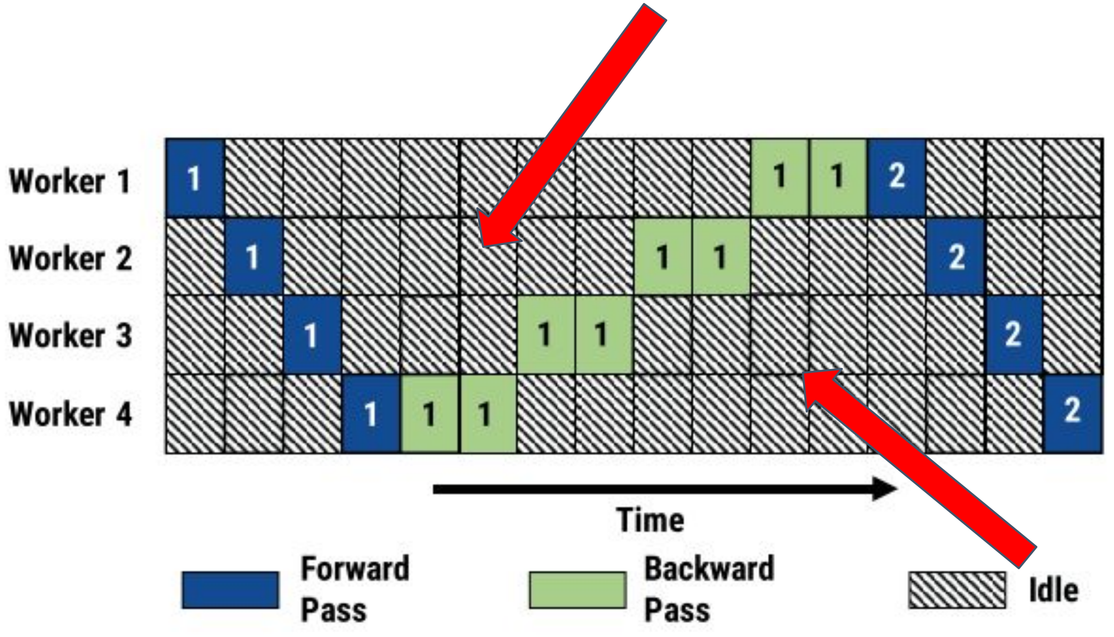
- Each worker will get the same data but have part of the model





The Right figure is from Huang et al. GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism. NeurIPS 2019.

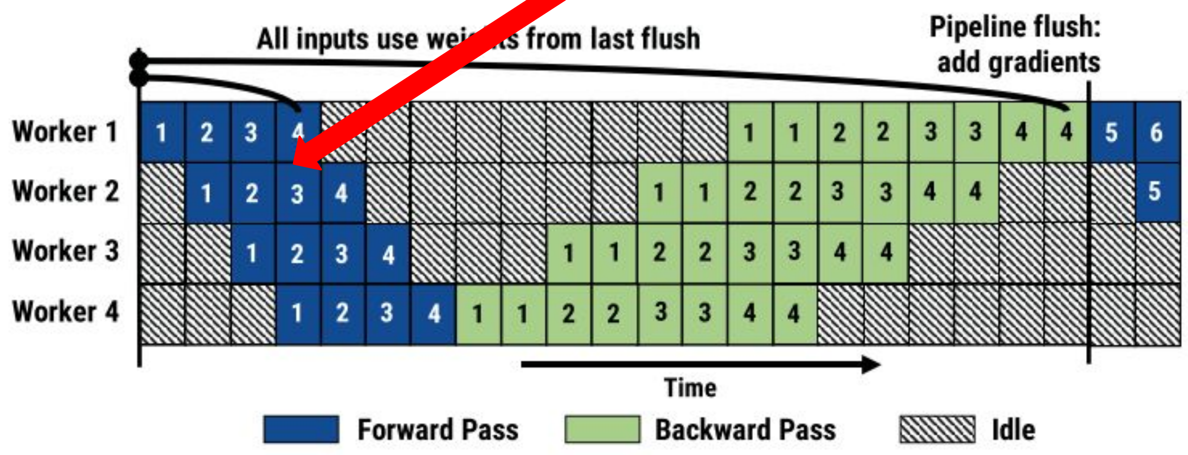
Model Parallelism

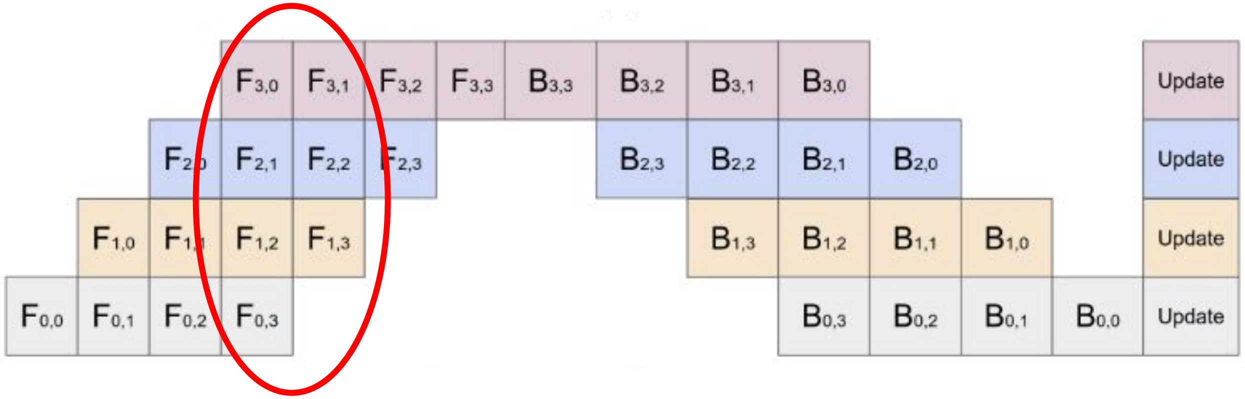


- The amount of data communicated is the size of intermediate outputs and and corresponding gradients
- Will not accelerate training
  - under-utilization of computing resources
- manually partition the mode

GPipe

Microbatch not minibatch



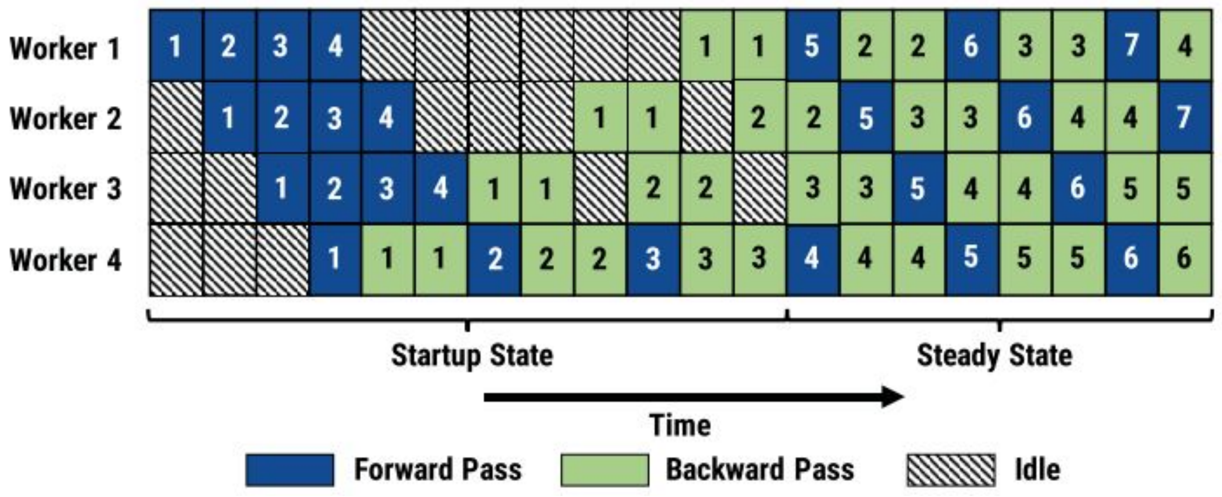


- Divide minibatch into microbatch. Pipeline microbatch across workers
- Works can be used concurrently (improves the model parallelism)

- Frequent pipeline flush reduces the hardware efficiency
- Still has relatively large idle in the system
- Manually partition model

The bottom figure is from Huang et al. GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism. NeurIPS 2019.

