Benchmarking AWS 100Gbps network

Yaroslav Bulatov

Summary **1**

Background and notation. **1**

Algorithm bandwidth 1

Optimal algbw over 100 Gbps network 2

Two machine case 3

Benchmarks **4**

nccl-test 4

synthetic PyTorch 4

Imagenet in 18 mins 5

Comparison with on-prem Infiniband 6

Environment setup notes **6**

Summary

- AWS allreduce of 4GB across 256 GPUs takes ≈1 second, 62% of theoretical max
- On-prem allreduce efficiency is 94% of theoretical max for 16 machines
- AWS nccl-test results are 55-67% of theoretical max for 2-32 machines
- AWS PyTorch results are 50-54% of theoretical max for 2-4 machines

Background and notation.

Algorithm bandwidth

Summarizing performance page from nccl-tests

"Algorithm bandwidth is using the most commonly used formula for bandwidth : size (*S*) / time (*t*). It is useful to compute how much time any large operation would take by simply dividing the size of the operation by the algorithm bandwidth."

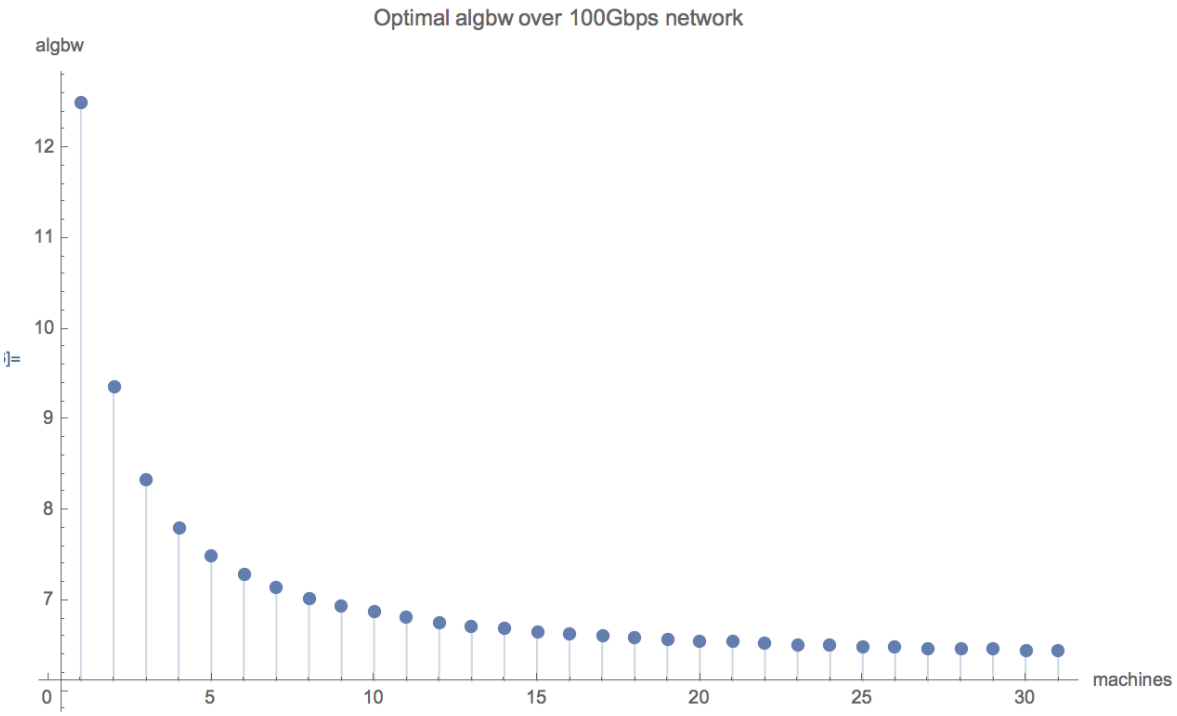
Since the size of the problem is measured in bytes rather than bits, algbw is also reported as GB/s rather than Gbps.

For instance, if we are reducing 4GB tensors and our algorithmic bandwidth is 4 GB/s, each operation would take 1 second.

