INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

ICPP/2021/CHICAGO/USA

acm In-Cooperation

sighpc

AUGUST 9-12, 2021

Prophet: Speeding up Distributed DNN Training with Predictable Communication Scheduling

Zhenwei Zhang[1], Qiang Qi[2], Ruitao Shang[3], Li Chen[†], Fei Xu[*]

Email: [1]10165102154@stu.ecnu.edu.cn [*] fxu@cs.ecnu.edu.cn

[1,2,3,]East China Normal University*

[†]University of Louisiana at Lafayette

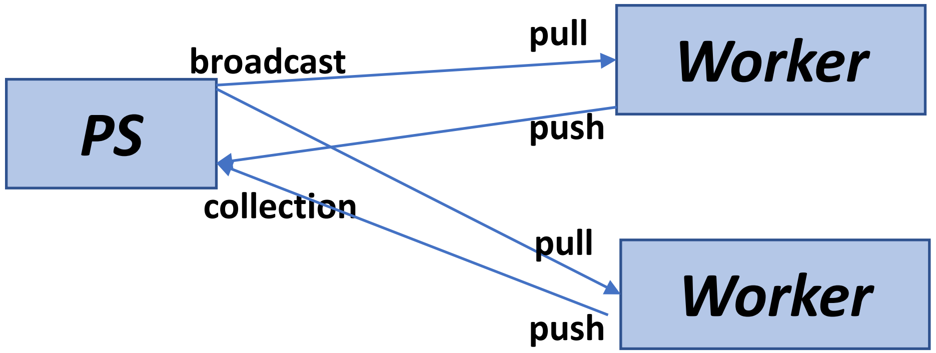
華東師範大學 EAST CHINA NORMAL UNIVERSITY

UNIVERSITY OF LOUISIANA Lafayette

Background
Data Transmission in Distributed Deep Learning

- In each training iteration, amounts of data (*e.g.*, gradients/parameters, intermediate data) should be transferred across devices.

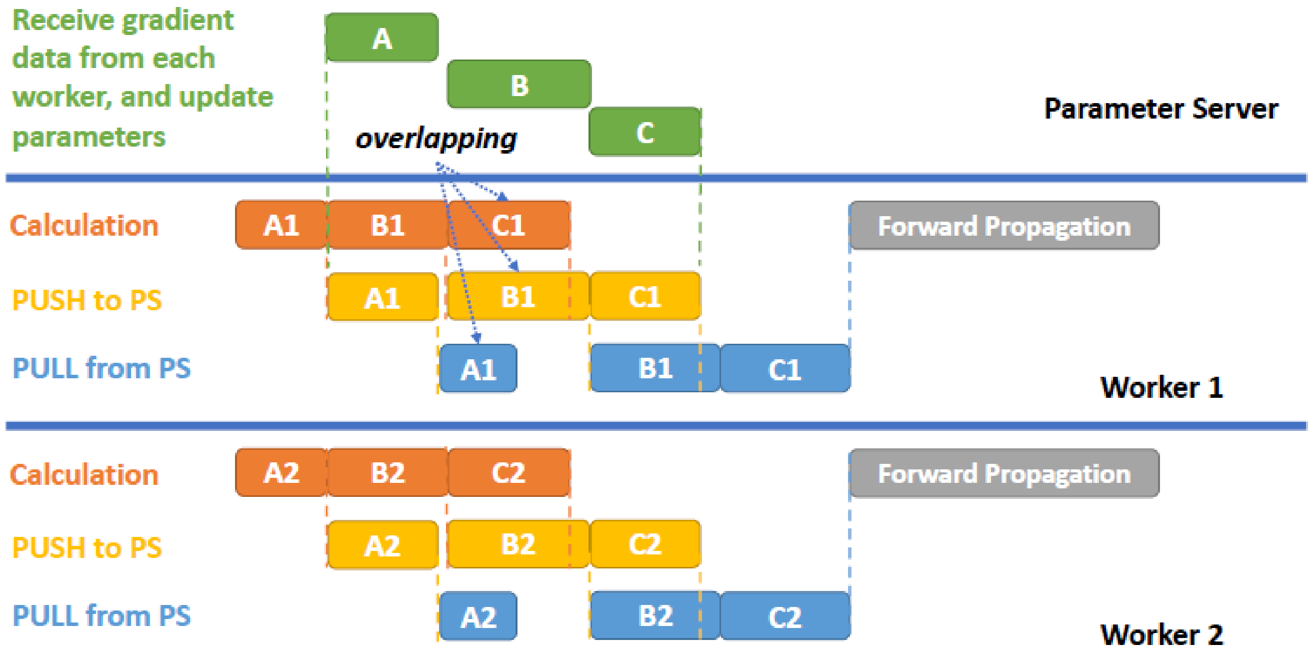
- Parameter Server, a *centralized* communication architecture is typically used for *data parallelism*.



Background
Bottleneck and Related Works

- **The communication can block the computation of *Workers*.**

- **Several related works design *priority-based* communication scheduling strategies.**
  - *e.g., P3* is dedicated to *overlapping* the backward propagation with the gradient push process.