PipeDream



- In the startup phase: Inject multiple minibatchs to keep pipeline full (Similar to GPipe)
- In the steady state: each worker alters between forward and backwarpasses.
- Each worker once complete the forward pass for a minibatch, each stage will asynchronously sends the output to the next stage while starting to process another minibatch

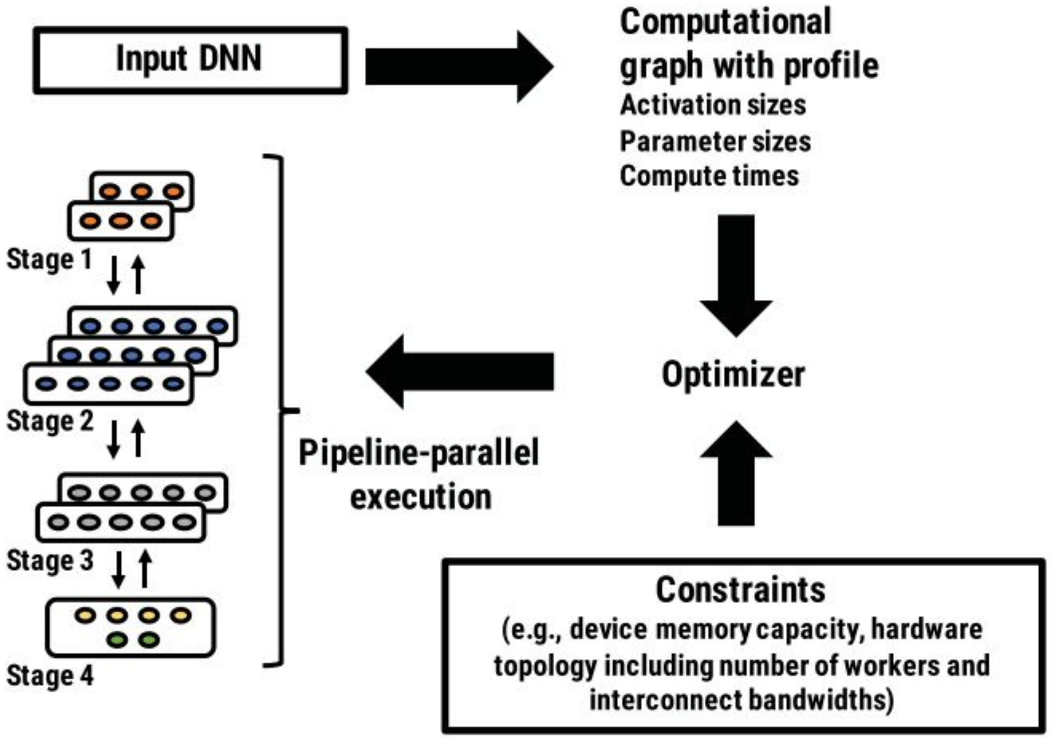
PipeDream

- No pipeline stalls in the steady state

- Pipelining communicates less. Each worker in PipeDream only need to communicate a subset of gradients and output activations, to only a single other worker. While in data parallelism, each worker need to send and receive the all parameters on the model.

- Pipelining overlaps computation and communication: Asynchronous communication of forward activations and backwards gradients across stages lead to the overlap of communication and computation

PipeDream Challenge 1: Work Partitioning



- PipeDream uses partitioning algorithm to automatically split the model
- Profiler:
  - Using a short (few minutes) profiling run 1000 minibatches on a single GPU
  - Record total computation time, the size of output activation (and input gradients) in bytes, and the size of weight parameters in bytes for each layer
  - Communication time is related to number of workers and parameter size in each layer

PipeDream Challenge 1: Work Partitioning

- Partitioning algorithm: (1) partition layers into stages, (2) the number of workers in each stage, and (3) optimal number of in-flight minibatches to full the pipeline
- The high level intuition is that the PipeDream's optimizer solves the dynamic programing problem and finds the optimal partitioning within a server and then split a model across servers
- $T^k(i \to j, m)$ is the total time taken by as single stage (layer i to j) replicated over m workers; $A^k(i \to j, m)$ is the slowest stage (layer i to j) in the optimal pipeline

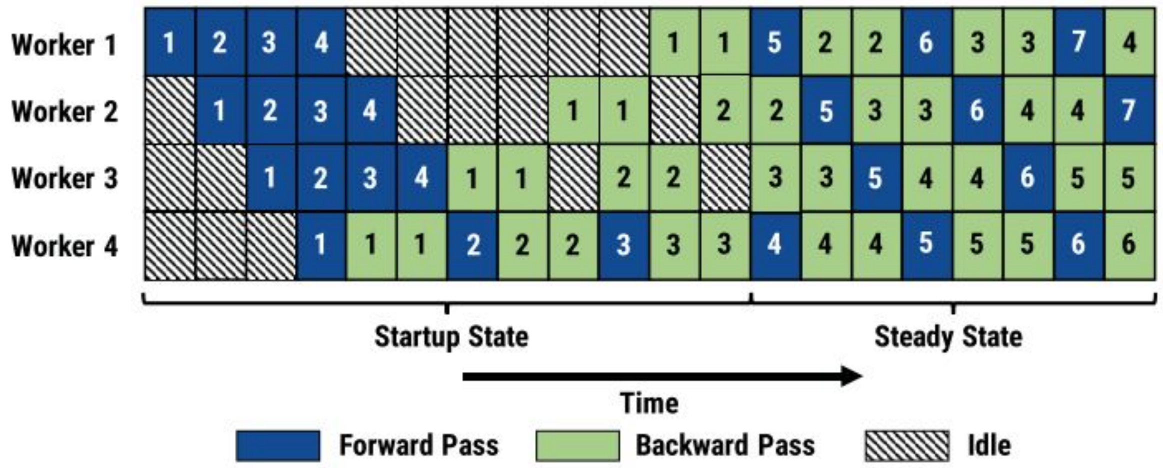
$$T^k(i \to j, m) = \frac{1}{m} \max \begin{cases} A^{k-1}(i \to j, m_{k-1}) \\ \dfrac{2(m-1)\sum_{l=i}^{j} |w_l|}{B_k} \end{cases}$$

$$A^k(i \to j, m) = \min_{i \le s < j} \min_{1 \le m' < m} \max \begin{cases} A^k(i \to s, m - m') \\ 2a_s / B_k \\ T^k(s+1 \to j, m') \end{cases}$$

