Distributed Learning - Data Parallelization

Amir H. Payberah
payberah@kth.se

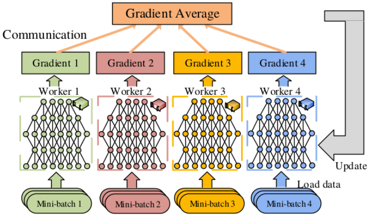
2020-10-12

The Course Web Page

https://fid3024.github.io

Data Parallelization (1/4)

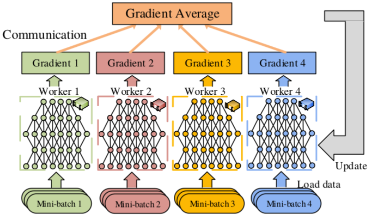
- Replicate a whole model on every device.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (1/4)

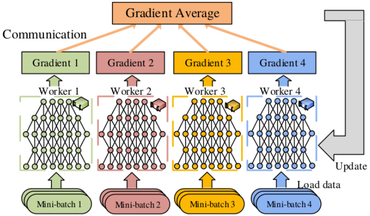
- Replicate a whole model on every device.
- Train all replicas simultaneously, using a different mini-batch for each.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

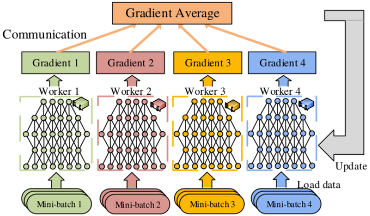
- k devices



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

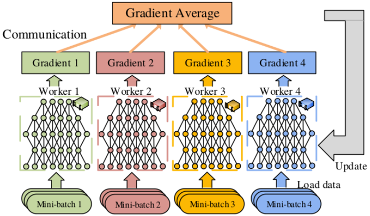
- $\mathtt{k}$ devices
- $\mathtt{J_i}(\mathbf{w}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \mathtt{l}(\mathbf{x},\mathbf{w}),\ \forall \mathtt{i} = 1, 2, \cdots, \mathtt{k}$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

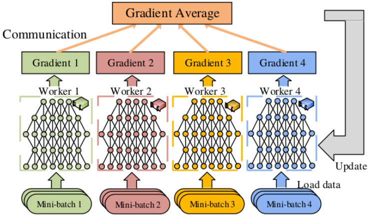
- $\mathtt{k}$ devices
- $\mathtt{J_i}(\mathbf{w}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \mathtt{l}(\mathbf{x}, \mathbf{w}), \forall \mathtt{i} = 1, 2, \cdots, \mathtt{k}$
- $\mathtt{G_i}(\mathbf{w}, \beta_\mathtt{i}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \nabla\mathtt{l}(\mathbf{w}, \mathbf{x})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

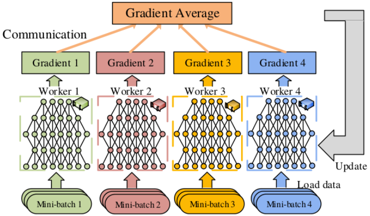
- $\mathtt{k}$ devices
- $\mathtt{J_i}(\mathbf{w}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \mathtt{l}(\mathbf{x},\mathbf{w})$, $\forall \mathtt{i} = 1, 2, \cdots, \mathtt{k}$
- $\mathtt{G_i}(\mathbf{w}, \beta_\mathtt{i}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \nabla \mathtt{l}(\mathbf{w},\mathbf{x})$
- $\mathtt{G_i}(\mathbf{w}, \beta_\mathtt{i})$: the local estimate of the gradient of the loss function $\nabla \mathtt{J_i}(\mathbf{w})$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (3/4)

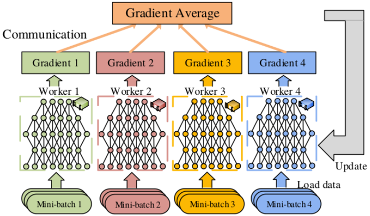
- Compute the gradients aggregation (e.g., mean of the gradients).
- $\mathtt{F}(\mathtt{G}_1, \cdots, \mathtt{G}_\mathtt{k}) = \frac{1}{\mathtt{k}} \sum_{\mathtt{i}=1}^{\mathtt{k}} \mathtt{G}_\mathtt{i}(\mathbf{w}, \beta_\mathtt{i})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (4/4)

- Update the model.
- $\mathbf{w} := \mathbf{w} - \eta F(G_1, \cdots, G_k)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization Design Issues

- The aggregation algorithm
- Communication synchronization and frequency
- Communication compression
- Parallelism of computations and communications

The Aggregation Algorithm

The Aggregation Algorithm

- How to aggregate gradients (compute the mean of the gradients)?

The Aggregation Algorithm

- How to aggregate gradients (compute the mean of the gradients)?
- Centralized - parameter server

The Aggregation Algorithm

- How to aggregate gradients (compute the mean of the gradients)?
- Centralized - parameter server
- Decentralized - all-reduce

The Aggregation Algorithm

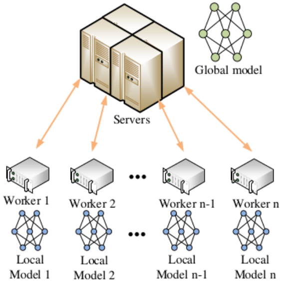
- How to aggregate gradients (compute the mean of the gradients)?
- Centralized - parameter server
- Decentralized - all-reduce
- Decentralized - gossip

Aggregation - Centralized - Parameter Server

- Store the model parameters outside of the workers.

Aggregation - Centralized - Parameter Server

- Store the model parameters outside of the workers.
- Workers periodically report their computed parameters or parameter updates to a (set of) parameter server(s) (PSs).



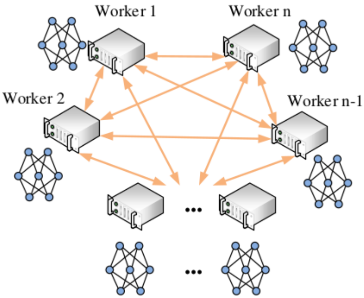
[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - All-Reduce

- Mirror all the model parameters across all workers (no PS).

Aggregation - Distributed - All-Reduce

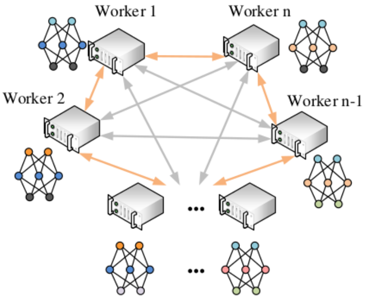
- Mirror all the model parameters across all workers (no PS).
- Workers exchange parameter updates directly via an allreduce operation.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

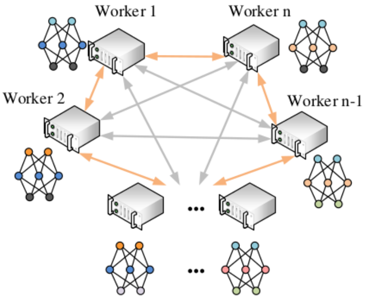
- No PS, and no global model.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

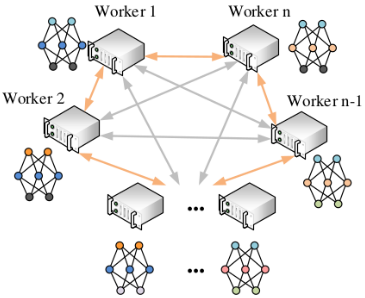
- No PS, and no global model.
- Every worker communicates updates with their neighbors.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

- No PS, and no global model.
- Every worker communicates updates with their neighbors.
- The consistency of parameters across all workers only at the end of the algorithm.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Reduce and AllReduce (1/2)

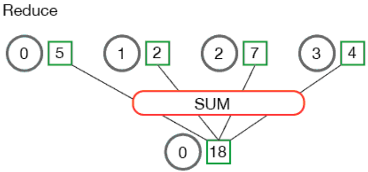
- Reduce: reducing a set of numbers into a smaller set of numbers via a function.

