SUPERCOMPUTING 24: DISTRIBUTED GPU COMPUTING BENCHMARK ANALYSIS

KEVIN WEISS

Abstract. This article presents the setup and results of experiments conducted during the Supercomputing24 (SC24) conference. We describe the hardware, software, and network configuration for two servers (Hubble and Voyager), as well as details on the distribution of a GPU workload from Voyager to Hubble. We compare the speed of computation using the Distributed package across wide area network (WAN) and local area network (LAN), as well as local execution. Our findings display the PyTorch Distributed package's ability to efficiently offload GPU workloads to remote servers using IPv6 for end-to-end connectivity, as well as some problems that arise when using a dual stack network.

Distributed GPU Computing, Computer Networking, SC24, Scientific Computing, Machine Learning, PyTorch

1. Introduction

Supercomputing 24 (SC24) is a leading conference showcasing advancements in high-performance computing (HPC). This article focuses on the setup and results of a computational experiment performed during SC24. In this demonstration, we utilize two servers. The servers used are Hubble, a Dell R730 located on the Foggy Bottom campus of The George Washington University (GW) in Washington, D.C., which serves as the remote server, and Voyager, another Dell R730 located on the conference floor in Atlanta, where the GPU workload is initialized. In the case of WAN we establish a high-performance network connection between the two servers by first linking Voyager to SciNet24, SC24's dedicated extreme-performance network. SciNet24 is then interfaced with Internet2, enabling connectivity to GW's Capital Area Advanced Research and Education Network (CAAREN), which provides the final link to Hubble. To distribute the workload, we utilize the Distributed package from the PyTorch library with Gloo as the backend. Native single stack IPv6 is used for end-to-end connectivity in all cases. For both local area and wide area tests, Hubble acts as the worker node while Voyager acts as the client.

1.1. Background.

Distributed computing has somewhat subtle but distinct differences from HPC. Distributed computing involves a network of computers that are often geographically dispersed across a wide area. HPC on the other hand refers to tightly coupled systems, usually called clusters, that are located in a single physical environment or data center with high-speed interconnects between the nodes. Distributed setups rely on networking protocols and middleware to coordinate tasks. One key advantage of distributed computing is scalability. While the addition of nodes in

an HPC cluster often requires careful consideration of interconnect speed and software parallelization, integrating additional machines into a distributed setup requires minimal changes to its architecture. Distributed systems are also inherently more fault-tolerant since they can continue functioning even if individual nodes fail [3]. HPC clusters focus on maximizing computational power and speed, while distributed computing emphasizes scalability, resource sharing, and workload distribution across multiple nodes to enhance efficiency and resilience.

Distributed computing systems show great promise for a number of applications including large-scale data processing, cloud computing, and real-time analytics [2]. This work focuses on distributing GPU workloads, and in particular the training of large neural networks. As the demand for computational resources required to train such models increases, developing both economically and environmentally conscious solutions is of great importance. We will show the feasibility of offloading GPU workloads (and subsequently training of large models) across wide areas using IPv6 for end-to-end connectivity and discuss the benefits and challenges of this approach.

2. Configuration

Here we outline the configuration for hardware, software, and network setup. For hardware we include the OS, distribution, kernel version, architecture, GPU, and network card. For software we outline the distributed setup, workload, environment, and backend. Network configuration includes circuit specifications, and routing details. Specifically, we detail the SciNet24 WAN circuits, the allocated 100GbE link from the California Institute of Technology, the Internet2 backbone, and the final connection to CAAREN, ensuring end-to-end IPv6 connectivity between Voyager and Hubble.

2.1. **Hardware Configuration.**

Tables 1 and 2 provide an overview of the hardware specifications for Voyager and Hubble. We chose Rocky 9.4 for its robust support of GPU-accelerated workloads in HPC. Hubble is equipped with an NVIDIA Tesla P100, offering 16GB of HBM2 memory and 4.7 teraflops of double-precision performance, making it ideal for deep learning in HPC. Notably, Voyager lacks a GPU.

2.1.1. *Hubble Hardware Overview.*

Component	**Specification**
Operating System (OS)	Linux
Distribution	Rocky 9.4
Kernel Version	5.14.0-427.40.1.el9_4.x86_64
Architecture	x86_64
GPU	NVIDIA Tesla P100
Network Card	Intel Ethernet Controller E810-C

Table 1. Hubble hardware specifications.