PipeDream Challenge 2: Work Scheduling

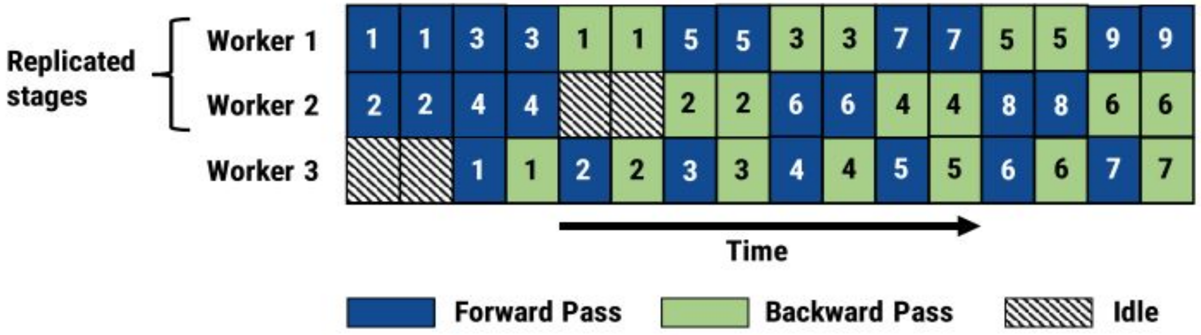
- To determine whether the worker should compute the forward or backward.

- In the startup phase: need to decide how many minibatch to inject to keep the pipeline full in the steady stage. [#workers / #replicas in the input stage]
- In the steady phase: one-forward-one-backward (1F1B) schedule. To ensure each stage to produce the output at roughly the same rate



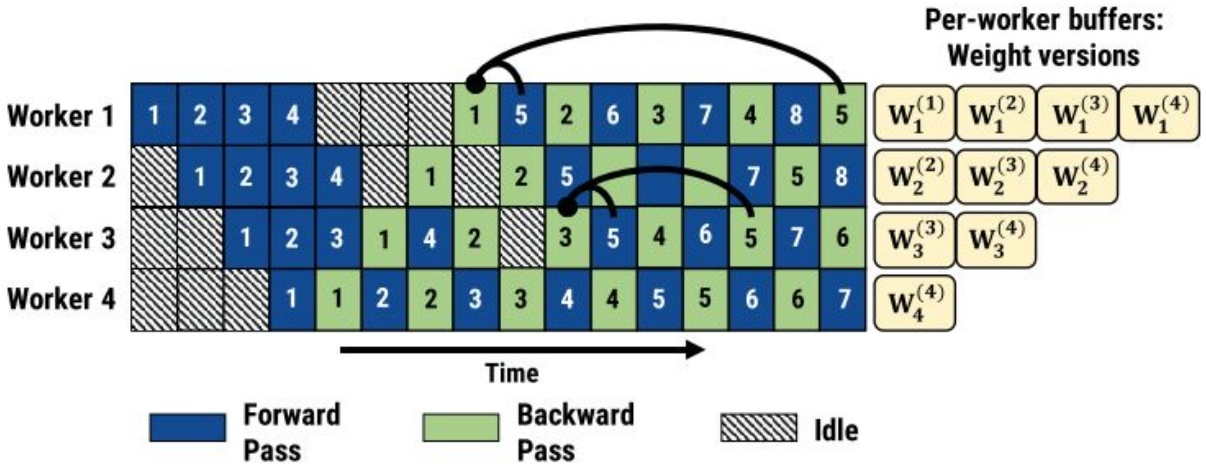
PipeDream Challenge 2: Work Scheduling

- When there is data parallelism in the stage, use the one-forward-one-backward round-robin (1F1B-RR).
- The figure below shows a 2-1 configuration. Worker 1 and worker 2 process different minibatch, while worker 3 process all minibatches.
- Worker 1 and 2 process replicated minibatches, worker 3 unreplicate.



PipeDream Challenge 3: Effective Learning

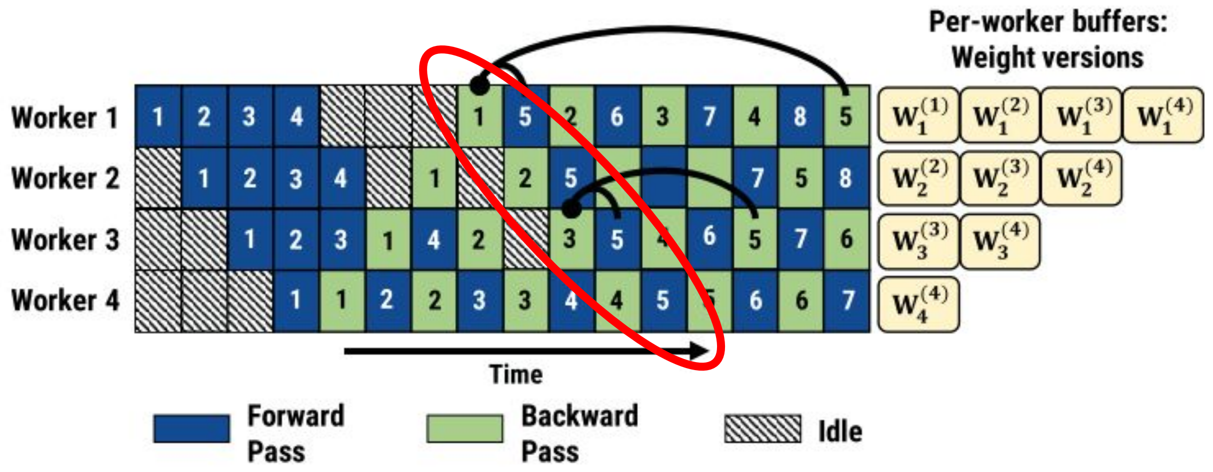
- Weight version mismatch challenge. The model parameters used for forward and backward should be the same for one minibatch
- Weight stashing: PipeDream will keep multiple weight versions one for each active minibatch.
- $$w^{(t+1)} = w^{(t)} - \nu \cdot \nabla f(w_1^{(t-n+1)}, w_2^{(t-n+2)}, \ldots, w_n^{(t)})$$



PipeDream Challenge 3: Effective Learning

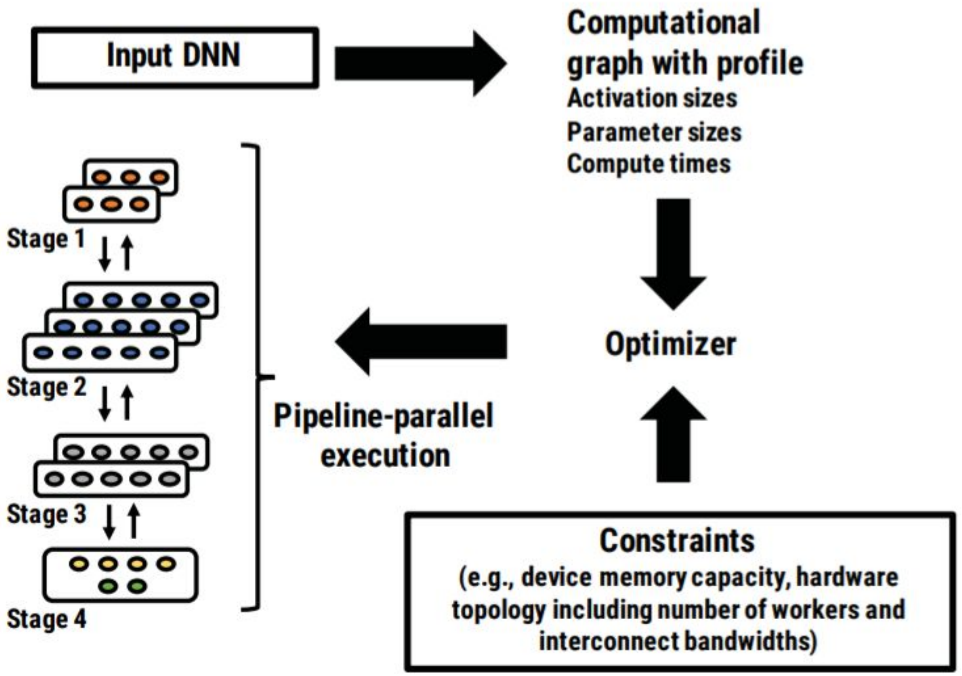
- Weight stashing does not guarantee the consistency of parameter versions for one minibatch across stages (Vertical Sync is an optional technique for this)