Reduce and AllReduce (1/2)

- Reduce: reducing a set of numbers into a smaller set of numbers via a function.
- E.g., `sum([1, 2, 3, 4, 5]) = 15`

Reduce and AllReduce (1/2)

- Reduce: reducing a set of numbers into a smaller set of numbers via a function.
- E.g., `sum([1, 2, 3, 4, 5]) = 15`
- Reduce takes an array of input elements on each process and returns an array of output elements to the root process.



Reduce and AllReduce (1/2)

- Reduce: reducing a set of numbers into a smaller set of numbers via a function.
- E.g., `sum([1, 2, 3, 4, 5]) = 15`
- Reduce takes an array of input elements on each process and returns an array of output elements to the root process.

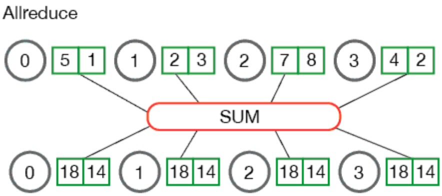
[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

Reduce and AllReduce (2/2)

- AllReduce stores reduced results across all processes rather than the root process.

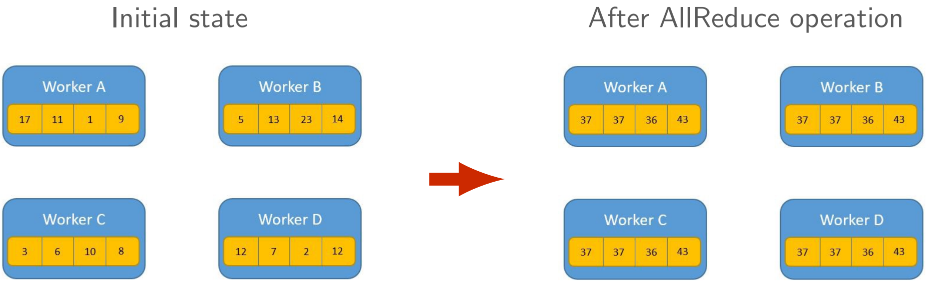
Reduce and AllReduce (2/2)

- AllReduce stores reduced results across all processes rather than the root process.



[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

AllReduce Example



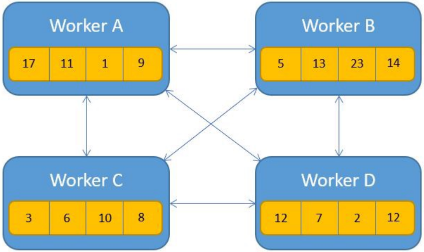
[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation

- All-to-all allreduce
- Master-worker allreduce
- Tree allreduce
- Round-robin allreduce
- Butterfly allreduce
- Ring allreduce

AllReduce Implementation - All-to-All AllReduce

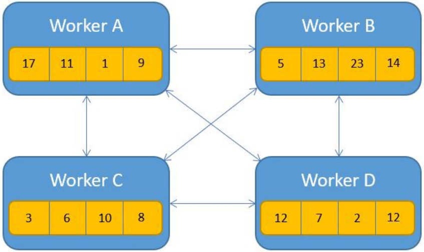
- Send the array of data to each other.
- Apply the reduction operation on each process.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - All-to-All AllReduce

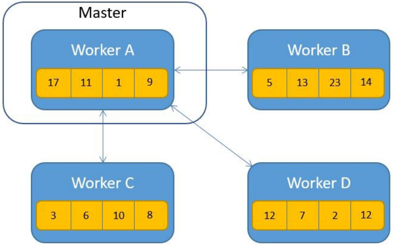
- Send the array of data to each other.
- Apply the reduction operation on each process.
- Too many unnecessary messages.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Master-Worker AllReduce

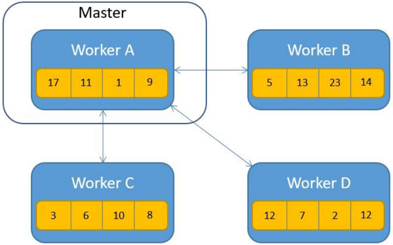
- Selecting one process as a master, gather all arrays into the master.
- Perform reduction operations locally in the master.
- Distribute the result to the other processes.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Master-Worker AllReduce

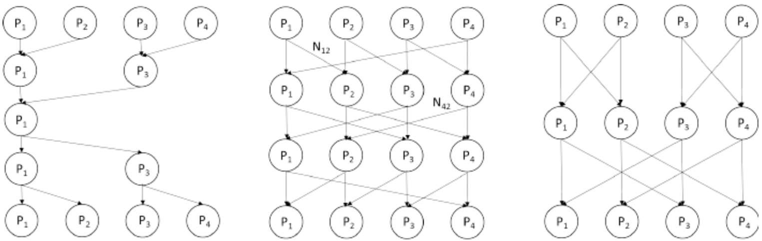
- Selecting one process as a master, gather all arrays into the master.
- Perform reduction operations locally in the master.
- Distribute the result to the other processes.
- The master becomes a bottleneck (not scalable).



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Other implementations

- Some try to minimize bandwidth.
- Some try to minimize latency.



(a) Tree AllReduce (b) Round-robin AllReduce (c) Butterfly AllReduce

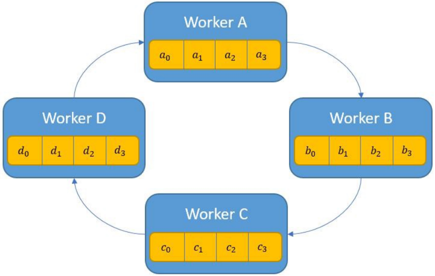
[Zhao H. et al., arXiv:1312.3020, 2013]

AllReduce Implementation - Ring-AllReduce (1/6)

- The Ring-Allreduce has two phases:
  1. First, the share-reduce phase
  2. Then, the share-only phase

AllReduce Implementation - Ring-AllReduce (2/6)

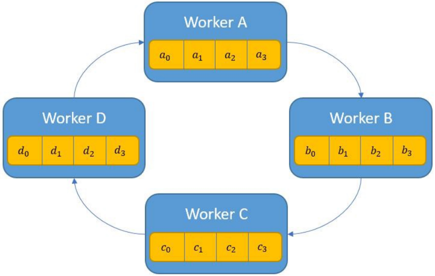
- In the share-reduce phase, each process `p` sends data to the process `(p+1)%m`
  - `m` is the number of processes, and `%` is the modulo operator.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (2/6)

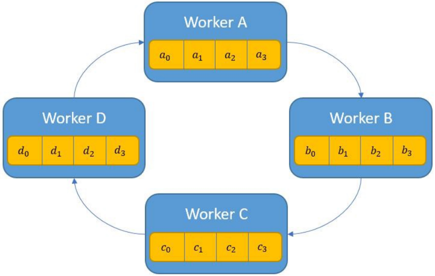
- In the share-reduce phase, each process `p` sends data to the process `(p+1)%m`
  - `m` is the number of processes, and `%` is the modulo operator.
- The array of data on each process is divided to `m` chunks (`m=4` here).