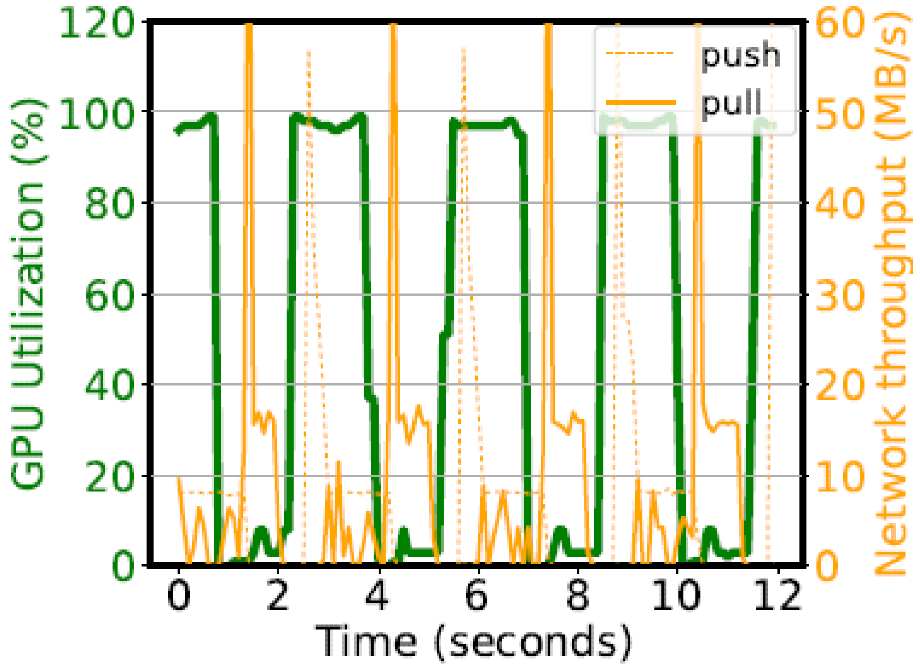


INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

acm In-Cooperation sighpc

Motivation
Existing Problems of Related Works

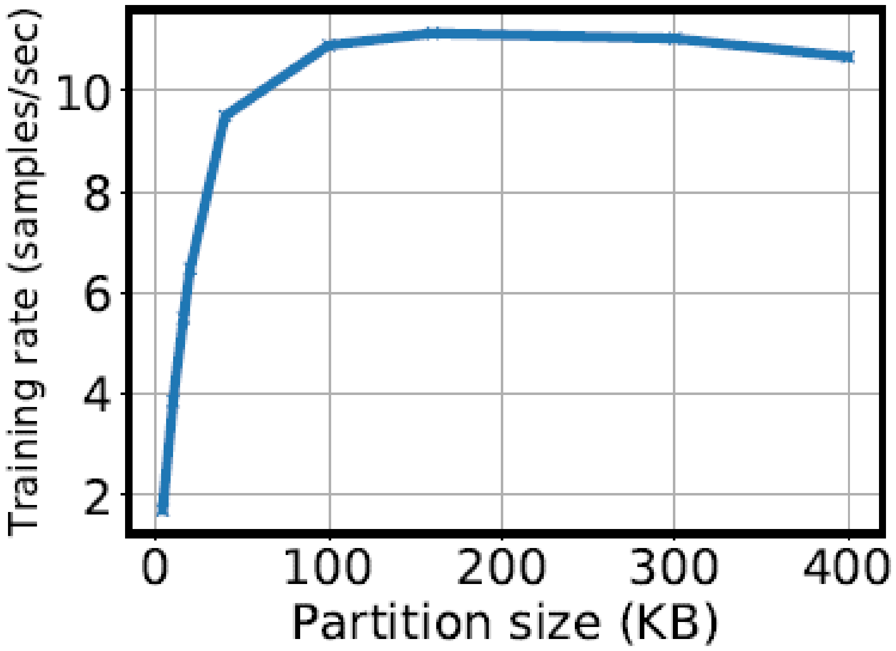
- **The GPU utilization is still low in some situations.**
  - the GPU utilization can dramatically decrease to zero (i.e., totally idle) during the `pull` operation of model parameters.



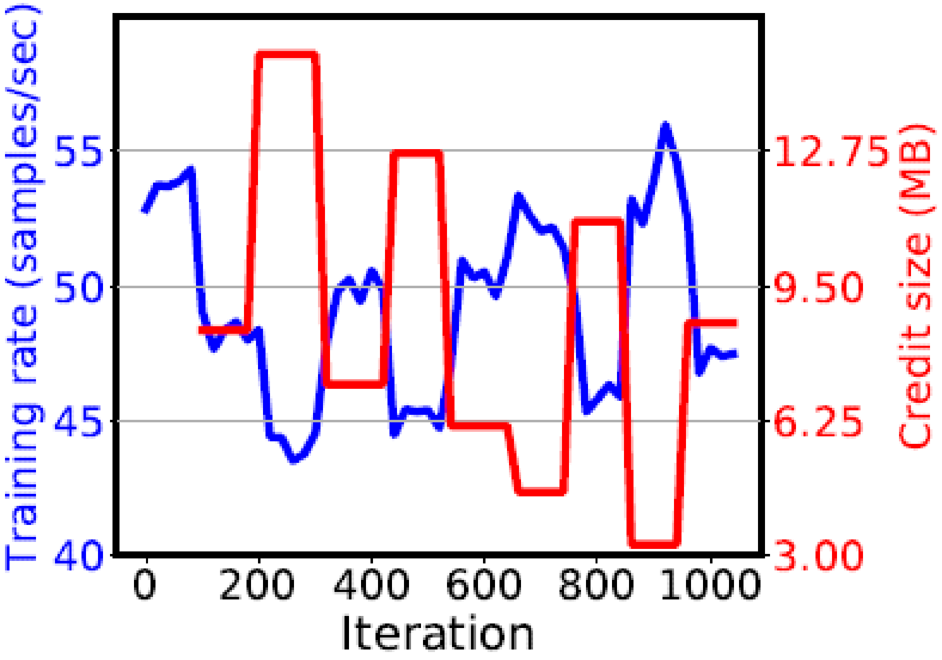
The slow network transmission makes the GPUs fail to timely acquire the model parameters and thus delays the computation (i.e., forward propagation).