- Weight stashing does not significantly increase the per-worker memory

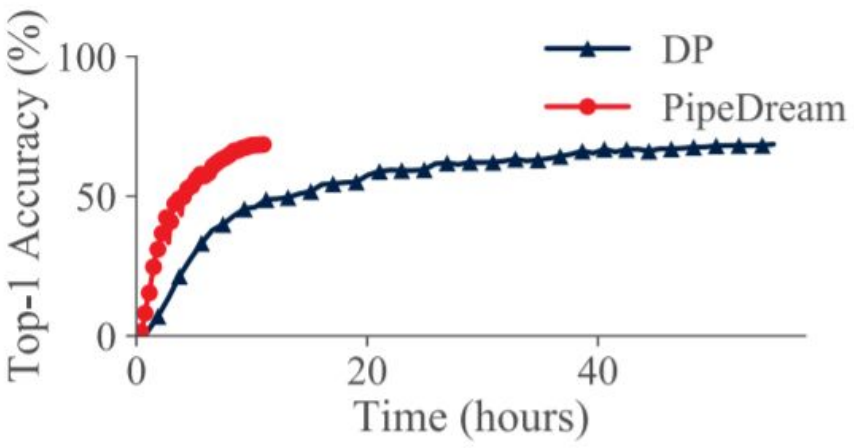


PipeDream Implementation

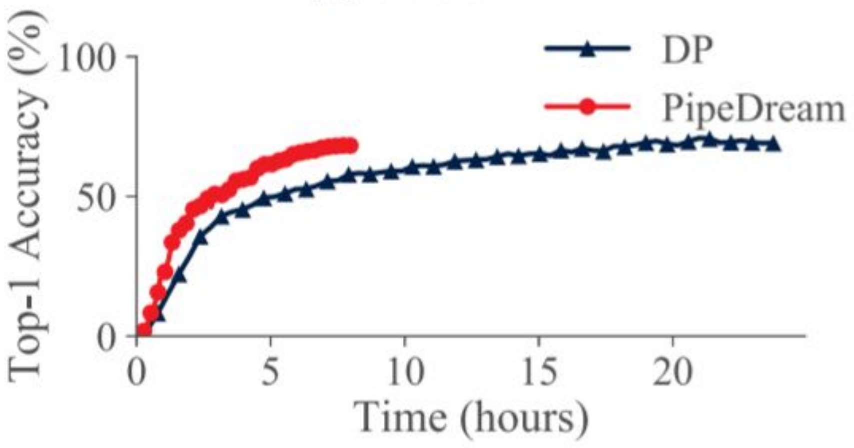
- A standalone Python library
- Uses PyTorch but can be integrated with other frameworks
- Some highlights:
    - Parameter State
    - Intermediate State
    - Stage Replication
    - Checkpointing



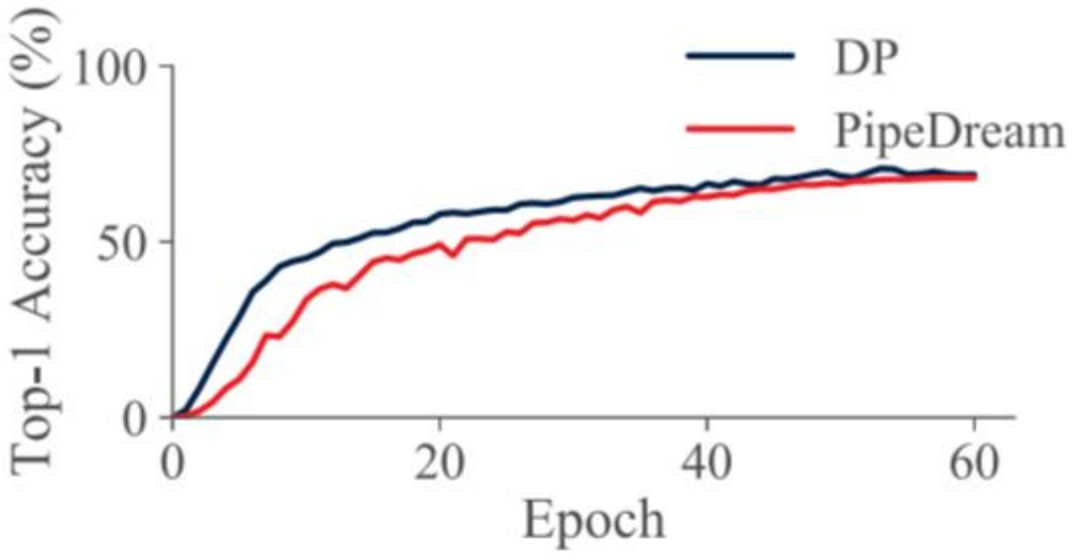
PipeDream Evaluation: Image Classification



(a) Cluster-A.



(b) Cluster-B.



PipeDream Evaluation: Other Tasks

Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP	
						Epoch time	TTA
Translation	GNMT-16 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.46×	2.2×
				4x4 (A)	Straight	2.34×	2.92×
				2x8 (B)	Straight	3.14×	3.14×
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

PipeDream Evaluation: Comparison To GPipe

- Pipeline depth equal to PipeDream optimal:
    - 55% throughput slowdown on Cluster-A
    - 71% slowdown on Cluster-B
- Pipeline depth to max:
    - 35% slowdown on Cluster-A
    - 42% slowdown on Cluster-B
- Due to more frequent pipeline flushes

Questions and Break

A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters

Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, Chuanxiong Guo

Tsinghua University, ByteDance, Google

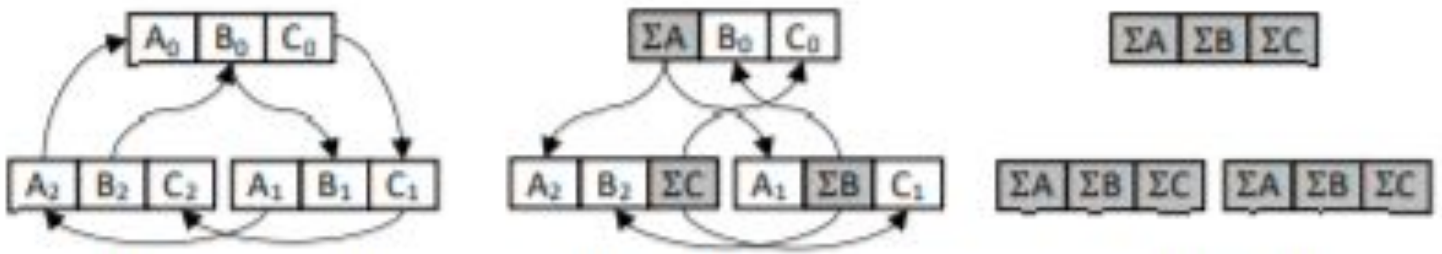
OSDI'20 - the 14th USENIX Symposium on Operating Systems Design and Implementation