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (2/6)

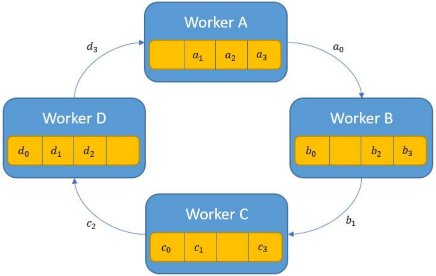
- In the share-reduce phase, each process `p` sends data to the process `(p+1)%m`
  - `m` is the number of processes, and `%` is the modulo operator.
- The array of data on each process is divided to `m` chunks (`m=4` here).
- Each one of these chunks will be indexed by `i` going forward.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (3/6)

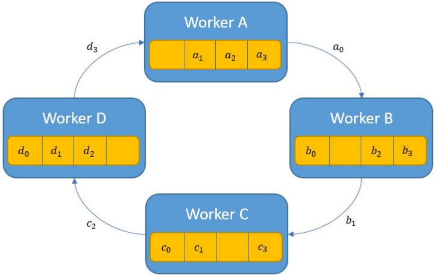
- In the first share-reduce step, process A sends a_0 to process B.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (3/6)

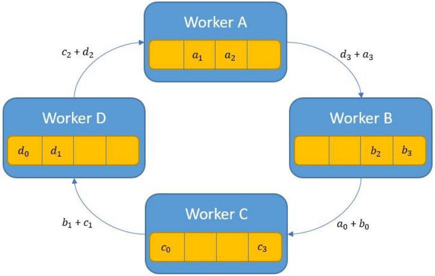
- In the first share-reduce step, process A sends a_0 to process B.
- Process B sends b_1 to process C, etc.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (4/6)

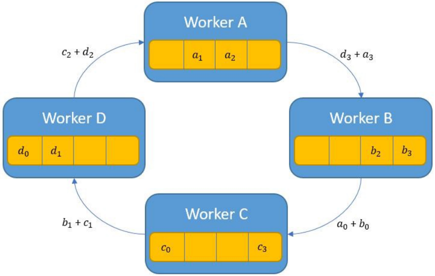
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (4/6)

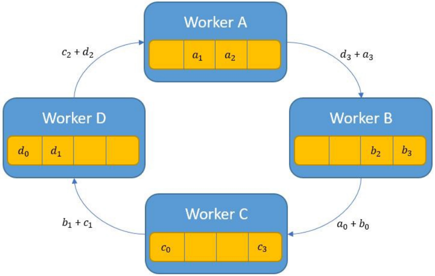
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
  - The reduce operator should be associative and commutative.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (4/6)

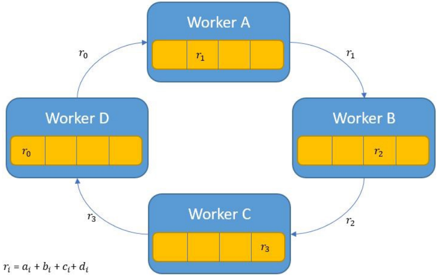
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
  - The reduce operator should be associative and commutative.
- It then proceeds to send it to the next process in the ring.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (5/6)

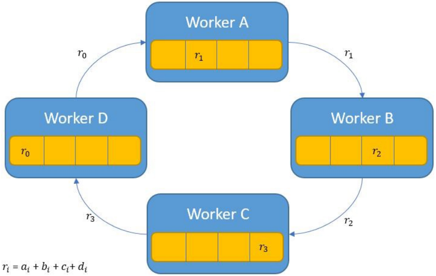
- The share-reduce phase finishes when each process holds the complete reduction of chunk `i`.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (5/6)

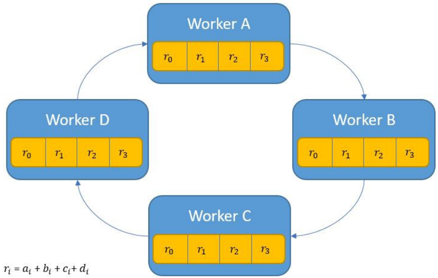
- The share-reduce phase finishes when each process holds the complete reduction of chunk `i`.
- At this point each process holds a part of the end result.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (6/6)

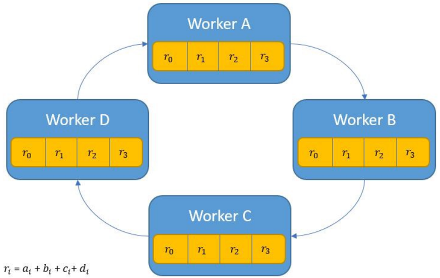
- The share-only step is the same process of sharing the data in a ring-like fashion without applying the reduce operation.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (6/6)

- The share-only step is the same process of sharing the data in a ring-like fashion without applying the reduce operation.
- This consolidates the result of each chunk in every process.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

Master-Worker AllReduce vs. Ring-AllReduce

- `N`: number of elements, `m`: number of processes

Master-Worker AllReduce vs. Ring-AllReduce

- `N`: number of elements, `m`: number of processes
- Master-Worker AllReduce

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.
- Ring-AllReduce

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.
- Ring-AllReduce
  - In the share-reduce step each process sends $\frac{\mathtt{N}}{\mathtt{m}}$ elements, and it does it $\mathtt{m} - 1$ times: $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.
- Ring-AllReduce
  - In the share-reduce step each process sends $\frac{\mathtt{N}}{\mathtt{m}}$ elements, and it does it $\mathtt{m} - 1$ times: $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.
  - On the share-only step, each process sends the result for the chunk it calculated: another $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.
- Ring-AllReduce
  - In the share-reduce step each process sends $\frac{\mathtt{N}}{\mathtt{m}}$ elements, and it does it $\mathtt{m} - 1$ times: $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.
  - On the share-only step, each process sends the result for the chunk it calculated: another $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1))$.

Communication Synchronization and Frequency

Synchronization

- When to synchronize the parameters among the parallel workers?

Communication Synchronization (1/2)

- Synchronizing the model replicas in data-parallel training requires communication
  - between workers, in allreduce
  - between workers and parameter servers, in the centralized architecture

Communication Synchronization (1/2)

- Synchronizing the model replicas in data-parallel training requires communication
  - between workers, in allreduce
  - between workers and parameter servers, in the centralized architecture

- The communication synchronization decides how frequently all local models are synchronized with others.