2.1.2. *Voyager Hardware Overvw.*

Component	Specification
Operating System (OS)	Linux
Distribution	Rocky 9.4
Kernel Version	5.14.0-427.40.1.el9_4.x86_64
Architecture	x86_64
GPU	None
Network Card	Mellanox ConnectX-5

TABLE 2. Voyager hardware specifications.

2.2. **Software Configuration.** The demonstration is run with Python 3.12.7 employing Torch 2.5.1+cu124. To offload the GPU workload, we utilize the Distributed package within the Torch library with Gloo as the backend.

2.2.1. *Environment.* The environment, which is identical for both servers, is set up with Conda and includes the following libraries:

- `numpy==2.1.3`
- `networkx==3.4.2`
- `sympy==1.13.1`
- `torch==2.5.1`
- `nvidia-cudnn-cu12==9.1.0.70`
- `nvidia-cublas-cu12==12.4.5.8`
- `nvidia-nccl-cu12==2.21.5`

2.2.2. *Workload.* The GPU workload consists of training ResNet34, a deep convolutional neural network with 34 layers that uses residual learning to improve training efficiency and accuracy in image classification tasks. We train the model on the CIFAR100 dataset. CIFAR100 is considered a difficult dataset with over 100 classes, and thus training requires significant computational resources. Training is run for 50 epochs.

2.2.3. *Backend.* In our setup Gloo serves as the backend for the PyTorch Distributed package. Gloo is a collective communication library designed to efficiently perform operations across distributed systems by implementing low level communication protocols designed specifically for distributed systems and machine learning frameworks such as PyTorch. While Gloo offers many advantages such as its minimalistic design, the primary reason for choosing it is its compatibility with hybrid CPU-GPU environments. This is particularly optimal in our case as we distribute a GPU workload from a non-GPU server.

2.3. **Network Configuration.** SciNet24 consists of 30 WAN circuits: 18 400GbE, 11 100GbE, and 1 10 GbE. The California Institute of Technology's booth hosts 6 100GbE circuits, one of which is allocated for our work.

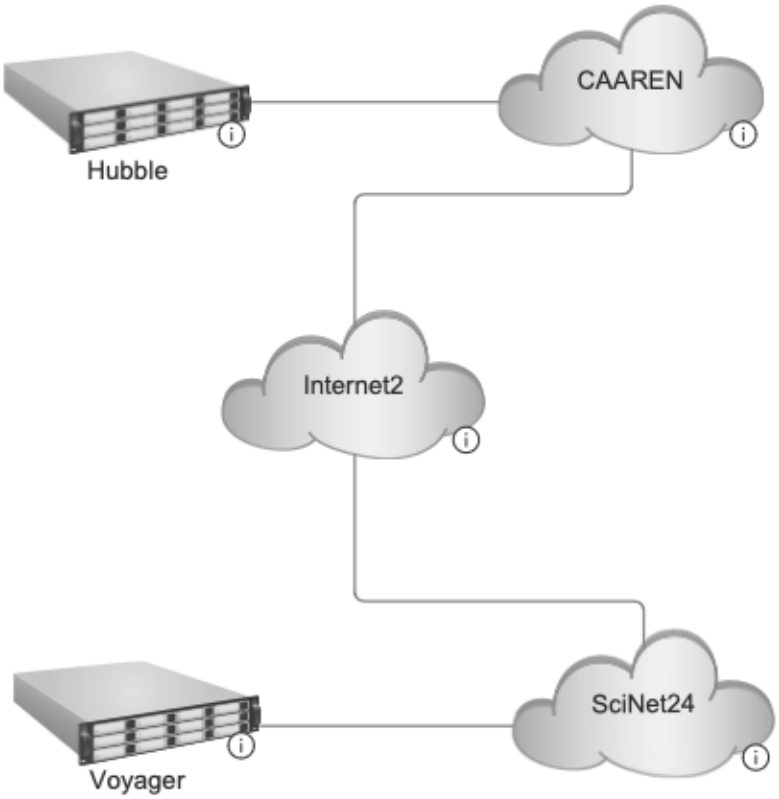

FIGURE 1. Network Diagram showing Voyager to Hubble connection.