Overview

- Recap
  - All-reduce
  - Parameter Server
- BytePS
  - Inter-machine communication
  - Intra-machine communication
- Implementation and Evaluation

Recap

All-reduce

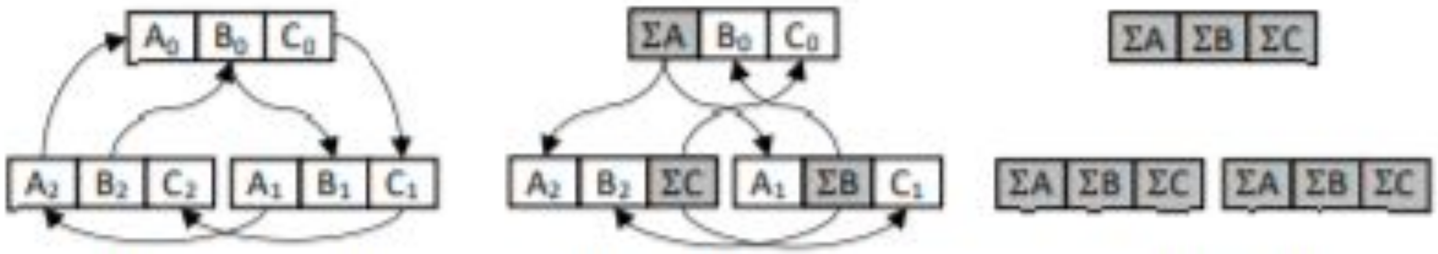


(a) Reduce-scatter (b) All-gather (c) Result

Figure 1: The communication workflow of all-reduce.

- utilize a ring-based structure to enable parallel communication
- each node serves as a starter of a ring and an end of another ring

Analysis



(a) Reduce-scatter (b) All-gather (c) Result

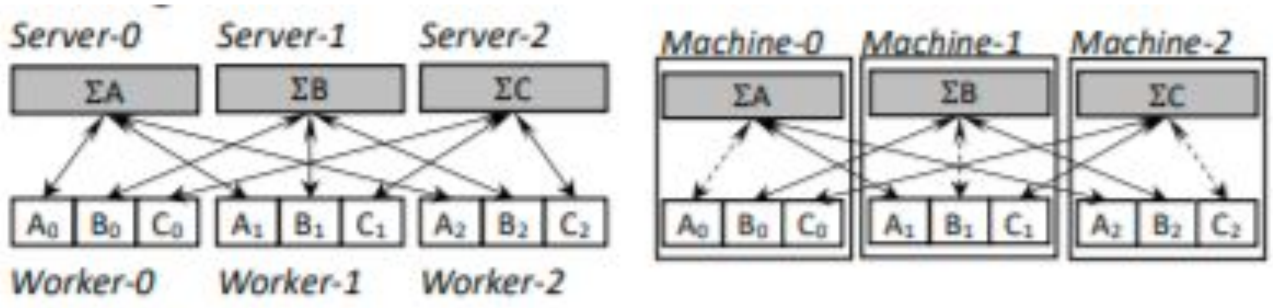
Figure 1: The communication workflow of all-reduce.

- Reduce-scatter: Each node sends(and receives) $\frac{(n-1)M}{n}$ bytes to(and from) other nodes
- All-gather: Each node sends(and receives) $\frac{(n-1)M}{n}$ bytes to(and from) other nodes
- Total traffic: $\frac{2(n-1)M}{n}$ bytes. Total communication time: $\frac{2(n-1)M}{nB}$

Notation: M: model size n: number of nodes B: network bandwidth

Continue recapping...

Parameter Server(PS)

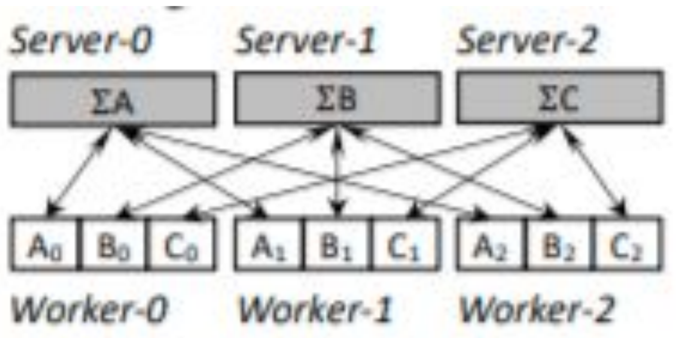


(a) Non-colocated mode (b) Colocated mode

Figure 2: The communication pattern of PS. A solid arrow line indicates the network traffic. A dashed arrow line represents the loop-back (local) traffic.

- consists of two types of nodes: servers and workers
- workers running on GPU machines push gradients to servers
- servers aggregate gradients and update the parameters
- workers pull the latest parameters from servers for next iteration

Analysis

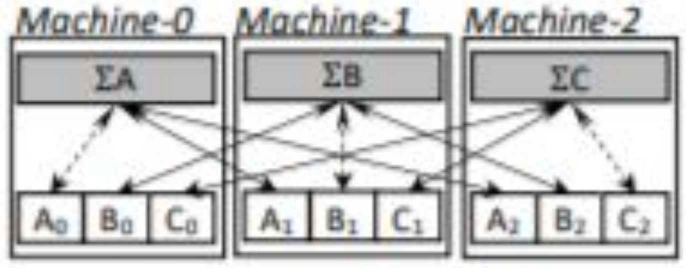


(a) Non-colocated mode

(a) Non-colocated mode
- server processes deployed on dedicated CPU machines
- model partitioned into k parts on k different CPU machines
- Each GPU worker sends and receives M bytes
- Each CPU server receives and sends $\frac{nM}{k}$ bytes
- Communication time per iteration: $\max\left(\frac{M}{B}, \frac{nM}{kB}\right)$

Notation: M: model size n: number of nodes B: network bandwidth k: number of dedicated CPU machines

Analysis



(b) Colocated mode

(b) Colocated mode

- server processes deployed on GPU machines
- model partitioned into k parts on k different CPU machines
- Each node sends and receives M bytes
- Total traffic: $\frac{2(n-1)M}{n}$ bytes. Total communication time: $\frac{2(n-1)M}{nB}$

PS in colocated mode has the same communication time per iteration as all-reduce!

However, both all-reduce and PS architecture have limitations...

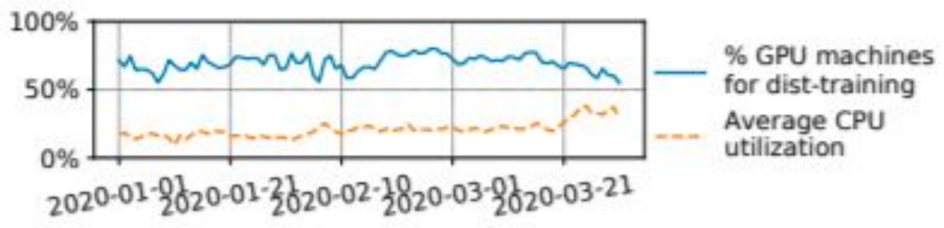


Figure 3: Daily statistics of our internal DNN training clusters from 2020-01-01 to 2020-03-31.