Communication Synchronization (2/2)

- It will influence:
  - The communication traffic
  - The performance
  - The convergence of model training

Communication Synchronization (2/2)

- It will influence:
  - The communication traffic
  - The performance
  - The convergence of model training
- There is a trade-off between the communication traffic and the convergence.

Reducing Synchronization Overhead

- Two directions for improvement:

Reducing Synchronization Overhead

- Two directions for improvement:
    1. To relax the synchronization among all workers.

Reducing Synchronization Overhead

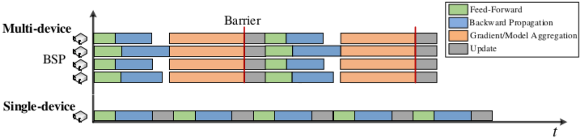
- Two directions for improvement:
  1. To relax the synchronization among all workers.
  2. The frequency of communication can be reduced by more computation in one iteration.

Communication Synchronization Models

- Synchronous
- Stale-synchronous
- Asynchronous
- Local SGD

Communication Synchronization - Synchronous

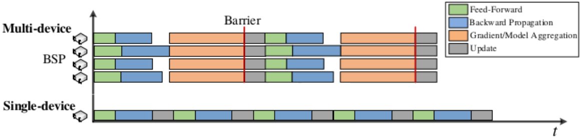
- After each iteration, the workers synchronize their parameter updates.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

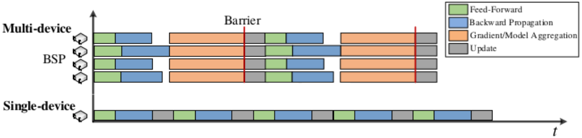
- After each iteration, the workers synchronize their parameter updates.
- Every worker must wait for all workers to finish the transmission of all parameters in the current iteration, before the next training.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

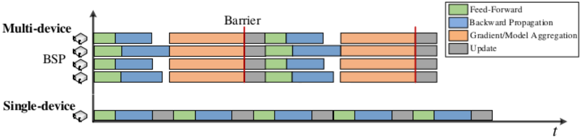
- After each iteration, the workers synchronize their parameter updates.
- Every worker must wait for all workers to finish the transmission of all parameters in the current iteration, before the next training.
- Stragglers can influence the overall system throughput.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

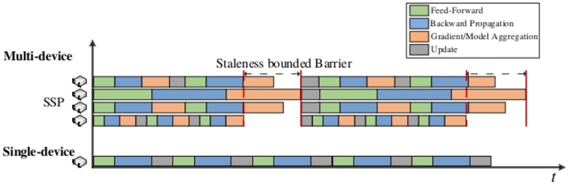
- After each iteration, the workers synchronize their parameter updates.
- Every worker must wait for all workers to finish the transmission of all parameters in the current iteration, before the next training.
- Stragglers can influence the overall system throughput.
- High communication cost that limits the system scalability.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

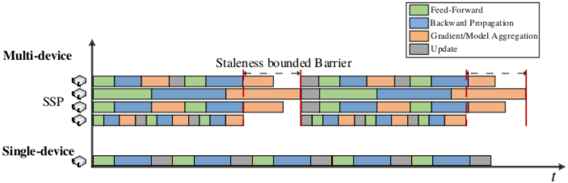
- Alleviate the straggler problem without losing synchronization.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

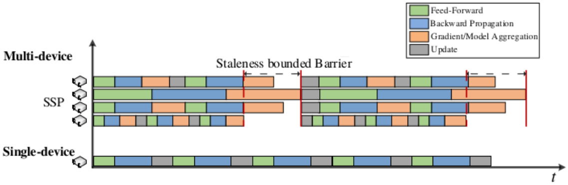
- Alleviate the straggler problem without losing synchronization.
- The faster workers to do more updates than the slower workers to reduce the waiting time of the faster workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

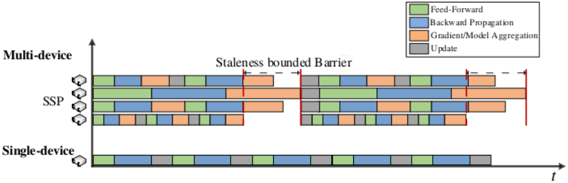
- Alleviate the straggler problem without losing synchronization.
- The faster workers to do more updates than the slower workers to reduce the waiting time of the faster workers.
- Staleness bounded barrier to limit the iteration gap between the fastest worker and the slowest worker.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

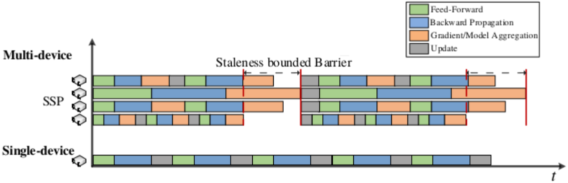
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k})\in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

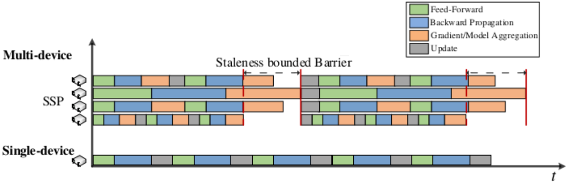
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k})\in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- The update has three parts:



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

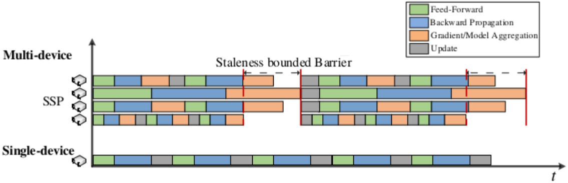
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k}) \in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- The update has three parts:
  1. Guaranteed pre-window updates from clock $\mathtt{1}$ to $\mathtt{t}$ over all workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

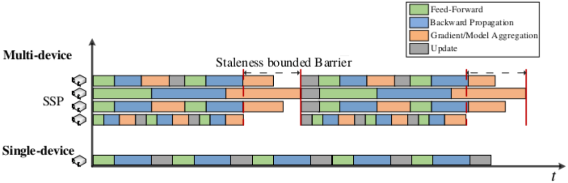
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k})\in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- The update has three parts:
  1. Guaranteed pre-window updates from clock $\mathtt{1}$ to $\mathtt{t}$ over all workers.
  2. Guaranteed read-my-writes in-window updates made by the querying worker $\mathtt{i}$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

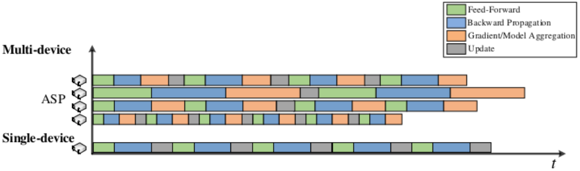
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+\mathtt{1}$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k})\in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- The update has three parts:
  1. Guaranteed pre-window updates from clock $\mathtt{1}$ to $\mathtt{t}$ over all workers.
  2. Guaranteed read-my-writes in-window updates made by the querying worker $\mathtt{i}$.
  3. Best-effort in-window updates. $\mathtt{S}_{\mathtt{i},\mathtt{t}+1}$ is some subset of the updates from other workers during period $[\mathtt{t}-\mathtt{s}]$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