Motivation
Existing Problems of Related Works

- **Non-negligible performance overhead of state-of-the-art priority-based communication scheduling strategies.**



The smaller size of partitions dramatically decreases the DDNN training rate

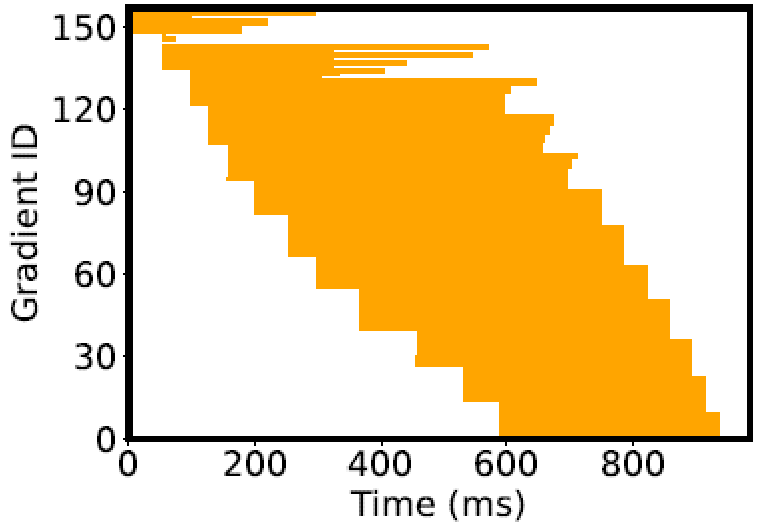


The training rate of the ResNet50 model fluctuates rapidly

Motivation
Stepwise Pattern

- **Communication characteristic of gradient transfer follows a stepwise pattern over time.**

  - The gradient data requires aggregation before transmission, which can be considered as the main cause of stepwise pattern.
  - Such a pattern is independent of the DDNN training frameworks, DNN models, datasets, and hardware architectures.