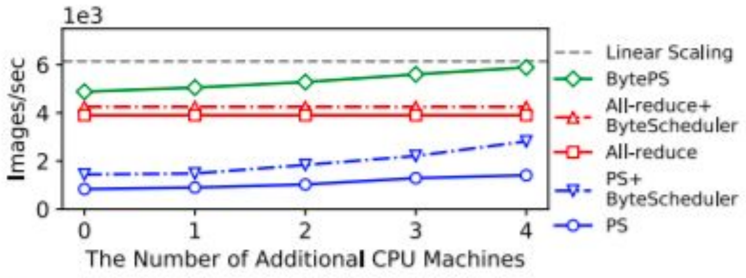


Figure 4: VGG-16 training performance of different architectures. We use 4 GPU machines with 32 GPUs in total. *Linear Scaling* represents the maximal performance (in theory) of using 32 GPUs.

- Spare CPUs and bandwidth are not fully utilized on production GPU clusters
- Existing all-reduce and PS architectures not insufficient in performance
- This is why we need BytePS architecture, which mitigates the low utilization of these spare CPUs and bandwidth.

BytePS Architecture

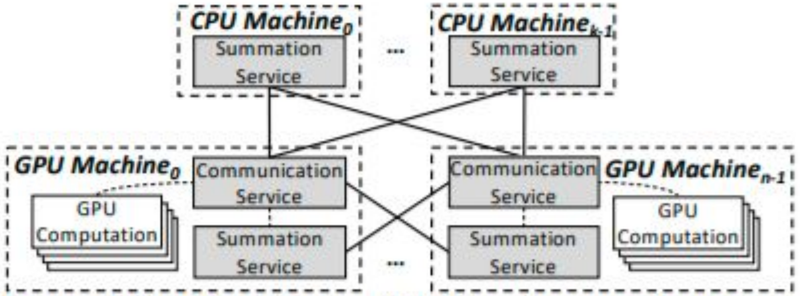


Figure 5: BytePS architecture. Solid lines: the connection between CPU machines and GPU machines. Dashed lines: the data flow inside GPU machines.

- Two main modules: Communication Service(CS) and Summation Service(SS)
- SS runs on the CPUs of both CPU and GPU machines
- CS performs GPU computation on every GPU machine

How to achieve optimal inter-machine communication for the BytePS architecture?

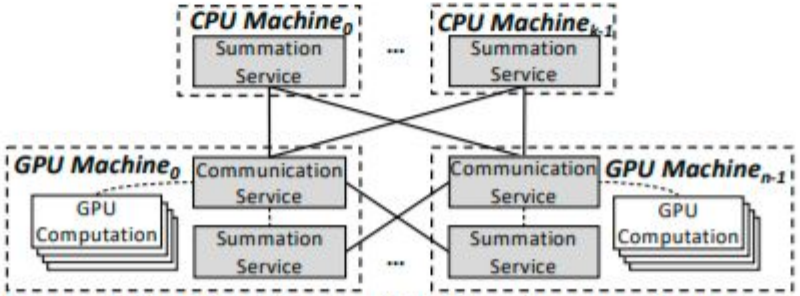


Figure 5: BytePS architecture. Solid lines: the connection between CPU machines and GPU machines. Dashed lines: the data flow inside GPU machines.

- An optimal communication requires wise SS workload assignment among CPU machines and GPU machines
- In the next slide, we use the following notation:

$M_{SS_{CPU}}$: workload of SS on CPU machines

$M_{SS_{GPU}}$: workload of SS on GPU machines

Analysis

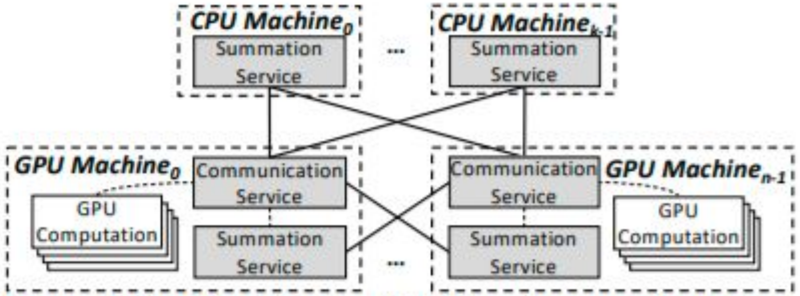


Figure 5: BytePS architecture. Solid lines: the connection between CPU machines and GPU machines. Dashed lines: the data flow inside GPU machines.

- a CS module sends and receives $M - M_{SS_{GPU}}$ bytes
- a SS module on a GPU machine receives and sends $M_{SS_{GPU}}$ bytes from other n-1 GPU machines
- a GPU machine requires communication time $t_g = \frac{M + (n-2)M_{SS_{GPU}}}{B}$
- a CPU machine requires communication time $t_c = M_{SS_{CPU}}/B$
- the sum of all SS workloads should be M

Analysis

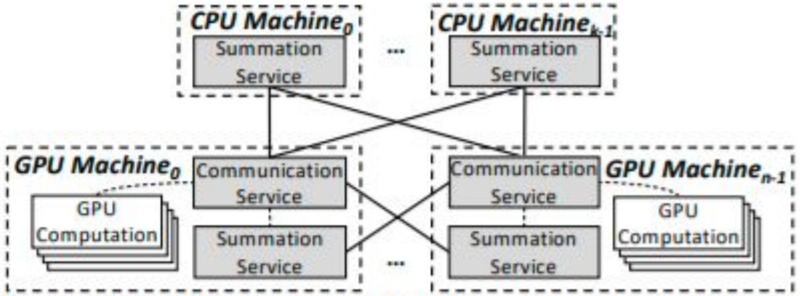


Figure 5: BytePS architecture. Solid lines: the connection between CPU machines and GPU machines. Dashed lines: the data flow inside GPU machines.

- GPU machine communication time t_g and CPU machine communication time t_c need to be equal to prevent one of them from being the bottleneck
- optimal workload assignment is given as

$$M_{SS_{CPU}} = \frac{2(n-1)}{n^2+kn-2k}M \qquad M_{SS_{GPU}} = \frac{n-k}{n^2+kn-2k}M$$

Intra-machine communication

PCIe-only topology

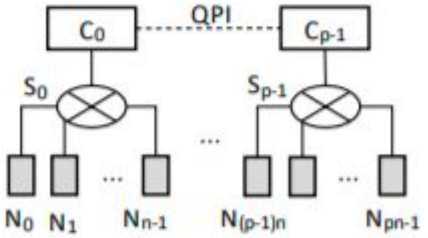


Figure 7: Notations of the PCIe-only topology.

- CPUs connected by QPI
- GPUs split into groups and connected to PCIe switches
- GPU-to-GPU memory copy across PCIe switches is more of a bottleneck

CPU-assisted aggregation(PCIe-only topology)

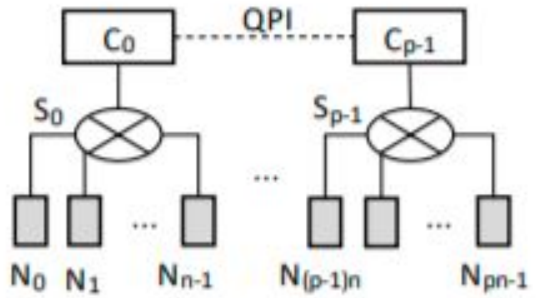
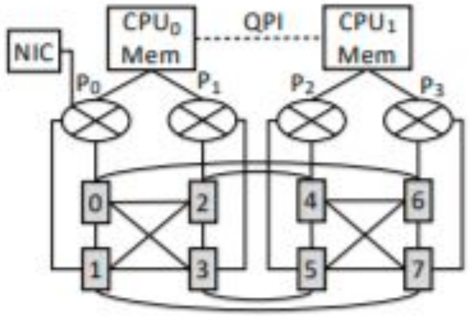


Figure 7: Notations of the PCIe-only topology.