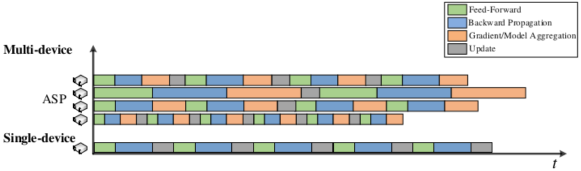
- It completely eliminates the synchronization.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

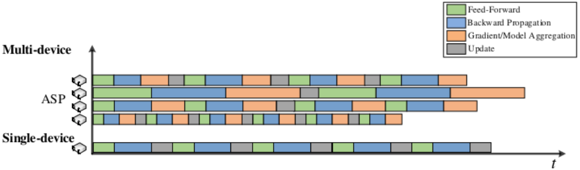
- It completely eliminates the synchronization.
- Each work transmits its gradients to the PS after it calculates the gradients.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

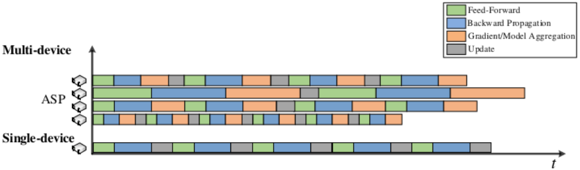
- It completely eliminates the synchronization.
- Each work transmits its gradients to the PS after it calculates the gradients.
- The PS updates the global model without waiting for the other workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (2/2)

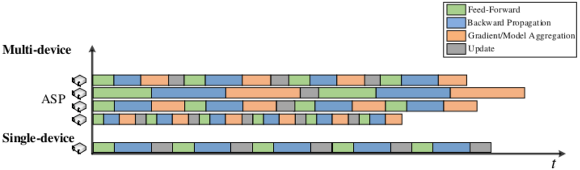
- $\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \sum_{i=1}^{n} G_{i,t-\tau_{k,i}}$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (2/2)

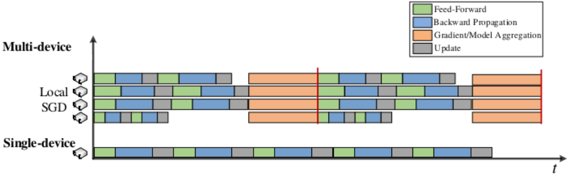
- $\mathbf{w}_{\mathtt{t+1}} := \mathbf{w}_{\mathtt{t}} - \eta \sum_{\mathtt{i=1}}^{\mathtt{n}} \mathtt{G}_{\mathtt{i,t}-\tau_{\mathtt{k,i}}}$
- $\tau_{\mathtt{k,i}}$ is the time delay between the moment when worker $\mathtt{i}$ calculates the gradient at the current iteration.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Local SGD

- All workers run several iterations, and then averages all local models into the newest global model.

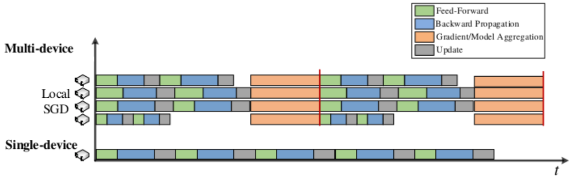


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Local SGD

- All workers run several iterations, and then averages all local models into the newest global model.
- If $\mathcal{I}_T$ represents the synchronization timestamps, then:

$$\mathbf{w}_{i,t+1} = \begin{cases} \mathbf{w}_{i,t} - \eta G_{i,t} & \text{if} \quad t+1 \notin \mathcal{I}_T \\ \mathbf{w}_{i,t} - \eta \frac{1}{n} \sum_{i=1}^{n} G_{i,t} & \text{if} \quad t+1 \in \mathcal{I}_T \end{cases}$$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression

Communication Compression

- Reduce the communication traffic with little impact on the model convergence.

Communication Compression

- Reduce the communication traffic with little impact on the model convergence.
- Compress the exchanged gradients or models before transmitting across the network.

Communication Compression

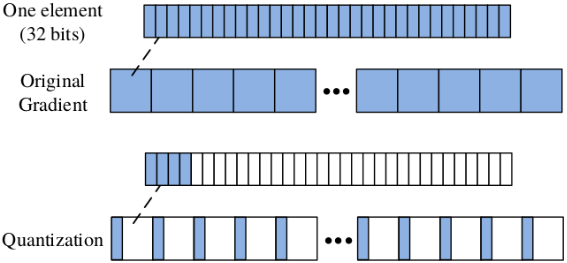
- Reduce the communication traffic with little impact on the model convergence.
- Compress the exchanged gradients or models before transmitting across the network.
- Quantization

Communication Compression

- Reduce the communication traffic with little impact on the model convergence.
- Compress the exchanged gradients or models before transmitting across the network.
- Quantization
- Sparsification

Communication Compression - Quantization

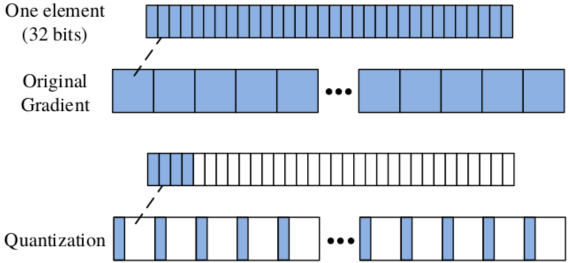
- Useing lower bits to represent the data.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Quantization

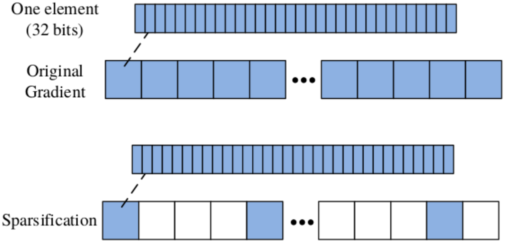
- Useing lower bits to represent the data.
- The gradients are of low precision.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

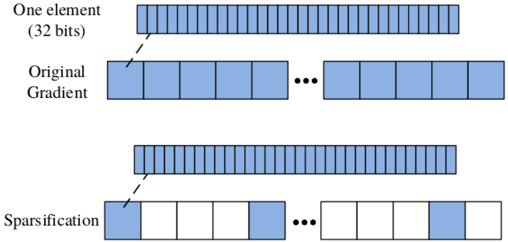
- Reducing the number of elements that are transmitted at each iteration.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

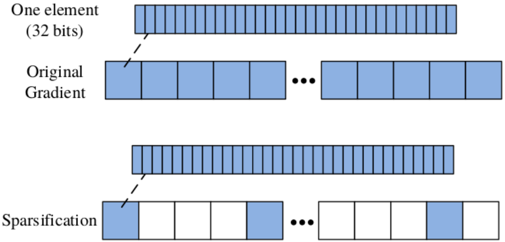
- Reducing the number of elements that are transmitted at each iteration.
- Only significant gradients are required to update the model parameter to guarantee the convergence of the training.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

- Reducing the number of elements that are transmitted at each iteration.
- Only significant gradients are required to update the model parameter to guarantee the convergence of the training.
- E.g., the zero-valued elements are no need to transmit.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Parallelism of Computations and Communications

Parallelism of Computations and Communications (1/3)

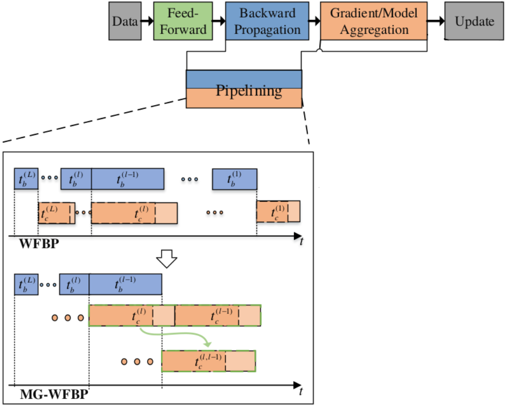
- The layer-wise structure of deep models makes it possible to parallels the communication and computing tasks.