Voyager is connected to SciNet24 through this circuit, which provides the next link to Internet2. Internet2 then links to CAAREN, allowing for the final hop to Hubble, with the entire network path utilizing IPv6 for end-to-end connectivity. This configuration is displayed in Figure 1.

3. RESULTS

We evaluate performance across three execution environments: local execution, local area network (LAN), and wide area network (WAN). The analysis focuses on the time required to train ResNet34 on the CIFAR100 dataset for 50 epochs, with additional bandwidth performance metrics reported for LAN and WAN configurations based on network throughput measurements. In addition to execution time, we also consider other factors such as network latency and overall data transfer efficiency. The execution times are shown in Table 3.

Method	Time (s)
Local Execution	2342.27
LAN Execution	2354.35
WAN Execution	2729.60

TABLE 3. Execution times for different environments.

3.1. **LAN Metrics.** To assess the network performance in the LAN environment, we conducted throughput measurements using `nuttcp`. `nuttcp` is a network throughput measurement tool that is used to assess the performance of TCP and UDP connections by generating traffic between two endpoints and reporting metrics such as throughput, latency, and packet loss. In all cases we evaluate TCP traffic. The results of the 10-second test are summarized below.

Metric	Value
Average Throughput	15 664.45 Mbps (15.66 Gbps)
Peak Throughput	16 484.53 Mbps (16.48 Gbps)
Total Data Transferred	18 697.25 MB
Total Retransmissions	486
Maximum Congestion Window (cwnd)	985 KB
Round-Trip Time (RTT)	0.28 ms
TX Utilization	31%
RX Utilization	99%

TABLE 4. LAN network performance metrics obtained via `nuttcp`.

3.1.1. *LAN Performance Discussion.* This data shows strong network performance with an average throughput of 15.66 Gbps over 10 s. A peak throughput of 16.48 Gbps was achieved during the first transmission. The total retransmissions (486) were primarily concentrated in the first and third seconds, likely due to transient packet loss. The congestion window (`cwnd`) remained stable at 985 KB, suggesting effective flow control. The low round-trip time (0.28 ms) is expected and a common trait of high-speed, low-latency networks. Furthermore, TX utilization at 31 % and RX utilization at 99 % indicate that the receiving server (Hubble) was nearly saturated, which aligns with the high throughput observed.

This high RX utilization is possibly a result of the Intel E810-C not being configured in a way that allows for maximum performance. Particularly, the E810-C requires PCIe 4.0 x 16 for maximal performance. Due to time constraints, we were unable to optimize to the latest version and maximum number of lanes. Suboptimal configurations, such as running the card on lower PCIe versions or using fewer lanes, could lead to bandwidth throttling ultimately affecting throughput.

3.2. **WAN Performance.** We evaluate WAN performance in a similar fashion to LAN, utilizing `nuttcp`. The WAN is studied with tests measuring traffic in two directions, both from Voyager to Hubble and from Hubble to Voyager.

3.2.1. *Voyager →Hubble.* Below in Table 5 we show WAN `nuttcp` data for traffic from Voyager to Hubble. We observe an average throughput of 2.69 Gbps. We note 336 total retransmissions with very low TX and RX utilization

Metric	**Value**
Average Throughput	2693.27 Mbps (2.69 Gbps)
Total Retransmissions	326
Congestion Window (cwnd)	6220 KB
Round-Trip Time (RTT)	15.12 ms
TX Utilization	4%
RX Utilization	7%

TABLE 5. Voyager to Hubble network performance metrics.

3.2.2. *Hubble →Voyager.* In Table 6 we show the `nuttcp` for test data from Hubble to Voyager. A strong average throughput of 26.66 Gbps is observed. TX Utilization is high at 98% and there are no retransmissions.

Metric	**Value**
Average Throughput	26 656.33 Mbps (26.66 Gbps)
Total Retransmissions	0
Congestion Window (cwnd)	126 785 KB
Round-Trip Time (RTT)	15.08 ms
TX Utilization	98%
RX Utilization	63%

TABLE 6. Hubble to Voyager network performance metrics.