Reduce-Scatter ⇨ GPU-CPU copy ⇨ CPU-reduce ⇨ Networking ⇨ CPU-GPU copy ⇨ All-Gather

- CPU-assisted aggregation is proved to be near-optimal in both theory and practice

NVLink-based topology



(a) NVLink-based topology

- GPUs split to different PCIe switches
- All GPUs interconnected by NVLinks
- CPU-assisted aggregation no longer needed
- Asymmetric architecture causes competition of PCIe bandwidth

BytePS's solution of Summation Service(SS)

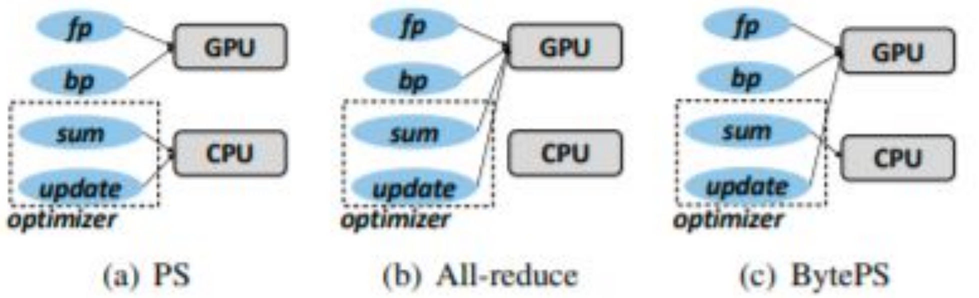


Figure 10: Component placement comparison between all-reduce, PS and BytePS.

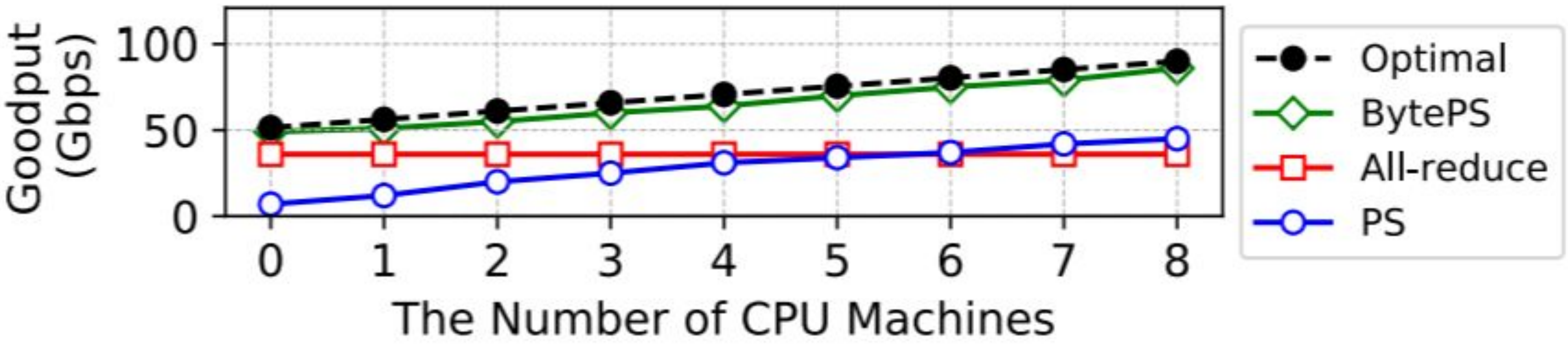
- CPU becomes a bottleneck as it cannot match increasing network bandwidth
- BytePS moves parameter update step to GPU machines

BytePS: Implementation

- Multi-stage Pipeline
  - Tensor partition and pipelining
- RDMA Performance Optimization
  - Reduces RDMA WRITE operation to one round-trip by storing remote buffer address
  - Eliminates internal loopback on GPU machines
  - Uses page-aligned memory
- Provides native Python interfaces

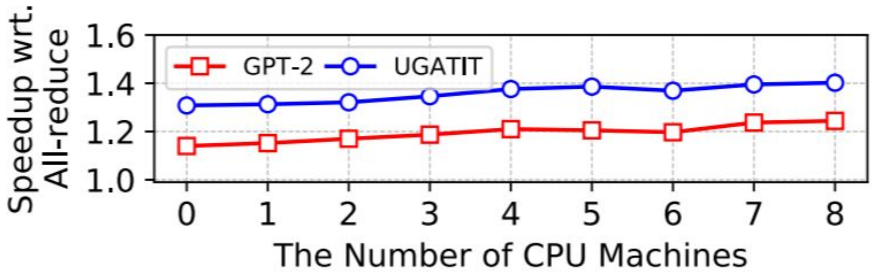
BytePS: Evaluation

- Performance is close to optimal, better than all-reduce/PS

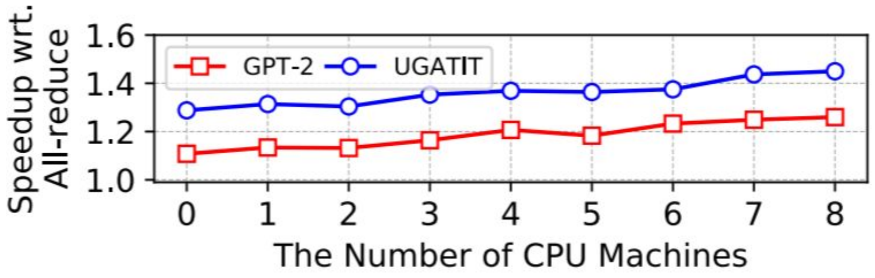


BytePS: Evaluation

- Running more CPU machines achieves up to 20% speed increase with <<10% cost, since GPU is much more expensive



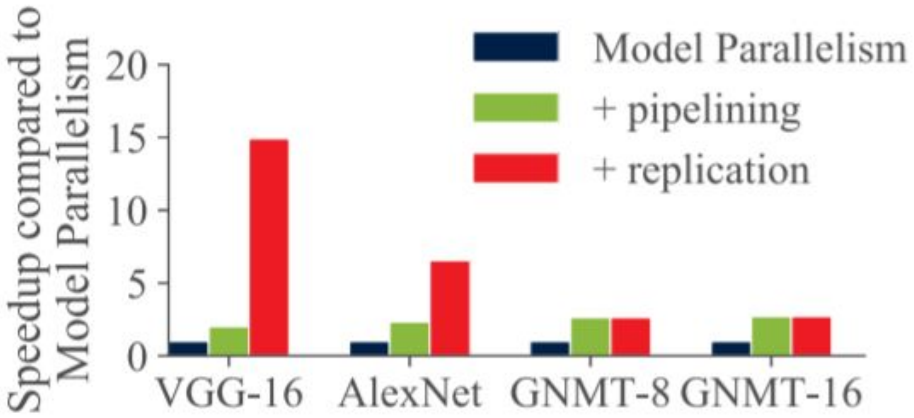
(a) PCIe-only GPU machines



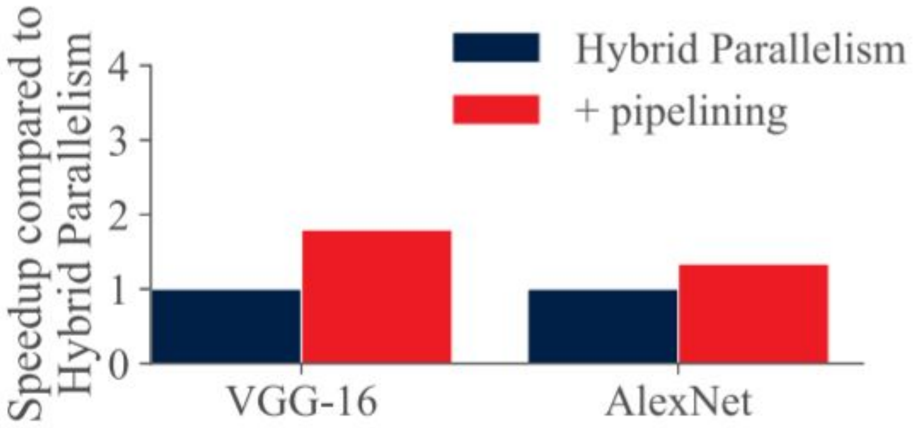
(b) NVLink-based GPU machines

Thank you!