Optimal algbw over 100 Gbps network

Optimal algorithmic bandwidth across n nodes with link bandwidth B is the following:

$$\text{algbw} = \frac{Bn}{2(n-1)}$$



Theoretical maximum alreduce algbw for 100 Gbps network

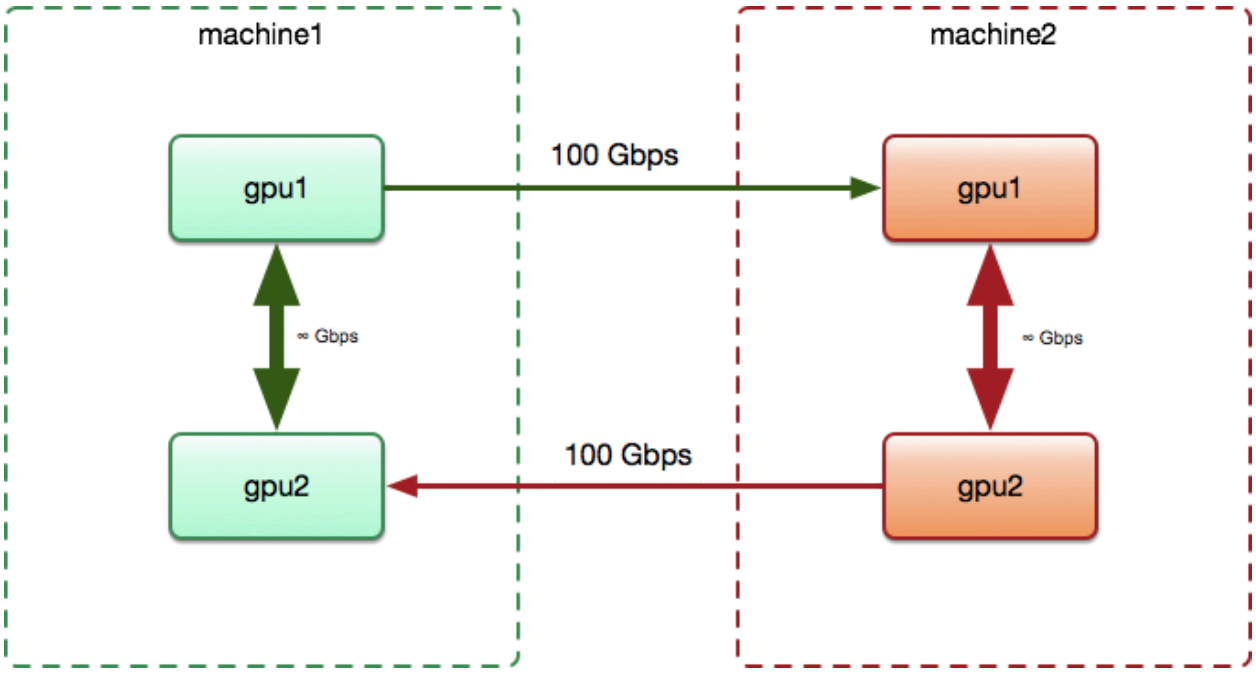
# of machines	optimal algbw GB/second
2	12.5

4	8.33
8	7.14
16	6.67
24	6.52
32	6.45

If we use hierarchical aggregation and within node aggregation time is negligible, algorithm bandwidth is determined by bandwidth between machines. For 100 Gbps network, algorithmic bandwidth is between 6.25 (infinite machines) and 12.5 (two machines)

Two machine case

For the case of 2 nodes, algorithmic bandwidth is equal to bus bandwidth -- 12.5 GB/second. The assumption here is that aggregation is done hierarchically and within-machine aggregation time is negligible compared to between-machine. For full duplex network, the allreduce time is determined solely by the time to send the load one way over the network.