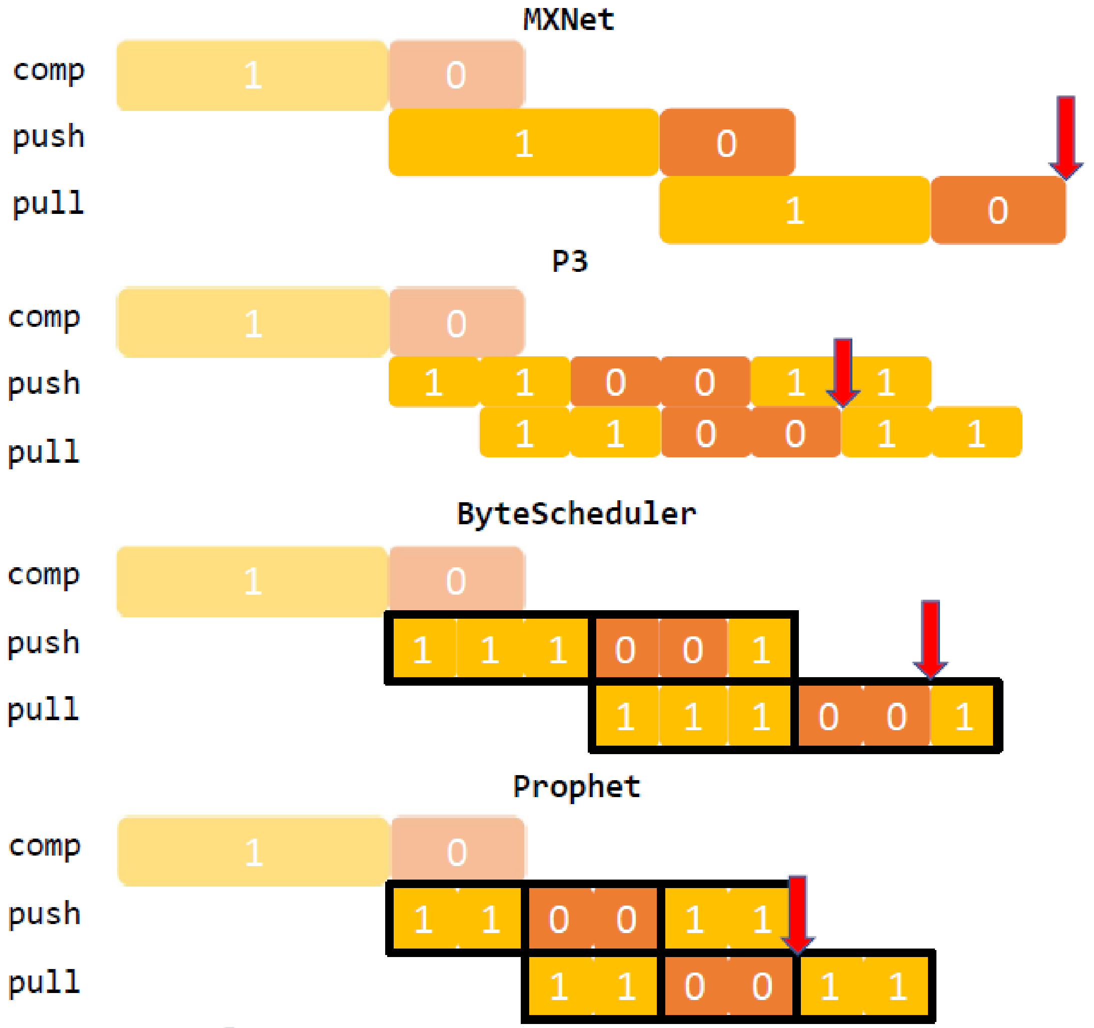
Motivation
Prophet

- **The *core idea* of *Prophet* is that: It predicts the transferred gradient data size by profiling the time interval between *blocks* and the available network bandwidth during model training.**

  - *Prophet* ensures that each gradient can be transferred by greedily utilizing the network bandwidth resources without blocking the higher-priority gradients, so that **the critical gradients** (*e.g.,* gradient 0) can be transferred as fast as possible, and with **negligible runtime overhead**.

Motivation
An Illustrative Example

- Default *MXNet* transmits gradients with the default FIFO order;
- *P3* slices the gradients into small partitions to ensure a timely preemption, but with non-negligible overhead;
- *ByteScheduler* configures the `credit size` to avoid the partition overhead, while keeping a relatively high preemption rate;
- *Prophet* introduces the concept of ***gradient blocks*** by identifying **the stepwise pattern** for DDNN training, greedily transferring the gradient data through a lightweight job profiling.



* The notation indicates the start time of forward propagation

* The notation denotes the assembled gradients that are transferred at a time

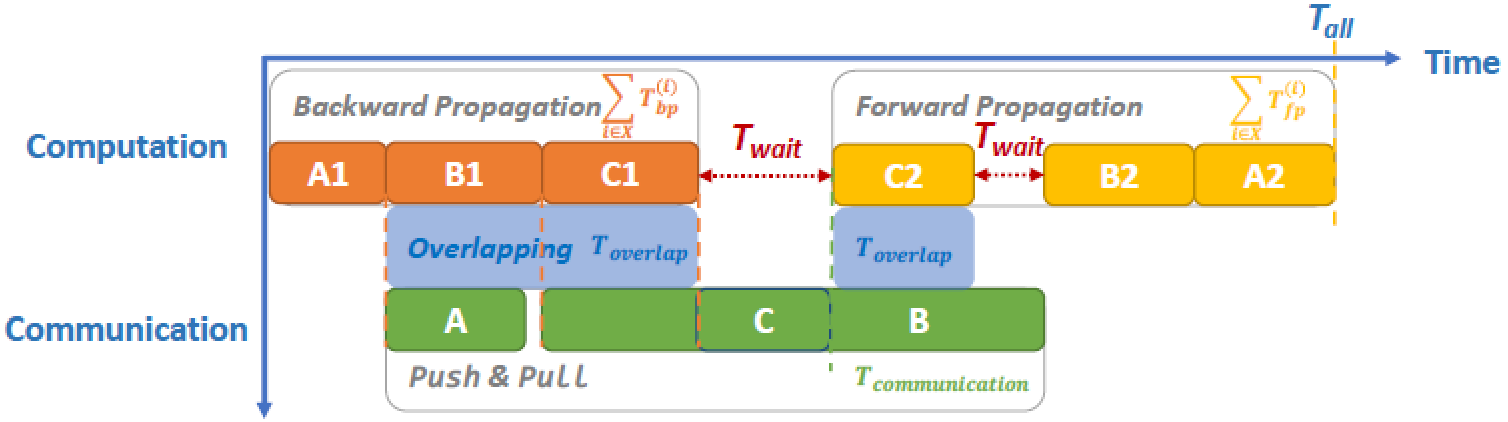
Modeling
Illustration of DDNN Training Time

$$\begin{aligned} T_{all} &= \boxed{\sum_{i\in X} T_{bp}^{(i)}} + \boxed{\sum_{i\in X} T_{fp}^{(i)}} + \boxed{T_{communication} - T_{overlap}} \\ &= \sum_{i\in X} T_{bp}^{(i)} + \sum_{i\in X} T_{fp}^{(i)} + T_{wait}, \end{aligned}$$

the total idle time of GPU resources during the entire training process

the time cost for backward propagation

the time cost for forward propagation



Modeling
Illustration of DDNN Training Time

$$T_{wait} = \boxed{\sum_{i \in X, i \neq 0} \left(\boxed{u^{(i)}} - \boxed{p^{(i-1)}}\right)^{+}} + \left(u^{(0)} - c^{(0)}\right).$$