3.2.3. *WAN Performance Discussion.* We see that traffic from Hubble to Voyager massively outperforms traffic in the opposite direction achieving roughly ten times the average throughput. It is unlikely this is due to asymmetric paths as we see virtually identical round trip times. However, we observe very low TX and RX Utilization in the underperforming direction (4% and 7% respectively), as well as a very small congestion window (6220 KB). This is often indicative of network congestion, which is plausible in our case given the network path is shared with other traffic on the conference floor. It is also possible SciNet24's architecture was designed to prioritize incoming traffic.

4. Discussion

4.1. **Performance.** We observe time to train is fastest for local execution, followed very closely by LAN. This suggests that network latency plays a minimal role in training performance over LAN. For WAN we see a 15.9% increase in time to train. This increase can be attributed to several factors, primarily network latency and bandwidth limitations. WAN can be significantly impacted by network conditions, and in some cases attribute to as much as 60% of a training iteration's time in distributed environments [4]. This highlights the high-performance nature of our connection established over WAN, as well as the feasibility of distributing GPU workloads across large geographic areas.

4.2. **RPC and Dual Stack Networks.** While network performance influences training efficiency, the way workloads are distributed also plays a critical role. Remote Procedure Call (RPC) is a framework within the PyTorch library that allows for the distribution of workloads across numerous worker nodes. During our experimentation we discovered a bug within this framework pertaining to the distribution of workloads via IPv6. On a dual stack network, RPC will attempt to communicate over IPv4 even when IPv6 is specified resulting in an error. This problem does not occur when using single stack native IPv6.

4.3. **Future Work.** With limitations on time, many methods of optimization were not fully explored. These methods include a more extensive analysis using `traceroute` to confirm symmetric paths for WAN, optimizing PCIe for LAN training, and investigating TCP congestion control and flow control mechanisms to improve send-side throughput. Additionally, tuning NIC settings such as interrupt moderation, buffer sizes, and offloading features could have mitigated potential bottlenecks, while alternative transport protocols like RDMA or QUIC might have offered further performance gains.

5. Conclusion

We observe time to train across LAN and local execution is virtually identical. This can be attributed to the high-performance nature of our LAN, providing minimal overhead and latency. We see a 15.9% increase in time to train across WAN, which demonstrates the practicality of distributing a GPU workload over large geographic areas. The ability to distribute these workloads efficiently was made possible in part by CAAREN, which facilitated seamless data transfer when interfaced with Internet2.

This experiment ultimately highlights the feasibility of training deep neural networks in a distributed environment utilizing single stack IPv6 for communication. This is particularly relevant as we see larger and larger data centers consume more and more energy. Small data centers can often achieve greater efficiency compared to large data centers due to factors like better localized cooling management, reduced power distribution losses, and the ability to optimize equipment layout for specific needs, allowing for more precise control over energy consumption [1]. By distributing

the workload across numerous small data centers significant improvements can be made in the overall energy cost of training large models.

6. Acknowledgments

We thank the SC24 organizing committee, our institutions, and funding agencies for their support. We also thank the fantastic team at GW's Research Technology Services, particularly Fong Banh, Andrew Gallo, Clark Gaylord, Rubeel Iqbal, Dr. Glen MacLachlan, Shashwitha Puttaswamy, and Dr. Adam Wong for both their support and involvement in the project. Additionally, we acknowledge the broader HPC community for their insights and collaboration, as well as the SciNet24 team for their hard work and dedication without which this demonstration could not have taken place.

References

[1] U.S. Department of Energy. Small data centers: Energy efficiency opportunities, 2024. Accessed: March 5, 2025.

[2] Iha Sharma. Distributed computing 101: How it effectively handles big data, 2024. Accessed: March 17, 2025.

[3] P. Tiwari and M. Sharma. Fault tolerance mechanisms in distributed systems. *International Journal of Computer Network and Information Security*, 7(6):1–11, 2015.

[4] Yunze Wei, Tianshuo Hu, Cong Liang, and Yong Cui. Communication optimization for distributed training: Architecture, advances, and opportunities. *arXiv preprint arXiv:2403.07585*, 2024.

The George Washington University, Washington D.C.
Email address: kevin.weiss1@gwu.edu