Benchmarks

nccl-test

overview here, raw measurements, code

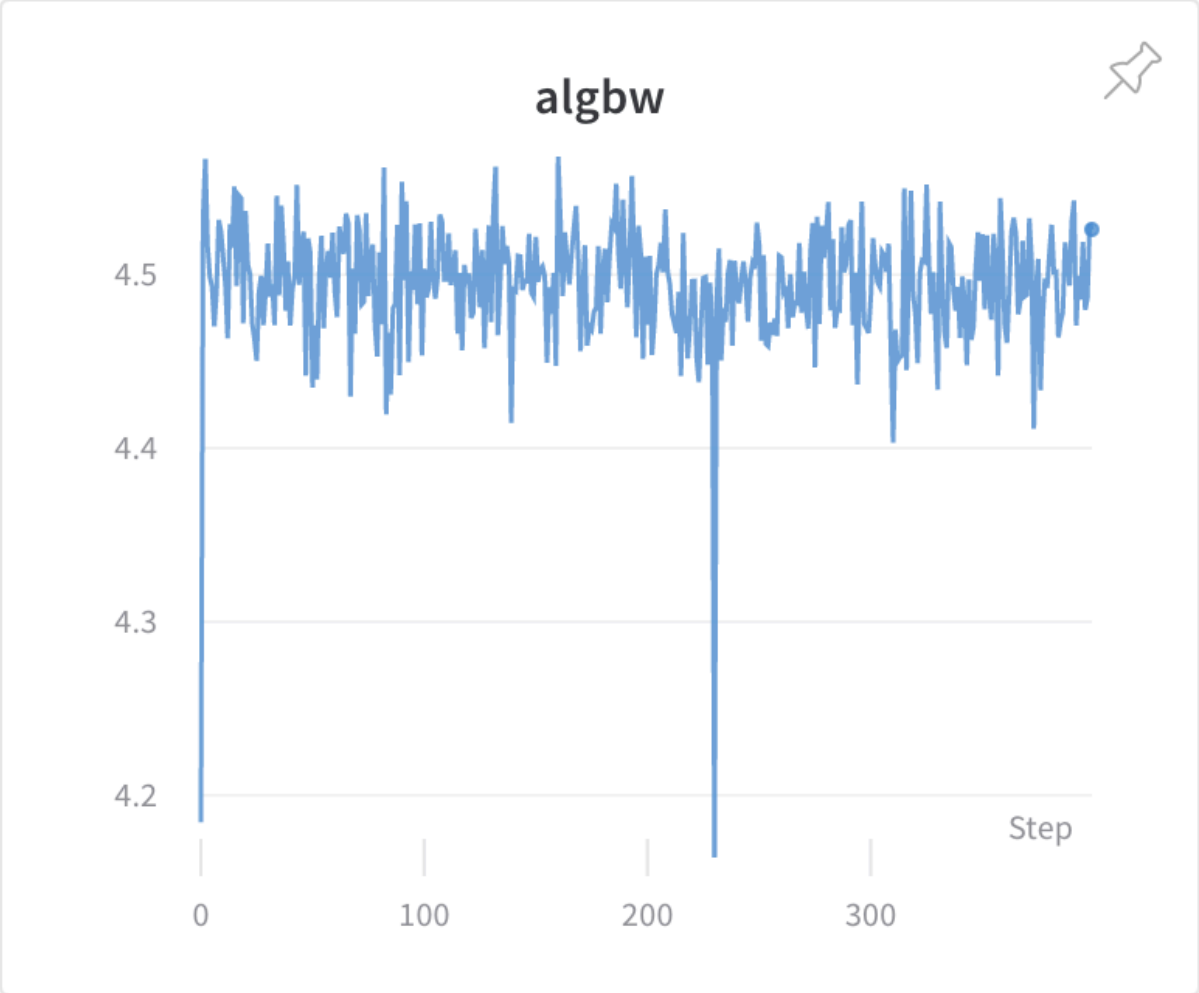
Summary:
- 8.42 algbw for 2 machines (67% of theoretical max) run
- 3.53 algbw for 32 machines (55% of theoretical max), run

synthetic PyTorch

Approximate common transformer architecture, 536 MB over 16 layers, with dummy layers instead of real computation to measure network overhead.

measurements, code
Summary:
- PyTorch must use mpirun instead of distributed launcher (example)

- using standard ethernet layer gets 1.7-4.4 algbw for 1-16 rings (1 ring, 16 rings)
using EFA layer:
- 6.21 algbw over 2 machines (50% of theoretical max), run
- 4.53 algbw over 4 machines (54% of theoretical max), run
- maxing out bucket cap increases bandwidth 3.6->4.2
- single machine step time is 15ms. Two machine step time is 86ms.



Imagenet in 18 mins

Objective: Rerun imagenet in 18 minutes code on EFA network

1. Scaled up to 2 GPUs over 2 machines commit

2. some issues remain for scaling up further, https://github.com/pytorch/pytorch/issues/23721
measurements (TODO after issues are solved)

Comparison with on-prem Infiniband

From sjeaugey@nvidia.com
The peak for a 100Gb/s NIC is a bus bw of 12.5 GB/s.
Some numbers on DGX1 with NCCL_IB_HCA=mlx5_0 (to use only one IB EDR NIC) :
64 GPUs / 8 nodes : 11.87 GB/s
128 GPUs / 16 nodes : 11.71 GB/s

Translating into algorithmic bandwidth for 4GB transfers

Pre 0.3 build:

machines	on-prem	AWS	ratio
8	6.78	4.68	0.69
16	6.25	3.75*	0.60

* (3.75 algbw for 4GB transfer extrapolated from 3.57 observed on 1GB transfer)

Theoretical maximum alreduce algbw vs on-prem vs AWS for 100 Gbps network.

# of machines	optimal algbw GB/second	from @sjeaugey	EFA 0.3 build
2	12.5		7.71
4	8.33		
8	7.14	6.78	4.30
16	6.67	6.25	3.46
24	6.52		
32	6.45		0.98*

*run for earlier build observed 3.53, this could be unlucky machine allocation

Environment setup notes

Follow AWS instruction to build AWS OFI plugin and apply AWS-specific patch. Install Nvidia drivers on top of base Amazon Linux AMI. Build PyTorch and NCCL from scratch.

- https://github.com/cybertronai/aws-network-benchmarks/blob/efa/prepare_efa_image.py
- https://github.com/cybertronai/aws-network-benchmarks/blob/efa/indu_build.sh