GPUs are in the working state during the backward propagation until gradient 0 is generated at $c_{(0)}$

GPUs are back to the working state when gradient 0 updates its parameters at $u_{(0)}$

completion time of parameter update

completion time of forward propagation

part of GPU idle time

$$p^{(i)} = \begin{cases} \max\{p^{(i-1)}, u^{(i)}\} + T_{fp}^{(i)} & \text{when} \quad i \neq 0, \\ u^{(0)} + T_{fp}^{(0)} & \text{when} \quad i = 0. \end{cases}$$

* the network communication of timely transferred gradients actually overlaps with the forward propagation (i.e., $u_{(i)} < p_{(i-1)}$), and thus we only use the positive part of $u_{(i)} - p_{(i-1)}$

gradient x(i) can start its forward propagation only when the previous gradient x(i−1) finishes the forward propagation and x(i) completes its parameter update process

Modeling
Problem Formulation

- How to schedule the gradient transfer time to minimize T_{wait} (and thus to minimize T_{all}) to maximize the GPU resource utilization.

$$\min_{t^{(i)}} \quad T_{wait} = \sum_{i \in \mathcal{X}, i \neq 0} \left(u^{(i)} - p^{(i-1)}\right)^{+} + \left(u^{(0)} - c^{(0)}\right)$$

$$\text{s.t.} \quad t^{(i)} \geqslant c^{(i)},$$

can only be pushed after it is generated

$$t^{(i)} \notin [t^{(j)}, t^{(j)} + E^{(j)}), \quad \forall j \in \mathcal{X}, j \neq i,$$

avoids the concurrent gradient transfer

$$t^{(i)} > t^{(k)}, \quad \forall k \in \mathcal{X} < i, \quad \text{when} \quad t^{(i)} > c^{(0)}.$$

during the forward propagation, un-transmitted gradients should be transmitted in the order of priority

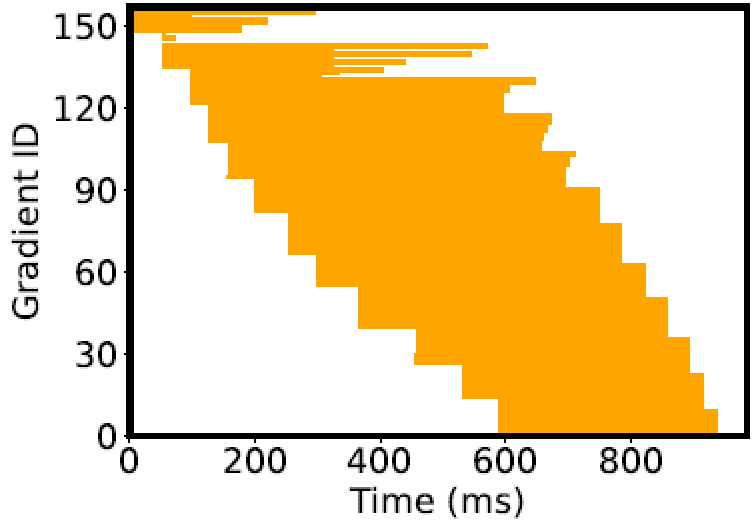
Modeling
Problem Formulation

➢ **The gradients should be transferred before any higher-priority gradient is generated in the backward propagation.**

$$t^{(i)} + E^{(i)} \leqslant c^{(k)}, \quad \forall k \in \mathcal{X} < i, \quad \text{when} \quad t^{(i)} \leqslant c^{(0)}.$$

This implies that we can take advantage of the stepwise pattern (e.g., block time interval) to find the optimal gradient transfer

** Gradients are transferred in the order of priorities in the forward propagation*



Algorithm Design Principles

- By leveraging the block time interval and the monitored network bandwidth, we determine the start time of gradient transfer during both the backward propagation and forward propagation.
  - Take advantage of our observed stepwise pattern.

Algorithm Design Pseudocode

Algorithm 1: *Prophet*: Communication scheduling strategy for improving resource utilization of workers and minimizing the DDNN training time.

Input: Current available network bandwidth B of workers, the gradient generation time $c^{(i)}$ and the gradient size $s^{(i)}$ for each gradient $i \in \mathcal{X}$.

Output: Start time $t^{(i)}$ of gradient transfer (*i.e.*, the start time to push each gradient $i \in \mathcal{X}$).