Parallelism of Computations and Communications (1/3)

- The layer-wise structure of deep models makes it possible to parallels the communication and computing tasks.
- Optimizing the order of computation and communication such that the communication cost can be minimized

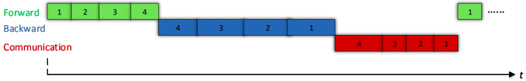
Parallelism of Computations and Communications (2/3)

- Wait-free backward propagation (WFBP)
- Merged-gradient WFBP (MG-WFBP)

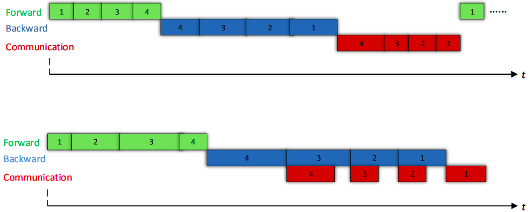


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Parallelism of Computations and Communications (3/3)

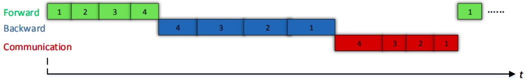


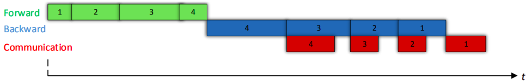
Parallelism of Computations and Communications (3/3)



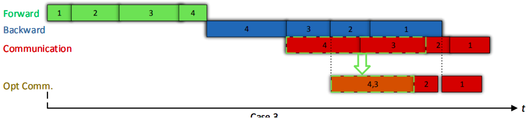
Wait-free backward propagation (WFBP)

Parallelism of Computations and Communications (3/3)





Wait-free backward propagation (WFBP)



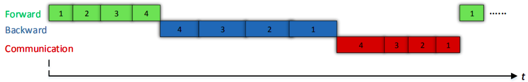
Merged-gradient WFBP (MG-WFBP)

[shi et al., MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms, 2018]

TicTac: Accelerating Distributed Deep Learning with Communication Scheduling

Computation vs. Communication

- The iteration time in deep learning systems depends on the time taken by
  1. Computation
  2. Communication
  3. The overlap between the two

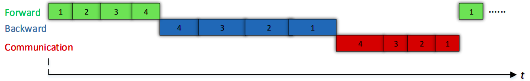


[shi et al., MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms, 2018]

Computation vs. Communication

- The iteration time in deep learning systems depends on the time taken by
  1. Computation
  2. Communication
  3. The overlap between the two

- When workers receive the parameters from the PS at the beginning of each iteration, all parameters are not used simultaneously.



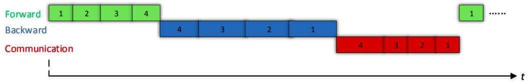
[shi et al., MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms, 2018]

Computation vs. Communication

- The iteration time in deep learning systems depends on the time taken by
  1. Computation
  2. Communication
  3. The overlap between the two

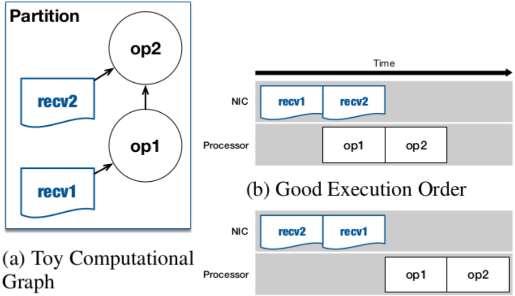
- When workers receive the parameters from the PS at the beginning of each iteration, all parameters are not used simultaneously.

- Identifying the best schedule of parameter transfers is critical for reducing the blocking on computation.



[shi et al., MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms, 2018]

Good vs. Bad Execution Order



(a) Toy Computational Graph

(b) Good Execution Order

(c) Bad Execution Order

[Hashemi et al., TicTac: Accelerating Distributed Deep Learning with Communication Scheduling, 2019]

High GPU Utilization

- High GPU utilization can be achieved in two ways:

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.
  2. With efficient overlap of communication and computation.

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.
  2. With efficient overlap of communication and computation.

- Techniques improve GPU utilization:

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.
  2. With efficient overlap of communication and computation.

- Techniques improve GPU utilization:
  - Increasing computation time

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.
  2. With efficient overlap of communication and computation.

- Techniques improve GPU utilization:
  - Increasing computation time
  - Decreasing communication time

High GPU Utilization

- High GPU utilization can be achieved in two ways:
  1. When total communication time is less than or equal to the computation time.
  2. With efficient overlap of communication and computation.

- Techniques improve GPU utilization:
  - Increasing computation time
  - Decreasing communication time
  - Better interleaving of computation and communication

Overlap Coefficient (1/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N+C-T}}{\mathtt{min(N,C)}}$

Overlap Coefficient (1/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N}+\mathtt{C}-\mathtt{T}}{\mathtt{min(N,C)}}$
- $\mathtt{T}$: the actual iteration time

Overlap Coefficient (1/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N+C-T}}{\mathtt{min(N,C)}}$
- $\mathtt{T}$: the actual iteration time
- $\mathtt{N}$: the communication time

Overlap Coefficient (1/2)

- Overlap coefficient: $\alpha = \frac{\texttt{N}+\texttt{C}-\texttt{T}}{\texttt{min(N,C)}}$
- `T`: the actual iteration time
- `N`: the communication time
- `C`: the computation time

Overlap Coefficient (1/2)

- Overlap coefficient: $\alpha = \frac{\texttt{N}+\texttt{C}-\texttt{T}}{\texttt{min}(\texttt{N},\texttt{C})}$
- $\texttt{T}$: the actual iteration time
- $\texttt{N}$: the communication time
- $\texttt{C}$: the computation time
- $\texttt{N} + \texttt{C}$ is the iteration time when there is no overlap

Overlap Coefficient (2/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N+C-T}}{\mathtt{min(N,C)}}$

Overlap Coefficient (2/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N+C-T}}{\mathtt{min(N,C)}}$
- The maximum overlap possible is given by $\mathtt{min}(\mathtt{N}, \mathtt{C})$, which is achieved when the smaller quantity completely overlaps with the large quantity.