PipeDream Evaluation: Other Intra-batch Schemes



(a) Model Parallelism.



(b) Hybrid Parallelism.

(red is Pipedream)

PipeDream Evaluation: Setup

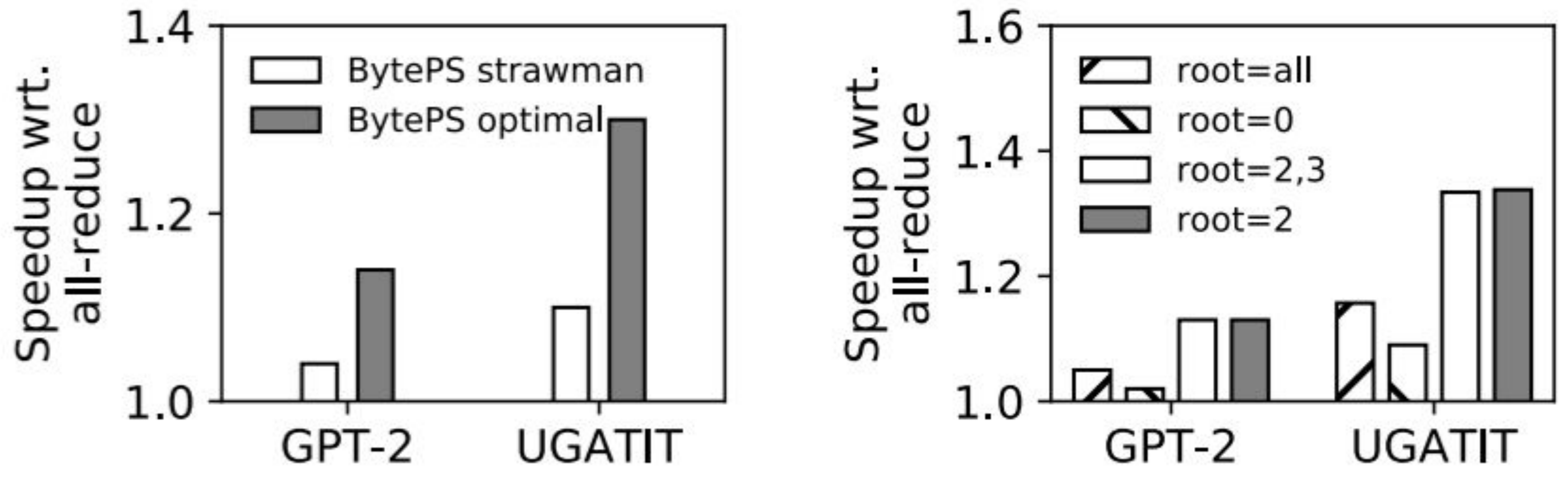
- Tasks:
  - Image Classification
  - Translation
  - Language Modeling
  - Video Captioning
- Clusters:

Cluster name	Server SKU	GPUs per server	Interconnects Intra-, Inter-server
Cluster-A	Azure NC24 v3	4x V100	PCIe, 10 Gbps
Cluster-B	AWS p3.16xlarge	8x V100	NVLink, 25 Gbps
Cluster-C	Private Cluster	1 Titan X	N/A, 40 Gbps

- Models:
  - VGG-16
  - ResNet-50
  - AlexNet
  - GNMT
  - AWD
  - S2VT
- Measure time trained to top-1 accuracy

BytePS: Evaluation

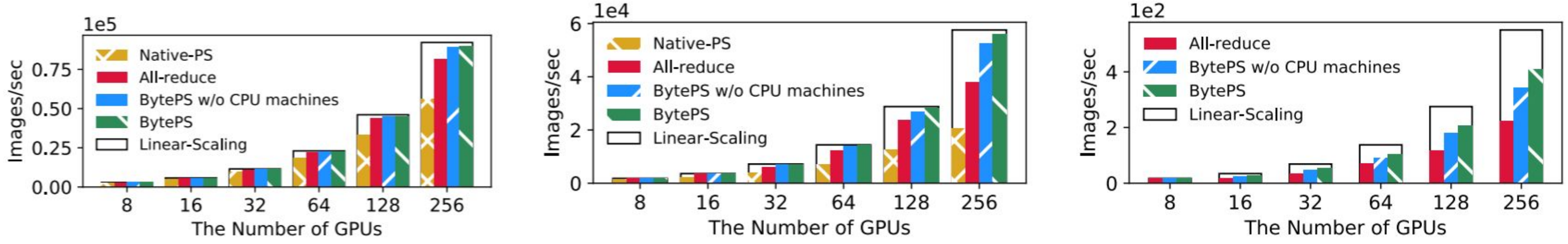
- Adaptation to intra-machine topology without optimization from CPU machines



(a) PCIe-only GPU machines (b) NVLink-based GPU machines

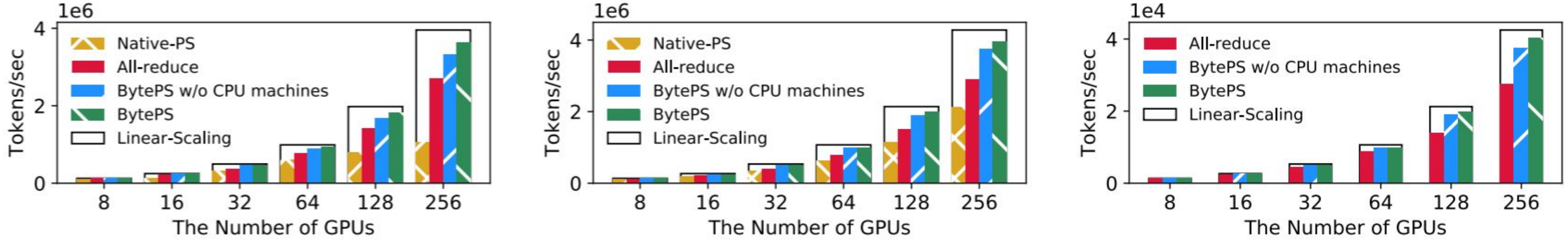
BytePS: Evaluation

- High scalability, speedup becomes larger as number of GPU machines increase



(a) TensorFlow, ResNet-50, batch=256 images (b) MXNet, VGG-16, batch=96 images (c) PyTorch, UGATIT, batch=2 images

Figure 15: Computer Vision models. The batch sizes are per GPU.



(a) TensorFlow, Transformer, batch=3072 tokens (b) MXNet, BERT-Large, batch=8192 tokens (c) PyTorch, GPT-2, batch=80 tokens

Figure 16: NLP models. The batch sizes are per GPU.