Initialize: Estimated gradient transmission time $E^{(i)} \leftarrow \frac{s^{(i)}}{B}$ (by Eq. (5) and Eq. (10)), and the expected transfer time interval $A^{(i)} \leftarrow \min |c^{(i)} - c^{(j)}|, j < i$, for each gradient $i \in \mathcal{X}$.

while exists gradients to be scheduled **do**
- $p \leftarrow$ the highest priority of gradients ready to be scheduled;
- **if** $p \neq$ `gradient 0` **then**
  - **if** the current time is in the backward propagation **then**
    - Initialize $T_{used} \leftarrow 0$, $q \leftarrow p$;
    - **while** the gradient q can be greedily transferred within the expected time interval $A^{(q)} - T_{used}$ **do**
      - Assemble the gradient q into a *gradient block* and update $t^{(q)} \leftarrow T_{used} + c^{(p)}$;
      - Update the used time $T_{used} \leftarrow T_{used} + E^{(q)}$;
      - Set q as the currently highest-priority gradient that is ready to be transferred;
    - **end while**; `// backward propagation.`
  - **else**
    - Set $t^{(p)}$ as the earliest available scheduling time t_{next};
    - Update the next available time $t_{next} \leftarrow t_{next} + E^{(p)}$;
  - **end if**; `// forward propagation.`
- **else**
  - Set $t^{(0)}$ as the generation time $c^{(0)}$ of `gradient 0`;
  - Update the next available time $t_{next} \leftarrow t^{(0)} + E^{(0)}$;
- **end if**

end while

return the start time $t^{(i)}$ of gradient transfer;

determine the start time of gradient transfer during both the backward and forward propagation

greedily assemble each gradient if it can be transmitted before any higher-priority gradient is generated

assemble the gradient into a gradient block

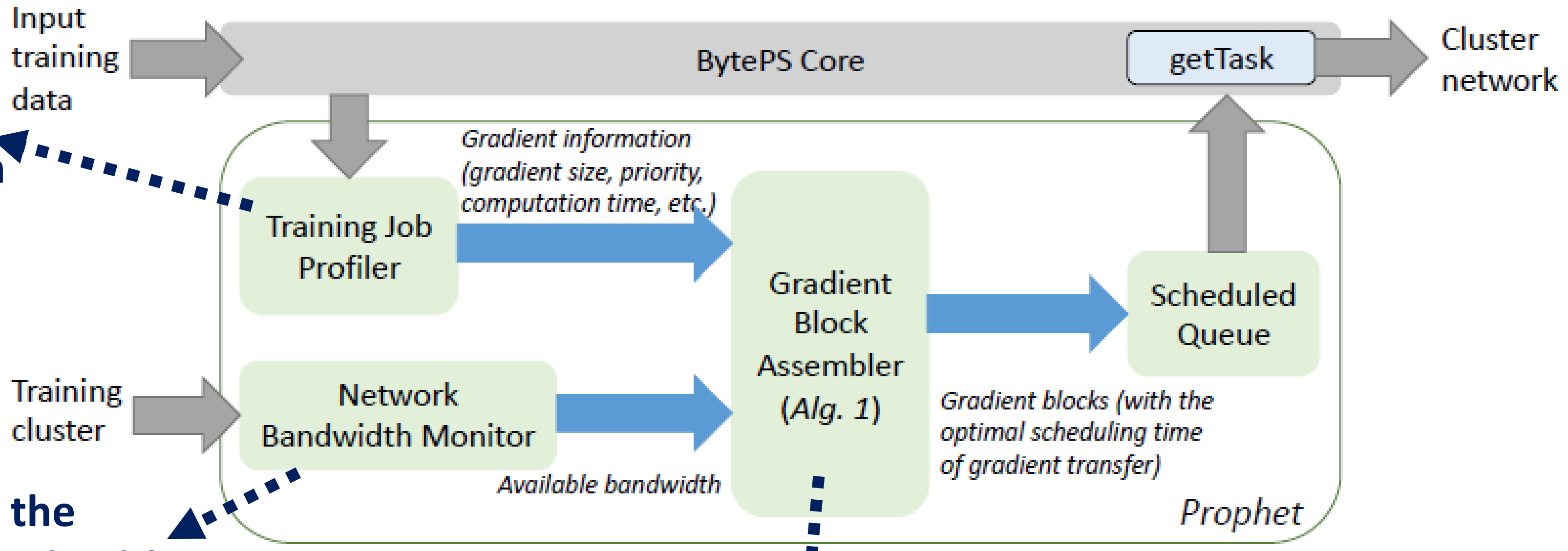
gradients are transferred in the order of priority in forward propagation

ensure critical gradients are transferred as fast as possible

Algorithm Design
Implementation of *Prophet*

- ***Prophet* is implemented based on the abstraction layer of *BytePS.***
  - designed to support multiple frameworks

pre-trains the DNN model to obtain the gradient information



periodically acquires the available network bandwidth

leverages the gradient and network bandwidth information to find the optimal start time of gradient transfer