Overlap Coefficient (2/2)

- Overlap coefficient: $\alpha = \frac{\mathtt{N}+\mathtt{C}-\mathtt{T}}{\mathtt{min}(\mathtt{N},\mathtt{C})}$
- The maximum overlap possible is given by $\mathtt{min}(\mathtt{N},\mathtt{C})$, which is achieved when the smaller quantity completely overlaps with the large quantity.
- GPU utilization: $\mathtt{U} = \frac{\mathtt{C}}{\mathtt{T}} = \frac{\mathtt{C}}{\mathtt{N}+\mathtt{C}-\alpha\mathtt{min}(\mathtt{N},\mathtt{C})} = \frac{1}{1+\rho-\alpha\mathtt{min}(\rho,1)}$
- $\rho = \frac{\mathtt{N}}{\mathtt{C}}$: the communication/computation ratio

Scheduling Algorithm

- Prioritize transfers that speed up the critical path in the DAG, by reducing blocking on computation caused by parameter transfers.

Scheduling Algorithm

- Prioritize transfers that speed up the critical path in the DAG, by reducing blocking on computation caused by parameter transfers.
- TIC: Timing-Independent Communication scheduling

Scheduling Algorithm

- Prioritize transfers that speed up the critical path in the DAG, by reducing blocking on computation caused by parameter transfers.
- TIC: Timing-Independent Communication scheduling
- TAC: Timing-Aware Communication scheduling

Scheduling Algorithm

- Prioritize transfers that speed up the critical path in the DAG, by reducing blocking on computation caused by parameter transfers.
- TIC: Timing-Independent Communication scheduling
  - Prioritize those transfers that reduces blocking on network transfers.
- TAC: Timing-Aware Communication scheduling

Scheduling Algorithm

- Prioritize transfers that speed up the critical path in the DAG, by reducing blocking on computation caused by parameter transfers.
- TIC: Timing-Independent Communication scheduling
  - Prioritize those transfers that reduces blocking on network transfers.
- TAC: Timing-Aware Communication scheduling
  - Prioritize those transfers that reduces the blocking of computation.

TIC

- Prioritize those transfers that reduces blocking on network transfers.

TIC

- Prioritize those transfers that reduces blocking on network transfers.
- Prioritize based only on vertex dependencies in the DAG.

TIC

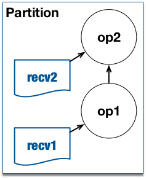
- Prioritize those transfers that reduces blocking on network transfers.
- Prioritize based only on vertex dependencies in the DAG.
- Higher priorities are given to transfers that are least blocking on computation.

TIC

- Prioritize those transfers that reduces blocking on network transfers.
- Prioritize based only on vertex dependencies in the DAG.
- Higher priorities are given to transfers that are least blocking on computation.
- Ignore the ops time, and use the number of communication ops instead.

TIC

- Prioritize those transfers that reduces blocking on network transfers.
- Prioritize based only on vertex dependencies in the DAG.
- Higher priorities are given to transfers that are least blocking on computation.
- Ignore the ops time, and use the number of communication ops instead.
- E.g., $\mathtt{op}_1.\mathtt{M} = \mathtt{Time}(\mathtt{recv}_1)$ and $\mathtt{op}_2.\mathtt{M} = \mathtt{Time}(\mathtt{recv}_1) + \mathtt{Time}(\mathtt{recv}_2)$.



TAC

- Prioritize those transfers that reduces the blocking of computation.

TAC

- Prioritize those transfers that reduces the blocking of computation.
- Prioritize transfers that maximize α by using information on:

TAC

- Prioritize those transfers that reduces the blocking of computation.
- Prioritize transfers that maximize α by using information on:
    1. Vertex dependencies among ops specified by the computational DAG.

TAC

- Prioritize those transfers that reduces the blocking of computation.
- Prioritize transfers that maximize α by using information on:
  1. Vertex dependencies among ops specified by the computational DAG.
  2. Execution time of each op estimated with time oracle.

TAC

- Prioritize those transfers that reduces the blocking of computation.
- Prioritize transfers that maximize α by using information on:
  1. Vertex dependencies among ops specified by the computational DAG.
  2. Execution time of each op estimated with time oracle.
- To achieve this goal, the algorithm focuses on two cases:
  1. Any communication and computation overlapping?
  2. If no, choose one which eliminates the computation block sooner.

CARAMEL: Accelerating Decentralized Distributed Deep Learning with Computation Scheduling

CARAMEL

- Decentralized aggregation (no PS)

CARAMEL

- Decentralized aggregation (no PS)
- Improve efficiency of decentralized DNN training

CARAMEL

- Decentralized aggregation (no PS)
- Improve efficiency of decentralized DNN training
- In terms of iteration time and GPU utilization

CARAMEL

- Decentralized aggregation (no PS)
- Improve efficiency of decentralized DNN training
- In terms of iteration time and GPU utilization
- CARAMEL achieves this goal through:

CARAMEL

- Decentralized aggregation (no PS)
- Improve efficiency of decentralized DNN training
- In terms of iteration time and GPU utilization
- CARAMEL achieves this goal through:
  1. Computation scheduling that expands the feasible window of transfer for each parameter (transfer boundaries)

CARAMEL

- Decentralized aggregation (no PS)
- Improve efficiency of decentralized DNN training
- In terms of iteration time and GPU utilization
- CARAMEL achieves this goal through:
  1. Computation scheduling that expands the feasible window of transfer for each parameter (transfer boundaries)
  2. Network optimizations that smoothen the load

Computation Scheduling (1/2)

- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.

Computation Scheduling (1/2)

- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.

- There are multiple feasible orders for executing operations in a DAG.

Computation Scheduling (1/2)

- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.
- There are multiple feasible orders for executing operations in a DAG.
- The parameters may become available at different workers in varying orders.

Computation Scheduling (1/2)

- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.
- There are multiple feasible orders for executing operations in a DAG.
- The parameters may become available at different workers in varying orders.
- The transfer boundaries of a parameter represent the window when a parameter can be aggregated without blocking computation.

Computation Scheduling (1/2)

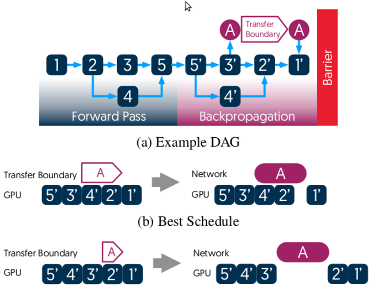
- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.
- There are multiple feasible orders for executing operations in a DAG.
- The parameters may become available at different workers in varying orders.
- The transfer boundaries of a parameter represent the window when a parameter can be aggregated without blocking computation.
- The start boundary is determined by the completion of the computation operation that updates the parameter.

Computation Scheduling (1/2)

- In decentralized aggregation, all workers should have the parameter available for aggregation before the transfer can be initiated.
- There are multiple feasible orders for executing operations in a DAG.
- The parameters may become available at different workers in varying orders.
- The transfer boundaries of a parameter represent the window when a parameter can be aggregated without blocking computation.
- The start boundary is determined by the completion of the computation operation that updates the parameter.
- The end boundary is the computation operation that reads the parameter.