Evaluation
Experimental Setup & Metrics

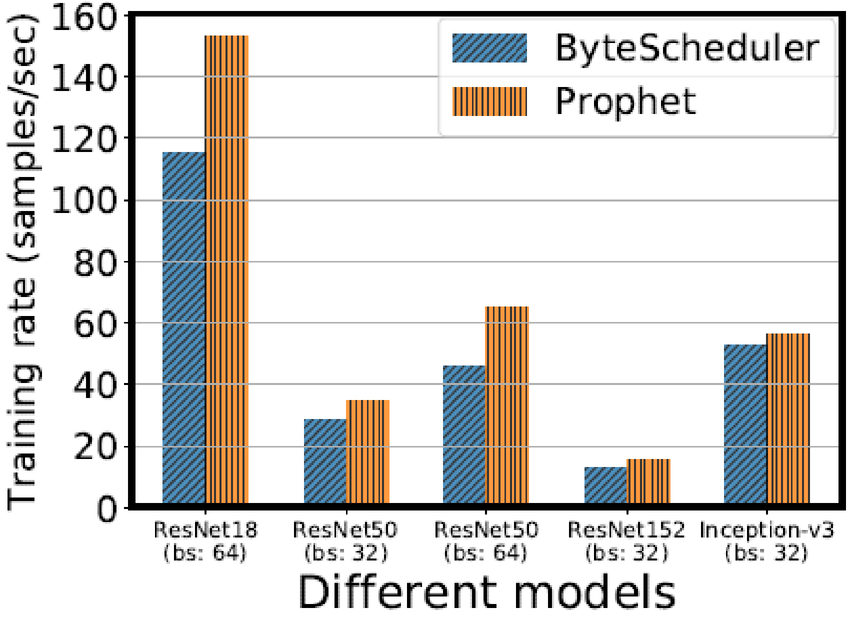
- **Eight `g3.8xlarge` instances (i.e., 1 PS and 7 worker nodes) in Amazon EC2**
  - 32 vCPUs (2.7 GHz Intel Xeon E5-2686 v4 Broadwell)
  - 2 GPUs (NVIDIA Tesla M60 GPU, each is equipped with 2048 parallel processing cores and 8 GB GPU memory)
  - 244 GB memory
  - varying network bandwidth from 1 Gbps to 10 Gbps.

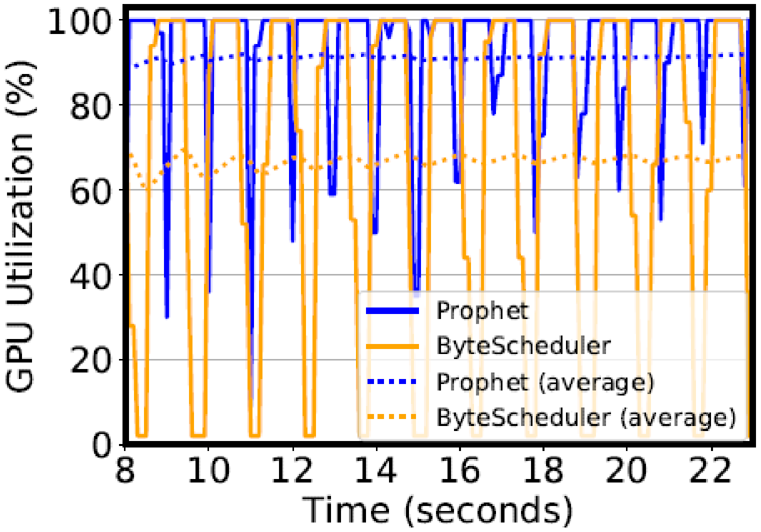
- **Four representative DNN models**
  - ResNet18
  - ResNet50
  - ResNet152
  - Inception-v3

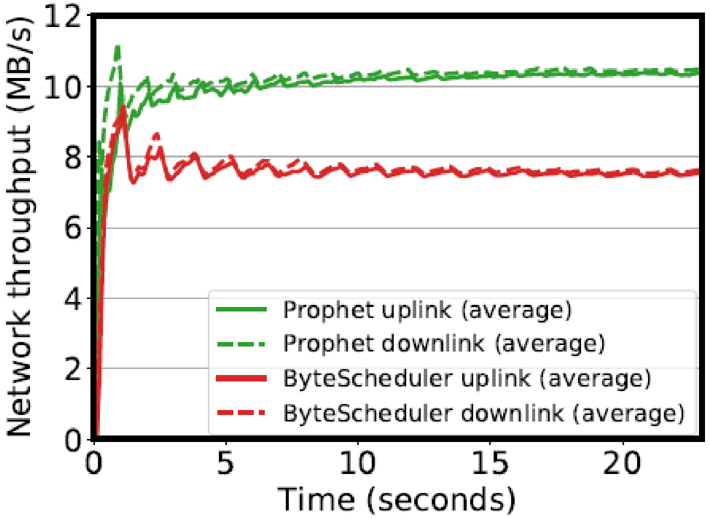
- **Three evaluation metrics**
  - **The model training rate**
  - **The GPU utilization and network uplink/downlink throughput**
  - **The wait time of each gradient data and start time of the forward propagations**

Evaluation
Effectiveness of *Prophet*

- ***Prophet*** **can significantly improve the training rate by 10% – 40% compared with** ***ByteScheduler*****, for different DNN models and batch sizes.**

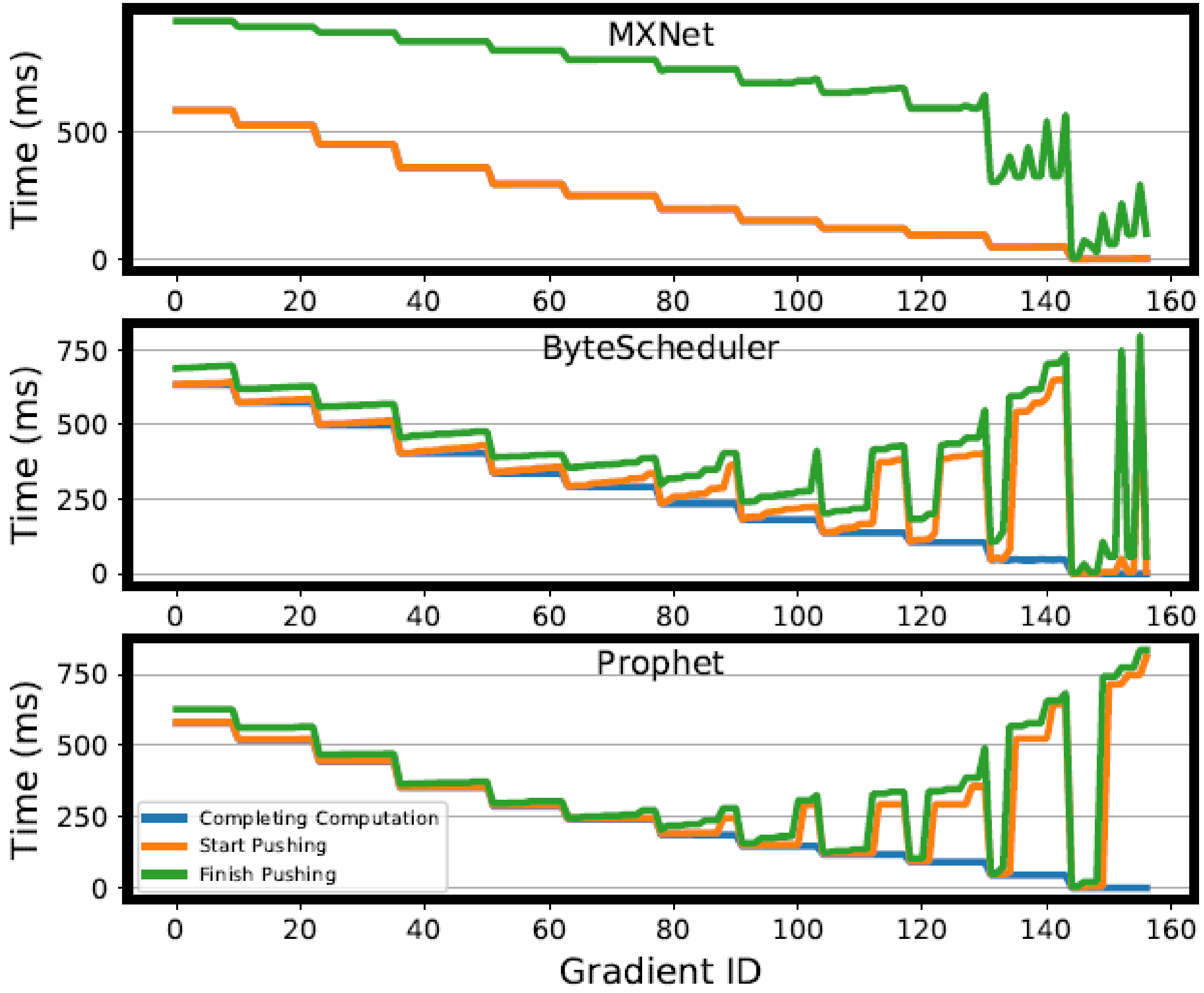






Evaluation
Effectiveness of *Prophet*

- ***Prophet*** **can significantly reduce both the wait time of gradient transfer and the transfer time of gradients.**



Evaluation
Robustness of *Prophet*

➢ ***Under different batch sizes:***

Model and batch size	Rate of *Prophet* (samples/sec)	Rate of ByteScheduler (samples/sec)	Performance improvement
ResNet18 (16)	32.46	29.06	11.6%
ResNet18 (64)	153	115	33%
ResNet50 (16)	14.44	14.22	1.5%
ResNet50 (32)	34.8	28.5	22%
ResNet50 (64)	60	44	36%

a larger mini-batch takes a longer time to compute the gradients, which inevitably makes the stepwise pattern more obvious and thus prolongs the time interval among blocks

➢ ***Under different bandwidth conditions:***

Prophet achieves relatively higher DDNN training performance by **11.7% - 39.1%** compared with default *MXNet* and *P3*.

➢ ***In heterogeneous environments:***

Both *Prophet* and *ByteScheduler* outperform the default *MXNet* in heterogeneous environments.

Prophet **slightly improves** the training performance by **2.3%** compared with *ByteScheduler* in heterogeneous environments.

Summary

- We design and implement a predictable communication scheduling strategy named *Prophet* to schedule the gradient transfer in an adequate order, with the aim of maximizing the GPU and network resource utilization.
- *Prophet* leverages our observed stepwise pattern of gradient transfer start time to make the forward propagation start as early as possible to greedily reduce the waiting (idle) time of GPU resources during the DDNN training process.
- *Prophet* can improve the DDNN training performance by up to 40% compared with the state-of-the-art priority-based communication scheduling strategies, yet with negligible runtime performance overhead.

Thank You

Zhenwei Zhang, Fei Xu*
"ECNU-iCloud" Research Group
School of Computer Science and Technology
East China Normal University

Prophet: Speeding up Distributed DNN Training with Predictable Communication Scheduling