Computation Scheduling (2/2)

- CARAMEL expands these boundaries through scheduling optimizations of the computation DAG, where it



(a) Example DAG

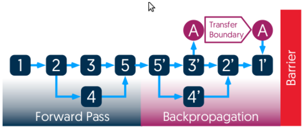
(b) Best Schedule

(c) Worst Schedule

[Hashemi et al., CARAMEL: Accelerating Decentralized Distributed Deep Learning with Model-Aware Scheduling, 2020]

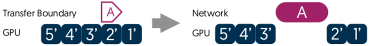
Computation Scheduling (2/2)

- CARAMEL expands these boundaries through scheduling optimizations of the computation DAG, where it
  1. Moves the start boundaries earlier.



(a) Example DAG

(b) Best Schedule

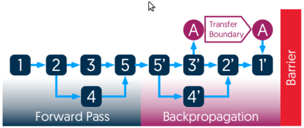


(c) Worst Schedule

[Hashemi et al., CARAMEL: Accelerating Decentralized Distributed Deep Learning with Model-Aware Scheduling, 2020]

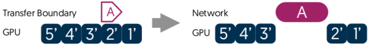
Computation Scheduling (2/2)

- CARAMEL expands these boundaries through scheduling optimizations of the computation DAG, where it
  1. Moves the start boundaries earlier.
  2. Pushes the end boundary by postponing the execution of some computation operations to the forward pass of next iteration.



(a) Example DAG

(b) Best Schedule



(c) Worst Schedule

[Hashemi et al., CARAMEL: Accelerating Decentralized Distributed Deep Learning with Model-Aware Scheduling, 2020]

Network Optimization

- Optimizations for smoothening the network load include:

Network Optimization

- Optimizations for smoothening the network load include:
  1. Batching of small parameters to reduce the network overhead.

Network Optimization

- Optimizations for smoothening the network load include:
  1. Batching of small parameters to reduce the network overhead.
  2. Adaptive splitting and pipelining of parameters to accelerate aggregation of large data.

Defining the Environment

- `T`: the actual iteration time
- `N`: the communication time
- `C`: the computation time

Defining the Environment

- $\mathtt{T}$: the actual iteration time
- $\mathtt{N}$: the communication time
- $\mathtt{C}$: the computation time
- Overlap coefficient: $\alpha = \frac{\mathtt{N+C-T}}{\mathtt{min(N,C)}}$

Defining the Environment

- $\mathtt{T}$: the actual iteration time
- $\mathtt{N}$: the communication time
- $\mathtt{C}$: the computation time
- Overlap coefficient: $\alpha = \frac{\mathtt{N}+\mathtt{C}-\mathtt{T}}{\mathtt{min}(\mathtt{N},\mathtt{C})}$
- GPU utilization: $\mathtt{U} = \frac{\mathtt{C}}{\mathtt{T}} = \frac{\mathtt{C}}{\mathtt{N}+\mathtt{C}-\alpha\mathtt{min}(\mathtt{N},\mathtt{C})} = \frac{1}{1+\rho-\alpha\mathtt{min}(\rho,1)}$
- $\rho = \frac{\mathtt{N}}{\mathtt{C}}$: the communication/computation ratio

CARAMEL Algorithm

- Dataflow DAG Optimizer
- Network Transfer Scheduler
- Parameter Batcher
- Adaptive Depth Enforcer

Dataflow DAG Optimizer

- Stage 1: Determining the best order.

Dataflow DAG Optimizer

- Stage 1: Determining the best order.
- Stage 2: Enforcing the best order.

Dataflow DAG Optimizer

- Stage 1: Determining the best order.
  - Increasing the overlap coefficient α by prioritizing the computations that activates the communication operations as early as possible.
- Stage 2: Enforcing the best order.

Dataflow DAG Optimizer

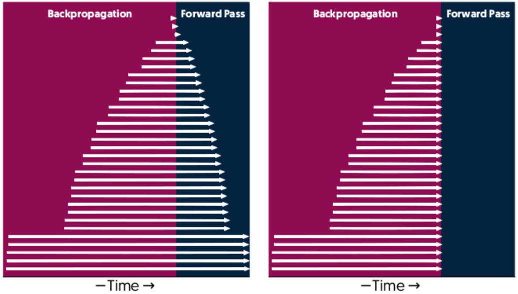
- Stage 1: Determining the best order.
  - Increasing the overlap coefficient α by prioritizing the computations that activates the communication operations as early as possible.
- Stage 2: Enforcing the best order.
  - Iteratively activate parameters in the best order chosen in the previous stage.

Dataflow DAG Optimizer

- Stage 1: Determining the best order.
  - Increasing the overlap coefficient α by prioritizing the computations that activates the communication operations as early as possible.
- Stage 2: Enforcing the best order.
  - Iteratively activate parameters in the best order chosen in the previous stage.
  - Ensureing that at each given time, only ops needed for the target parameter update can be executed.

Network Transfer Scheduler

- Increasing the overlap coefficient α by scheduling parameter transfers efficiently.
- Transfers are scheduled in both backward pass and forward pass.



(a) Parameter Server (b) Decentralized aggregation

[Hashemi et al., CARAMEL: Accelerating Decentralized Distributed Deep Learning with Model-Aware Scheduling, 2020]

Parameter Batcher

- Small parameters incur large overhead.

Parameter Batcher

- Small parameters incur large overhead.
- Combining small parameters in to groups.

Parameter Batcher

- Small parameters incur large overhead.
- Combining small parameters in to groups.
- Parameters larger than a certain threshold are transferred without batching.

Adaptive Depth Enforcer

- Two stages in decentralized algorithms: transferring and aggregating data across nodes.

Adaptive Depth Enforcer

- Two stages in decentralized algorithms: transferring and aggregating data across nodes.
  - In each step, data is transferred on the network, and is sent to application to be reduced, before the result is sent again over the network.

Adaptive Depth Enforcer

- Two stages in decentralized algorithms: transferring and aggregating data across nodes.
  - In each step, data is transferred on the network, and is sent to application to be reduced, before the result is sent again over the network.
- This process reduces the network utilization since the network is not utilized during the reduction at the application layer.

Adaptive Depth Enforcer

- Two stages in decentralized algorithms: transferring and aggregating data across nodes.
  - In each step, data is transferred on the network, and is sent to application to be reduced, before the result is sent again over the network.
- This process reduces the network utilization since the network is not utilized during the reduction at the application layer.
- Chunk (break) the data in to a few pieces, and transfer each chunk independently in parallel.

Adaptive Depth Enforcer

- Two stages in decentralized algorithms: transferring and aggregating data across nodes.
  - In each step, data is transferred on the network, and is sent to application to be reduced, before the result is sent again over the network.
- This process reduces the network utilization since the network is not utilized during the reduction at the application layer.
- Chunk (break) the data in to a few pieces, and transfer each chunk independently in parallel.
- While one chunk is being reduced on the CPU, another chunk can be sent over the network: this enables pipelining of network transfer and application-level processing across various chunks.

Summary

Summary

- Data-parallel
- The aggregation algorithm
- Communication synchronization
- Communication compression
- Parallelism of computations and communications
- TicTac
- Caramel

Reference

- Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020
- Hashemi et al., TicTac: Accelerating Distributed Deep Learning with Communication Scheduling, 2019
- Hashemi et al., CARAMEL: Accelerating Decentralized Distributed Deep Learning with Model-Aware Scheduling, 2020

Questions?